

**QUANTITATIVE ORGANIZATIONAL MODELING AND  
DESIGN FOR MULTI-AGENT SYSTEMS**

A Dissertation Presented

by

**BRYAN HORLING**

Submitted to the Graduate School of the  
University of Massachusetts Amherst in partial fulfillment  
of the requirements for the degree of

**DOCTOR OF PHILOSOPHY**

February 2006

Department of Computer Science

© Copyright by Bryan Horling 2006  
All Rights Reserved

# QUANTITATIVE ORGANIZATIONAL MODELING AND DESIGN FOR MULTI-AGENT SYSTEMS

A Dissertation Presented

by

BRYAN HORLING

Approved as to style and content by:

---

Victor Lesser, Chair

---

Brian Levine, Member

---

Shlomo Zilberstein, Member

---

Anna Nagurney, Member

---

W. Bruce Croft, Department Chair  
Department of Computer Science

*To my wife and partner, Maura.*

## ACKNOWLEDGMENTS

This dissertation would not have been possible without the help, support and guidance from many people. No single person has influenced this process more than my adviser, Victor Lesser. Your encouragement, suggestions, comments and critiques have helped shape my work, and without your persistence I would not have even had the opportunity to do so.

I would like to thank the members of my committee, Brian Levine, Anna Nagurney and Shlomo Zilberstein, for all their hard work. Your questions helped guide me towards interesting and challenging paths, and our discussions helped frame the broader context of this work.

My family has been my foundation throughout my graduate career, giving me the opportunity and encouragement I needed to see it through. My wife and partner Maura deserves the most praise, as the time I needed to do my work frequently required sacrifice on her part. Your emotional support, patience, and willingness to listen as I ramble on about my research were all invaluable. I would like to thank Megan as well, for giving me the opportunity to get some writing done during your Monday afternoon naps, for only throwing my papers on the floor once, and for being there to play with when I didn't feel like working.

My mother, father and sister have supported me for more than thirty years now, and I clearly would not be where I am today without their help. There are too many points along the way that could be mentioned, but most of all it was your belief in me that provided the motivation I needed to see me through.

I would like to thank the Fennellys - Paul, Kathy, Jeff, Eileen and Kevin, for their encouragement, and for providing a place of refuge for Maura and I when one or both of us needed a break.

Over the course of the last two years I had the opportunity to speak with many individuals about my work, and benefited greatly from these interactions. In no particular order I would like to thank Dan Corkill, Roger Mailler, Anita Raja, Sherief Abdallah, Mark Sims, Mark Fox, Neil Immerman, Les Gasser, Jim Kurose, Haizeng Zhang, Jiaying Shen, Raphen Becker, Katia Sycara, Frank Dignum, Virginia Dignum, Carl Hewitt and Tom Wagner for those conversations.

I would like to thank Michele Roberts, without whom papers and forms would not reach their destinations, salaries would not be paid, and life in the lab would be generally less pleasant. I would also like to thank the past and present members of the Multi-Agent Systems Lab game night, which helped provide a needed distraction almost every week: Sherief Abdallah, Peter Amstutz, Mike Atighetchi, Raphen Becker, Brett Benyo, Ross Fairgrieve, Roger Mailler, Stephen Murtagh, Dan Neimann, Shichao Ou, Rodion Podorozhny, Kyle Rawlins, Jiaying Shen, Regis Vincent, Tom Wagner, and Ping Xuan. I have also benefited from my interactions with

the many other members of the MAS lab, including Ana Bazzan, Andrew Fast, Nadia Ghamrawi, AnYuan Guo, Frank Klassner, Hala Mostafa, Mike O’Neill, John Ostwald, Anita Raja, Zach Rubinstein, Haizheng Zhang, and Shelley Zhang.

I would like to single out Regis Vincent, Roger Mailler, Raphen Becker, Jiaying Shen and Kyle Rawlins for their help in designing and implementing the distributed sensor network platform analyzed in this work. I would also like to thank Haizheng Zhang for his research in information retrieval, which this work also exploits.

The fine people at Freeverse Software (Ian, Colin & Steve) deserve thanks for giving me a creative outlet. Thanks also go to the Amherst College Computing Center and the authors of the JEP and JDOM software libraries.

## ABSTRACT

# QUANTITATIVE ORGANIZATIONAL MODELING AND DESIGN FOR MULTI-AGENT SYSTEMS

FEBRUARY 2006

BRYAN HORLING

B.Sc., TRINITY COLLEGE

M.Sc., UNIVERSITY OF MASSACHUSETTS

Ph.D., UNIVERSITY OF MASSACHUSETTS AMHERST

Directed by: Professor Victor Lesser

As the scale and scope of distributed and multi-agent systems grow, it becomes increasingly important to design and manage the participants' interactions. The potential for bottlenecks, intractably large sets of coordination partners, and shared bounded resources can make individual and high-level goals difficult to achieve. To address these problems, many large systems employ an additional layer of structuring, known as an organizational design, that assigns agents particular and different roles, responsibilities and peers. These additional constraints can allow agents to operate effectively within a large-scale system, with little or no sacrifice in utility. Different designs applied to the same problem will have different performance characteristics, therefore it is important to understand and model the behavior of candidate designs.

In the multi-agent systems community, relatively little attention has been paid to understanding and comparing organizations at a quantitative level. In this thesis, I show that it is possible to develop such an understanding, and in particular I show how quantitative information can form the basis of a predictive, proscriptive organizational model. This can in turn lead to more efficient, robust and context-sensitive systems by increasing the level of detail at which competing organizational designs are evaluated.

To accomplish this, I introduce a new, domain-independent organizational design representation able to model and predict the quantitative performance characteristics of agent organizations. This representation, capable of capturing a wide range of multi-agent characteristics in a single, succinct model, supports the selection of an

appropriate design given a particular operational context. I demonstrate the representational capabilities and efficacy of the language by comparing a range of metrics predicted by detailed models of a distributed sensor network and information retrieval system to empirical results. In addition to their predictive ability, these same models also describe the range of possible organizations in those domains. I show how general search techniques can be used to explore this space, using those quantitative predictions to evaluate alternatives and enable automated organizational design.



# TABLE OF CONTENTS

	Page
<b>ACKNOWLEDGMENTS</b> .....	<b>v</b>
<b>ABSTRACT</b> .....	<b>vii</b>
<b>LIST OF TABLES</b> .....	<b>xiii</b>
<b>LIST OF FIGURES</b> .....	<b>xiv</b>
 <b>CHAPTER</b>	
<b>1. INTRODUCTION</b> .....	<b>1</b>
1.1 Introduction .....	1
1.2 Major Ideas .....	7
1.2.1 Basic Assumptions .....	7
1.2.2 Organizational Design .....	8
1.2.3 Representing Organizations .....	10
1.3 Guide to the Dissertation .....	11
<b>2. REPRESENTING ORGANIZATIONS</b> .....	<b>13</b>
2.1 Organizational Effects .....	13
2.1.1 The Distributed Sensor Network Domain .....	13
2.1.2 Empirical Demonstration of Organizational Effects .....	17
2.1.3 Geographic Coalitions .....	18
2.1.4 Functional Differentiation .....	19
2.1.5 Organizational Maintenance .....	20
2.1.6 Generality of Effects .....	22
2.2 Organizational Representation .....	24
2.2.1 ODML .....	25
2.2.2 Distributed Sensor Network Model .....	31
2.2.2.1 Has-A Relationships .....	32
2.2.2.2 Is-A Relationships .....	34
2.2.2.3 Templates and Instances .....	35
2.2.2.4 Quantitative Expressions .....	36
2.2.3 Supported Data Types .....	38

2.2.3.1	Numeric Data .....	39
2.2.3.2	Lists .....	39
2.2.3.3	Distributions .....	39
2.2.4	Limitations and Tradeoffs .....	42
2.3	Modeling Characteristics in the DSN Domain .....	44
2.3.1	Environmental and System Constants .....	44
2.3.2	Local Characteristics .....	45
2.3.3	Entity Interactions .....	46
2.3.4	Multiple Role Assignments .....	47
2.3.5	Dynamic Behaviors .....	49
2.3.6	Heterogeneity .....	50
2.3.7	Conflicts, Constraints and Resolution .....	50
2.3.8	Organizational Utility .....	52
2.3.9	Evaluation of Representation .....	54
2.4	Conclusions .....	57
<b>3.</b>	<b>MODELING AN INFORMATION RETRIEVAL SYSTEM .....</b>	<b>59</b>
3.1	Information Retrieval Model .....	59
3.2	IR Simulation .....	63
3.3	Representing IR Characteristics .....	64
3.3.1	Roles .....	64
3.3.2	Data Sources and Collection Signatures .....	64
3.3.3	Probabilistic Search and Query .....	65
3.3.4	Query Response Time .....	69
3.3.5	Constraints .....	78
3.3.6	Organizational Utility .....	78
3.4	Conclusions .....	84
<b>4.</b>	<b>DESIGNING ORGANIZATIONS .....</b>	<b>85</b>
4.1	Designing Organizations .....	86
4.1.1	Design Complexity .....	87
4.2	Algorithmic Search Techniques .....	92
4.2.1	Exploiting Hard Constraints .....	93
4.2.1.1	Monotonic Trends in the DSN Model .....	97
4.2.1.2	Non-monotonic Trends in a SUBSET-SUM Model .....	100
4.2.1.3	Results .....	101
4.2.2	Equivalence Classes .....	102
4.2.2.1	Results .....	104
4.2.2.2	Selecting Appropriate Discriminators .....	105
4.2.3	Using General Mathematical Solvers .....	106
4.2.4	Avoiding Redundant Search .....	110
4.2.5	Independent Sub-Problems .....	112
4.2.6	Domain-Specific Techniques .....	112
4.3	Heuristic Modeling Techniques .....	113

4.3.1	Homogeneity .....	114
4.3.2	Abstraction .....	114
4.3.3	Heuristic Modeling Results .....	116
4.4	Designing Organizations in Practice .....	118
4.4.1	Efficiently Evaluating Organizations .....	119
4.4.2	Constructing Organizations .....	121
4.4.2.1	Top-Down Construction .....	121
4.4.2.2	Bottom-Up Construction .....	125
4.4.3	Applying Designs to Actual Systems .....	125
4.4.4	Model Inheritance and Reuse .....	127
4.5	Adapting Organizations .....	128
4.5.1	Monitoring for Problems .....	128
4.5.2	Searching for Solutions .....	130
4.6	Conclusions .....	131
<b>5.</b>	<b>MODELING OTHER CHARACTERISTICS AND PARADIGMS .....</b>	<b>133</b>
5.1	Modeling Common Organizational Characteristics .....	133
5.1.1	Non-Uniformity .....	133
5.1.2	Temporal Interactions .....	138
5.1.3	Levels of Abstraction .....	140
5.2	Modeling Common Organizational Paradigms .....	140
5.2.1	Hierarchies and Holarchies .....	141
5.2.2	Coalitions and Congregations .....	143
5.2.3	Federations .....	145
5.2.4	Markets .....	150
5.2.5	Matrix Organizations .....	152
5.2.6	Societies .....	154
5.2.7	Teams .....	156
5.2.8	Compound Organizations .....	159
5.3	Conclusions .....	160
<b>6.</b>	<b>RELATED WORK .....</b>	<b>161</b>
6.1	Organizational Representations .....	161
6.2	Organizational Evaluation and Design .....	169
<b>7.</b>	<b>CONCLUSIONS .....</b>	<b>175</b>
7.1	Summary .....	175
7.2	Contributions .....	178
7.3	Discussion .....	181
7.4	Future Directions .....	183

## APPENDICES

<b>A. TRANSLATING ODML TO MATHEMATICA</b> .....	<b>187</b>
<b>B. A SURVEY OF MULTI-AGENT ORGANIZATIONAL PARADIGMS</b> .....	<b>190</b>
B.1 Hierarchies .....	191
B.2 Holarchies .....	194
B.3 Coalitions .....	197
B.4 Teams .....	200
B.5 Congregations .....	203
B.6 Societies .....	206
B.7 Federations .....	210
B.8 Markets .....	212
B.9 Matrix Organizations .....	216
B.10 Compound Organizations .....	218
B.11 Other Organizational Topics .....	221
B.12 Discussion .....	223
<b>C. DISTRIBUTED SENSOR NETWORK ODML MODEL</b> .....	<b>227</b>
<b>D. INFORMATION RETRIEVAL ODML MODEL</b> .....	<b>235</b>
<b>E. TILING REDUCTION MODEL</b> .....	<b>244</b>
<b>F. SUBSET-SUM REDUCTION MODEL</b> .....	<b>246</b>
<b>BIBLIOGRAPHY</b> .....	<b>247</b>

## LIST OF TABLES

Table	Page
2.1 ODML's built-in functional operators. By convention, $x$ and $y$ are simple numeric values, $\bar{x}$ is a list, and $D$ is a distribution. ....	36
2.2 ODML's built-in data types. ....	39
3.1 The relative error (i.e., $(observed - predicted)/observed$ ) between the predicted and empirical response recall values from Figure 3.4. ....	69
4.1 Results from organizational search in small-scale information retrieval templates. Number of agents and utility are given for the optimal found organization. ....	116
4.2 Results from organizational search in large-scale information retrieval templates. Number of agents and utility are given for the optimal found organization. ....	118
6.1 A comparison of the characteristics and capabilities of several different organizational representations. ....	169
6.2 A comparison of the characteristics and capabilities of several different organizational design and adaptation schemes. ....	174

## LIST OF FIGURES

Figure	Page
1.1 Example ODML (a) template and (b) instance structures for a sensor network organization. ....	4
2.1 Organization-centric view of the DSN architecture. ....	14
2.2 The DSN architecture in four phases. A: sectorization of the environment, B: distribution of the scan schedule, C: negotiation over tracking measurements, D: tracking data fusion. ....	15
2.3 Effect of sector size on messaging. ....	18
2.4 Messaging disparity vs. sector size. ....	19
2.5 Message types vs. sector size. ....	20
2.6 Effect of sector size on RMS error. ....	21
2.7 Average communication distance. ....	21
2.8 Communication disparity (a) and RMS error differences (b) with varied sector sizes and target densities. ....	23
2.9 Pseudocode for the <code>get_value</code> function of a node $N$ . This function is used to quantify the characteristics of instance nodes. ....	30
2.10 Example ODML (a) template and (b) instance structures for the sensor network organization. ....	32

2.11	A portion of the ODML specification for the <i>track_manager</i> and <i>s_tm_relation</i> nodes. The structural has-a relations are defined in <i>track_manager</i> first, using an estimate of the role's migration rate and the number of extant <i>sensor</i> and <i>sector_manager</i> nodes. An estimate of the target bounds is expressed next, and that bound used with the environmental sensor density to estimate the number of sensors that will be contacted for data. The <i>sm_tm_relation</i> shows how modifiers are used to first propagate that demand to a <i>sensor</i> , and later to inform the track manager of the the resulting actual measurement rate. . . . .	33
2.12	An ODML specification for the agent types in the DSN domain showing how is-a relationships allow characteristics to be shared among derived nodes, and how those derived nodes can be differentiated through their additional local definitions. . . . .	35
2.13	A graph showing some of the equations and equation-based interactions between nodes that are used to predict RMS tracking error in the DSN model. Solid arrows represent has-a relationships, while dashed arrows between nodes indicate where a modifier is used to propagate values between nodes. . . . .	37
2.14	Pseudocode for the hierarchical Monte Carlo trial procedure. . . . .	42
2.15	A portion of the ODML specification for the sensor node showing how message load values, number of sensors controlled and role indicators are passed to the <i>agent</i> with modifiers. . . . .	48
2.16	A portion of the ODML specification that estimates the delay in directory responses that is experienced by a track manager. The <i>sm_tm_relation</i> nodes determine the individual delays for each sector manager, while the <i>track_manager</i> uses these values to estimate a worst case delay that is used to reduce the requested measurement rate. . . . .	51
2.17	Performance predicted by the ODML sensor network model versus empirical observations for a) Total messaging, b) Messaging disparity, c) Message type totals and d) RMS error. Predicted lines are solid, empirical are dashed. . . . .	54
2.18	A comparison of the average actual and model-predicted characteristics by role, for agents operating in the distributed sensor network. . . . .	55

2.19	Performance predicted by the ODML model as the number of targets is varied for a) Messaging disparity, and b) RMS error. . . . .	56
3.1	The control and communication sequence involved in handling a query in the information retrieval organization. Steps in the numbered trace correspond to messaging events, except for bracketed elements that indicate local processing. . . . .	60
3.2	a) An ODML template structure for the information retrieval domain. b) A small organizational instance produced from that template. . . . .	62
3.3	An information retrieval instance with variously ranked mediators. . . . .	67
3.4	A comparison of the predicted and empirical response recall values across a range of search and query size parameters. . . . .	68
3.5	A portion of the ODML specification for the <i>mediator</i> and <i>database</i> nodes detailing the fields used to estimate response time. . . . .	72
3.6	Sample (a) pdf and (b) cdf profiles for the database waiting time distribution. . . . .	73
3.7	A comparison of the waiting time distributions for differently sized sets of databases. . . . .	74
3.8	A comparison of the response time distributions predicted by the ODML model and observed in organizations with (a,b) five [1M,4D], (c,d) fifteen [1M,6A,8D], and (e,f) twenty-eight [1M,7A,20D] agents. In the designs shown in (a,c,e), node M is the mediator, A are aggregators, and D are databases. . . . .	77
3.9	The utility predicted for the range of possible six-database organizations when the query rate (queries per second) is varied. Mediators and aggregators are shown as hollow circles, while the solid databases form the leaves. Higher is better, optimal values for each rate are shown in bold. . . . .	80
3.10	Similar to Figure 3.9, the utility predicted for the range of possible six-database organizations when the query rate (queries per second) is varied, but with the query and search sizes set to one. Note differences in organizations 11-18. . . . .	82



3.11	The range of optimal organization instances derived from the IR template, when 1,3,6,9,12 or 15 databases are available. The <i>environment</i> and <i>organization</i> nodes have been omitted for clarity. ....	83
3.12	A comparison of the waiting time distributions for the different optimal organizations from Figure 3.11. The distributions widen and shift right as the number of databases increase. ....	84
4.1	A sample TILING problem and consistent solution. ....	90
4.2	(top) An example ODML template used to reduce a TILING problem. (bottom) A valid organizational instance created from that template.....	91
4.3	The first eight organizational decisions made during a search of the DSN organizational template. ....	93
4.4	Pseudocode for the constraint satisfiability and trend estimation procedures. ....	96
4.5	Partial dependency graph for the distributed sensor network model. ....	98
4.6	Dependency graph for the SUBSET-SUM example. ....	100
4.7	A comparison of the search time differences with and without the constraint estimation algorithm across three different models.....	101
4.8	A comparison of the number of alternatives that are considered with and without equivalence classes across four differently sized sensor networks. The models allowed designs with 3, 6, 9 and 12 total sensors, respectively. ....	104
4.9	Three stages from the search process using an external optimization engine. The original template (a), the instance with deferred variables (b), and the final organizational instance (c).....	109
4.10	The result of the ODML translation process. This code is passed into Mathematica to perform the optimization. ....	110
4.11	Performance of the numeric solver versus conventional search across different problem complexities. ....	111

4.12	Two information retrieval templates, derived from Figure 3.2a. a) Incorporates homogeneity, by limiting aggregator selection to two distinct choices. b) Incorporates abstraction, by eliminating the assignment of roles to distinct agents. ....	113
4.13	The optimal organization found by the baseline template for the small-scale scenario.....	117
4.14	Optimal organization instances produced by the Homogeneous + Abstract (a) and (b) templates for the small-scale scenario.....	117
4.15	Search performance with and without cached values. ....	120
4.16	The partitioning of a local search tree. Strikeouts indicate visited choices, bolded are the current choice, while a question mark indicates additional unknown choices remain. ....	123
4.17	Characteristics of the distributed search using different numbers of processing agents. The speedup factor is shown in (a), and the number of messages required in (b). ....	124
4.18	Reusing common agent models in new domains. ....	127
5.1	A depiction of the revised DSN template. Important modifications from the original DSN model are shown in the shadowed boxes.....	134
5.2	The sigmoid function used to calculate sensor demand in the DSN model. This reflects the sensor's range of approximately 30 feet. ....	135
5.3	A depiction of the performance predictions by the enhanced DSN model using non-uniform target paths (upper) and sensor locations (lower). The layouts (a,c) show the sensor and target arrangements for two different scenarios. Sensors are circles, targets are triangles and their paths are dashed lines. The graphs (b,d) show the demand levels by sensor (top, darker is target 0, lighter is target 1) and the measurement levels by track manager (bottom, darker is the amount received, lighter is the additional amount that was requested but not able to be satisfied). ....	137
5.4	Performance predictions by the enhanced DSN model, demonstrating the effects that shifting starting points have on performance. Legend is the same as Figure 5.3. ....	139

5.5	A range of modeling possibilities, each with different levels of abstraction. . . . .	140
5.6	Using a hierarchy template in the IR domain. The design shown in Figure 3.2b shows a possible corresponding instance. . . . .	141
5.7	An ODML holarchy a) template and b) example instance. The structure is quite similar to the hierarchy in Figure B.1, differences lie primarily between their internal behaviors. . . . .	142
5.8	An ODML coalition a) template and b) example instance. . . . .	143
5.9	An ODML federation a) template and b) example instance. A federation consist of a set of members that have relinquished control to a distinguished <i>intermediary</i> . The <i>broker</i> and <i>reflector</i> nodes represent two possible intermediaries. Only intermediaries interact between federations, as represented by the links through <i>relationship</i> nodes in b). . . . .	146
5.10	An alternate ODML federation template that allows greater differentiation among intermediary styles but is less able to inherit and reuse common elements. . . . .	149
5.11	An ODML market a) template and b) example instance. . . . .	150
5.12	An ODML matrix a) template and b) example instance. The <i>wm-relation</i> node is used to propagate the effects of each manager to a worker. . . . .	153
5.13	An ODML team a) template and b) example instance. Has-a relationships from <i>teams</i> to <i>goals</i> are used to allocate work among the entities. . . . .	157
5.14	Portions of an ODML team model supporting goal alternatives. . . . .	158
A.1	ODML model element to Mathematica translation table. . . . .	187
A.2	ODML built-in function to Mathematica translation table. . . . .	189
B.1	A hierarchical organization. . . . .	191
B.2	A holarchical organization. . . . .	194

B.3	A coalition-based organization. ....	197
B.4	A team-based organization. ....	200
B.5	Congregations of agents. ....	203
B.6	An agent society. ....	206
B.7	An agent federation. ....	210
B.8	A multi-agent marketplace. ....	212
B.9	A matrix organization. ....	216
B.10	A compound organization. ....	218
B.11	Comparing the qualities of various organization paradigms. ....	224
C.1	Graphical view of the ODML DNS model. ....	228
D.1	Graphical view of the ODML information retrieval model. ....	236
D.2	Graphical view of the complete ODML information retrieval model obtained from the source in this appendix. ....	237

# CHAPTER 1

## INTRODUCTION

### 1.1 Introduction

Many of the decisions made in multi-agent system design, and in computational systems in general, are predicated on the idea that one wishes to minimize the “bad” characteristics of the system while maximizing the “good”. This practice manifests itself in blanket, axiomatic objectives such as “minimizing communication”, “reducing uncertainty”, and “maximizing profit”. While these are worthy, abstract goals that have critical practical and research importance, when a system is deployed and situated in context, such ideals may no longer have the same level of relevance. Consider the underlying issues that drive these objectives. Why should communication be minimized? Why do we care about the combinatorics of a particular technique? Why should centralization be avoided? In each case, we presume the existence of some limiting factor, some bounded resource that motivates these objectives. However, when the system is placed in a particular context where these bounds can be quantified, the intangible nature of these blanket statements is no longer sufficient. For example, if ample communication bandwidth is available and additional utility may be derived by using it, then a strategy that always minimizes communication may lead to a solution that fails to reach its potential. If a particular resource is bounded, and the qualitative side effects of using some or all of that resource are the same, then the system should exploit it as best it can in service of satisfying or maximizing the system’s specified goals.

Because of this, I believe that any real-world system must be tailored to the environment in which it exists, if it is to make effective use of the resources and flexibility available to it. I explore this tailoring through the system’s *organizational design*. The notion of an organizational design is used in many different fields, and generally refers to how members of a society act and relate with one another. This is true of multi-agent systems, where the organizational design of a system can include a description of what types of agents exist in the environment, what roles they take on, and specifications guiding how they act both independently and with one another. More generally, if we assume an entity has a set of possible choices to make during its operation, the organizational design will identify a particular subset of those choices that should actually be considered at runtime. By working with this typically smaller set, the entity’s decision process is facilitated. This additional structure becomes increasingly important as the system scales in number and scope [32]. Imagine how difficult it would be for a large human organization, such as a corporation

or government, to function if individuals lacked job descriptions and long-term peer relationships. Agent systems face similar challenges, and can derive similar benefits from an explicit organizational design.

Consider the problem of designing a solution for a complex, resource-bounded domain, such as a distributed network of sensors that is used for tracking. Such systems typically consist of an array of sensor nodes that are deployed to obtain the measurement data needed to track mobile targets in an environment. Assume in this case that each sensor is host to a local process called an *agent* that is responsible for controlling the sensor. Let us further assume that the sensor nodes must collaborate in some way to be successful, because multiple sensors must illuminate a target simultaneously to correctly obtain its position. Given these assumptions, a designer must determine a way to structure the agents' behaviors so that tracking may be accomplished. One strategy would create or delegate a single agent to be the *manager* of the entire sensor network. The manager would decide when, where and how each sensor should take measurements, and then process the resulting data to estimate the targets' positions. This layout of responsibilities constitutes a rudimentary organizational design. It specifies what roles agents take on, who they interact with, and where decision making authority is located.

Under some conditions, this simple solution will perform optimally, because the manager can maintain an omniscient view of the entire network's state and use that view to find the best assignment of sensing tasks. However, under real world conditions, where bandwidth and computational power is limited, communication and data processing takes time, and the number of sensors can be arbitrarily large, the weaknesses of this approach quickly become apparent. A different strategy, in the form of a different organizational design, can compensate for these more challenging conditions. For example, we might distribute the manager role among multiple agents to more evenly balance the communication and computational loads. We might also create an information dissemination hierarchy among the agents that prioritizes, summarizes, and propagates measurement data to use the available bandwidth more efficiently. However, distributing the role can lead to conflicts among managers and lower utility assignments, because no single agent necessarily has the local context to make the right decision. Similarly, the summarization process of a hierarchical distribution scheme can introduce additional latency and imprecision. Because of these tradeoffs, the organization can be a double-edged sword, both helping and hindering the system in potentially complex ways. The questions I address in this thesis revolve around finding a general way to determine the most appropriate organizational strategy for a given situation when there are many such strategies to consider.

Implicit in this example is the idea that different organizations will affect the performance of a working system in different ways. Intuitively, changing the manner in which agents interact or the pattern that those interactions take on can change how the system behaves from both global and local perspectives. The objectives of a particular design will depend on the desired solution characteristics, so for different problems one might specify organizations which aim toward scalability, reliability, speed, or efficiency, among other things. Confounding the search for such a design

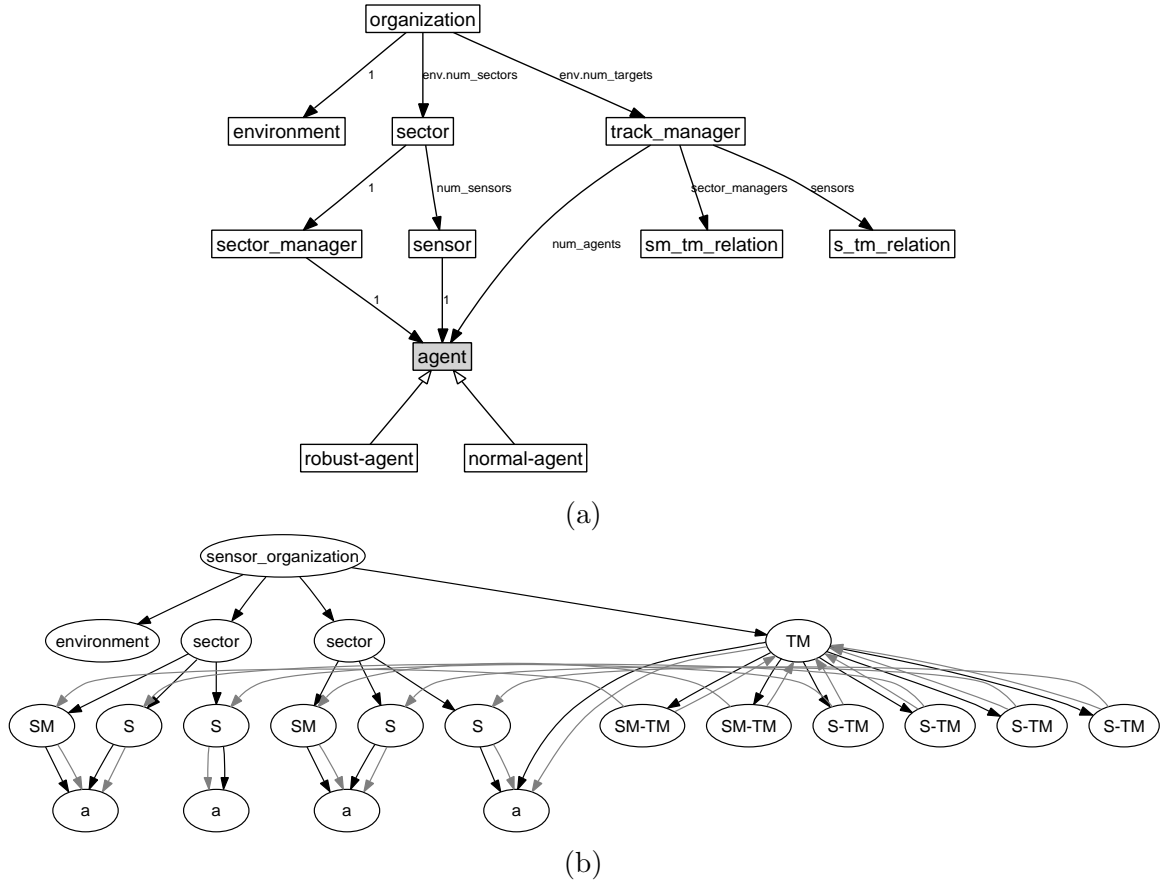
is the fact that many potentially important characteristics can be subtle, not readily identified as the system is being developed, or have complex interactions.

For example, at what point do the benefits of the dissemination hierarchy proposed above outweigh its costs? The additional communication and processing resources required to implement it may not be readily available. Obtaining them may require a monetary cost if new systems must be purchased, or a complexity cost if the new responsibilities are spread among the existing systems. At the same time, one must reason about the dimensions of the hierarchy – how tall and wide should it be? Which entities should be assigned the responsibilities present at each node? Should the tree dimensions be kept small, potentially concentrating the burden, or be made large to more evenly distribute the load? The designer will likely have an intuitive grasp of what is required, which is how existing systems are typically developed. However, all of these features are interrelated along with the goals of the system, the expected tasks it will experience, and the nature of the available resources. Intuition can fall short when such interactions allow small changes to lead to unexpected outcomes. These so-called phase transitions or tipping points require a deeper understanding and a more concrete representation to be addressed.

Although individuals have created mathematical models for particular aspects of organizationally-driven agent behaviors [166, 123, 163, 39, 68, 160], none have explored the utility of a general modeling language capable of incorporating arbitrary quantitative information. It is my belief that understanding the fundamental causes of characteristics like those described above, and using that information to develop accurate, predictive models of their effects are both critical to selecting an appropriate design, particularly as the agent population grows in scale or complexity. If we are to understand these effects and develop the means by which they can be exploited or avoided through organizational design, we must have a representation capable of expressing the range of ways the design can be created and capturing the characteristics each design will exhibit.

Many different representations have been created to describe agent organizations [187, 40, 109, 143, 45, 171, 50, 59, 122, 88, 174]. Most fall into one of two categories: either they represent a wide range of organizational characteristics abstractly, or they can capture a smaller set of characteristics concretely. The former are usually good at representing what entities exist or could exist, but cannot compare alternatives in a quantitative way. The latter may contain quantitative knowledge, but have difficulty relating that knowledge to specific organizational concepts, mitigating their usefulness if one is hoping to understand the effects a particular organizational design will have.

More specifically, existing organizational representations are either flexible and qualitative or inflexible and quantitative. In this work I demonstrate that it is possible to create a representation that is both flexible and quantitative. I introduce a new representation, the Organizational Design Modeling Language (ODML), designed to capture organizational information in a single unified, predictive structure. This representation, described in detail in Chapter 2 has the capability to model a wide range of organizational paradigms and characteristics, at different levels of abstraction across many different domains. At the same time, it is able to integrate concrete numeric information in the form of expressions and predictive equations using a range



**Figure 1.1.** Example ODML (a) template and (b) instance structures for a sensor network organization.

of mathematical techniques. Using this representation, it is possible to create a wide range of integrated models that possess a level of quantitative detail that is not possible with existing languages.

ODML models are superficially graph-based structures that consist of nodes and edges. Figure 1.1a shows an example ODML template model from a more detailed distributed sensor network architecture that is introduced in Chapter 2. Nodes in the graph are used to represent organizational components, while edges are used to represent the relationship between those components. In this example, there are nodes that correspond to agents, manager roles, the operating environment, as well as more abstract organizational constructs. Edges correspond to relational characteristics, such as the fact that roles are bound to agents, that sectors contain sector managers and sectors, and that robust-agent and normal-agent are both types of agents.

This graph representation depicts only the structural aspects of the organization. Of equal importance are the quantitative details that exist within each node, which are not shown in Figure 1.1. Each node contains a set of fields that quantitatively describe the relevant characteristics of the node using mathematical expressions. These



expressions can affect or be affected by the characteristics of other nodes, implicitly creating a second, more detailed layer of relationships through which one aspect of the organization may affect another. In this way, quantitative information about the environment, resources, agents, tasks, goals, or other components relevant to the system's performance can be incorporated into a single model and tied together in a way that captures the interdependencies that exist between them. This web of equations is a key component of ODML that differentiates it from existing representations.

As mentioned above, understanding the quantitative effects of organization is a necessary prelude to determining which design is most appropriate for a given operational context. ODML's capacity to represent detailed numeric information therefore gives it part of the functionality needed to address this problem. To provide the remaining requisite functionality, it must also be able to express the variety of ways the design can be manifested. This is accomplished in ODML through the creation of a template (Figure 1.1a) that has decision points embedded within it. Different choices for those decisions will result in different candidate designs. It is in this way that the template embodies the space of organizational alternatives. Given a template and a set of choices for those decision points, one can create a particular instance of that organization (Figure 1.1b). In that example two sectors have been created, each with a sector manager (SM) and two sensors (S). A single track manager (TM) is connected to these entities through a series of relationships. By using the embedded expressions in the instance model to relate, predict and evaluate its characteristics, the instance may be automatically compared and ranked against other competing designs. This ranking is then used as part of a search process to select the most appropriate design.

It is during this evaluation and ranking phase that the web of equations is used most. Other representations typically perform their evaluation using a fixed set of characteristics limited by the language, through simulations or model-specific heuristic analyses, or through more qualitative or logical comparisons. ODML is differentiated by the fact that one can embed arbitrary mathematical expressions within the model, and use those to produce fast, precise predictions of whatever characteristics are deemed relevant to evaluating design utility.

The flexibility of the ODML representation, its ability to model a wide range of concepts and functionality, is derived from the nature of the language itself. Nearly all existing organizational representations are structured around a well-defined set of required or permissible structures. For example, they will have concrete and explicit notions of an agent, a role, norms or goals. These concepts can be represented in ODML, but this representation is accomplished using only the primitive notions of node, relationship and quantitative characteristics outlined above; they have no pre-defined semantics.

Although having such built-in structures can be beneficial, their existence, particularly if they are required, means that any model created with such a language must abide by the assumptions associated with those structures. These assumptions can be sufficiently constraining or inflexible that the representation is no longer usable, or that the accuracy of the resulting model is compromised. For example, a proposed organization may be large enough that one would not want to have an explicit no-

tion each individual agent, because to do so would result in a model so large as to be impractical. In other cases the agent characteristics supported by the language may be insufficient or inappropriate to capture the nature of the domain in question. Because ODML makes no such assumptions, the first designer could choose to omit an agent node, creating a more abstract but correspondingly more scalable model. The second designer has the freedom to incorporate only those characteristics they deem appropriate. ODML's relatively primitive language structure is another characteristic that differentiates it. I will demonstrate how the flexibility this primitive nature provides allows it to model a range of detailed concepts not easily captured in other representations.

The drawback to having a language lacking in high-level concepts is that it makes the search process more difficult. For example, in languages that explicitly support agents and roles it is possible to embed into the search process the idea that roles must be bound to agents, and create heuristics or strategies designed to take advantage of that special relationship (e.g., [174, 159, 138, 35]). Lacking such structural landmarks, a search process can only attempt to infer that this is the case. Given that such a relationship might be modeled in different ways by different designers (or omitted altogether as above), the applicability of such concept-specific techniques is limited.

Because of this, and because the search space created by an ODML template can be quite large, solving the search problem is difficult. In Chapter 4 I demonstrate this difficulty by proving that the problem of finding a valid instance within an ODML template space is NEXP-complete. That chapter also describes a range of techniques that can be used to exploit the mathematical nature of the representation and make the search more efficient. In particular, I show it is possible to exploit constraints embedded in the model to bound the search space, and to devise notions of choice equivalence to avoid redundant parts of the space. Both techniques can result in a significant reduction of the time required to search. I will also demonstrate techniques that change the model itself, by incorporating abstraction or homogeneity to reduce the search space. Finally, a distributed search process is described that attacks the complexity through parallelism.

To evaluate the efficacy of ODML and the quality of the automated design process, I have developed complete models for two different operational domains: a distributed sensor network and a peer-to-peer, distributed information retrieval system. The distributed sensor network model has been created based on the design of an existing, real-world architecture [86]. The original system was developed and analyzed prior to the creation of ODML, and thus provides a unique opportunity to evaluate ODML's ability to represent and predict a number of organizational characteristics. This environment and model are described in detail in Chapter 2. The second model is inspired by an information retrieval system developed by Zhang and Lesser [215]. It features a network of flexible hierarchies that can be structured in a vast number of ways, providing a space of designs that is large enough to make simple brute-force approaches impractical. The interactions between entities are also complex, requiring the integration of different mathematical techniques to be correctly captured. This work is described in Chapter 3. The predictions produced by both models are also

empirically verified in those sections. Several other models are also presented in Chapter 5 as a demonstration of its applicability.

To summarize, the primary objective of this dissertation is to demonstrate the feasibility of a highly quantitative representation and the increased utility that such a representation brings to the organizational design problem. The following contributions will be made to that end.

1. I show that a flexible, accurate organizational representation grounded in arbitrary quantitative information can be created. This is accomplished through the design and implementation of the ODML language itself.
2. I show that it is possible to use such a language to quantitatively model and accurately predict the complex, interrelated characteristics of organizations operating in realistic domains. This is demonstrated through the creation and empirical evaluation of several ODML models that address different problems in different domains, each of which has a different set of relevant characteristics.
3. I demonstrate that such models are capable of capturing a range of different organizational designs at different levels of abstraction. This is accomplished by integrating decision points within ODML models that reflect the choices that must be made to create an organizational instance, and by demonstrating that by varying such choices different designs will be produced.
4. I demonstrate that techniques can be devised that automatically search and evaluate the design space to solve the organizational design problem. This is accomplished by devising techniques that use the mathematical substructure to bound the search space, and use quantitative predictions of instance behaviors to evaluate and rank competing designs.

I will return to this list of contributions in Chapter 7 to provide additional details on how they have been accomplished.

## 1.2 Major Ideas

This section continues introducing the dissertation by enumerating the basic assumptions I have made when pursuing the objectives outlined above. Sections 1.2.2 and 1.2.3 expand upon the high-level concepts that support and motivate this work through additional discussion of the organizational design and representation problem.

### 1.2.1 Basic Assumptions

The strategy that I present to satisfy the objectives outlined above assumes that it is both possible to model the system in question and that different systems have measurable differences. More completely, I assume the following conditions are true:

1. The characteristics of the environment, resources, agents, tasks, goals, or any other object relevant to the system’s performance are known, and they may be either determined exactly or approximated. This may be accomplished analytically, through repeated observation of the characteristic in question, or be defined by some other external process.
2. There are a range of design decisions that can be made, resulting in an array of candidate designs that exhibit different characteristics.
3. Active environmental entities, such as agents, are able to appropriately incorporate and respect any design choices specified by the selected model.
4. Quantifiable, measurable characteristics exist that can be used to differentiate candidate systems.
5. The set of characteristics relevant to the system’s intent can be combined in some manner to produce a single, numeric value that can be used to assign a preference order to the candidates. This value is the system’s *utility*.

With the possible exception of point 1, these same assumptions are also made in the majority of related work. The essence of point 1 is an assumption that the underlying mechanics of the system are either known or can be determined through analysis. This does not mean that there is no uncertainty, but it does mean that the level of uncertainty is also known, so that it can be represented and reasoned about. I will argue in Section 4.5 that, in the absence of perfect data, runtime adaptation can be still used to address problems that arise from designs based on incorrect or out-of-date knowledge.

I do not make any strong assumptions about the underlying architecture employed by agents or other entities within the organization, except to the extent that: 1) they can correctly enact relevant organizational decisions suggested by the model and 2) they exhibit behaviors that can be captured by that same model. Section 2.2 demonstrates that even complex inter- and intra-agent behaviors can frequently be captured with sufficient fidelity by a succinct set of expressions [84]. By employing abstraction or statistical characterizations, the germane aspects of such phenomena can be represented, reasoned over, and guided by an organizational design.

Later sections will present organizational models where additional assumptions may be made concerning the behaviors, conditions, or other features of the specific agents, environment or resources depicted in those models. I describe any such assumptions within their respective contexts.

## 1.2.2 Organizational Design

The organizational design of a multi-agent system is the collection of roles, relationships, and authority structures that govern the system’s behavior. All multi-agent systems possess some or all of these characteristics, and therefore all have some form of organization, although it may be implicit and informal. Just as with human organizations, agent organizations guide how the members of the population interact with

one another, not necessarily on a moment-by-moment basis, but over the potentially long-term course of a particular goal or set of goals. This guidance might influence the data flow, resource allocation, coordination patterns or any number of other system characteristics [74, 23]. This can help groups of simple agents exhibit complex behaviors, and help sophisticated agents reduce the complexity of their reasoning.

Implicit in this concept is the assumption that the organization serves some purpose — that the shape, size and characteristics of the organizational structure can effect the behavior of the system [62]. It has been repeatedly shown that the organization of a system can have significant impact on its short and long-term performance [24, 156, 84, 128, 6, 180, 17], dependent on the characteristics of the agent population, scenario goals and surrounding environment. Because of this, the study of organizational characteristics, generally known as computational organization theory, has received much attention by multi-agent researchers.

An organizational design can influence the system at many different levels of abstraction. For example, consider the sensor network variants discussed in Section 1.1. One aspect of the design might dictate how the agents are arranged from a high-level perspective, in this case using a completely centralized or a more distributed topology. This aspect is typically associated with deciding or putting limits on entity interactions. For example, in the centralized configuration, all sensors directly interact with the manager but never with each other. A design can also influence internal behaviors, such as specifying managerial or authority relationship. For example, the manager was given the authority to assign tasks to individual sensors in the network.

I also believe that an organizational design’s influence can extend to almost any aspect of any entity in the environment, agent or otherwise, that is relevant to the system’s performance; although this expanded definition is not universally held. For example, I assume an organizational design can specify the protocols that agents use to communicate, the algorithms used to find peers, and even the manner in which agents resolve purely internal conflicts. This is because these low-level characteristics can affect the performance of the conventionally high-level organizational structure and vice versa. Therefore, when selecting an appropriate design, all relevant factors should be considered, regardless of where in the architecture they may occur. This view is consistent with ODML’s approach to specifying design alternatives, which permits design alternatives to be modeled as an internal agent decision while permitting the ramifications of that decision to propagate elsewhere in the model.

It is generally agreed that there is no single type of organization that is suitable for all situations [91, 32, 112, 24]. In some cases, no single organizational style is appropriate for a particular situation, and a number of different, concurrently operating organizational structures are needed [63, 86]. Some researchers go so far as to say no perfect organization exists for any situation, due to the inevitable tradeoffs that must be made and the uncertainty, lack of global coherence and dynamism present in any realistic population [150]. Although I do not subscribe to this particular view, what is agreed upon is that all approaches have different characteristics that may be more suitable for some problems and less suitable for others.

Organizations can be used to limit the scope of interactions, provide strength in numbers, reduce or manage uncertainty, reduce or explicitly increase redundancy or

formalize high-level goals that no single agent may be aware of [113, 61]. At the same time, organizations can also adversely affect computational or communication overhead, reduce overall flexibility or reactivity, and add an additional layer of complexity to the system [84]. By discovering and evaluating these characteristics, and then encoding them using an explicit representation, one can facilitate the process of *organizational self-design* [31] whereby a system automates the process of selecting and adapting an appropriate organization dynamically [112, 161]. This approach will ultimately enable suitably equipped agent populations to organize themselves autonomously, eliminating at least some of the need to exhaustively determine all possible runtime conditions a priori. Before this can occur, the space of organizational options must be mapped, and their relative benefits and costs understood.

### 1.2.3 Representing Organizations

In the preceding section I repeatedly implied that there is or must be some way of explicitly representing the organization. For example, if one is to understand the organization's characteristics in a concrete and manipulable sense, there must be some data structure capturing those characteristics. Similarly, if one is attempting to predict the characteristics of a theoretical organization, there must be some model capturing the relevant features that underlie those characteristics.

Not all applications demand the creation of an explicit organizational design. In fact, most agent and distributed systems function without one. In these systems the organization still exists, but it is generally defined only implicitly. The roles entities take on may be homogeneous or hard-coded. Relationships may emerge through search or happenstance. Local behaviors are hard-wired or heuristically driven. No single, coherent model ties everything together. Clearly these systems perform adequately, or they would have little to demonstrate. I do not believe that an explicit representation is essential, but I do contend that it can be eminently useful. For example, a deep understanding of the ramifications of role assignment requires some model of what the role demands of the agent it is assigned to, and how those demands relate to other aspects of the agent's existence. Knowing when and with whom to form a relationship can be decided locally, but the more global consequences of this choice can be made more apparent by recognizing the potentially nonlocal effects it can have. The same is true of local decision making patterns – discovering and evaluating the nonlocal effects of these decisions can be greatly facilitated by an appropriate, explicit organizational representation.

The computational structure one uses to formally encode the organization's characteristics is the foundation upon which other organizational activities are based. Thus, if the structure is lacking in fidelity or capability, some activities may prove difficult or impossible to accomplish. For example, if the organization does not contain quantifiable measures of load or performance, it will be difficult for the agents to discriminate among competing strategies based on that criteria. Similarly, if the structure is too abstract, it may be unable to support the practical decisions that must be made during an agent's lifetime. Conversely, if it is too complex, it may prove to be computationally infeasible to work with the structure. Thus, there is a

tension between expressiveness and ease of use that must be addressed when selecting or defining an organization description language. ODML’s approach is in general more complex and detailed than related representations, resulting at the same time in more precise information and a more difficult search process. I believe this tradeoff is warranted by the additional precision it can produce, leading to more robust designs and a model that has value in a larger number of situations.

## 1.3 Guide to the Dissertation

Chapter 2 describes the organizational representation problem. It begins in Section 2.1 with an example of an organizationally-driven solution to the distributed sensor network problem mentioned above. A corresponding set of experiments provides concrete examples of how the design of an organization can affect performance, which motivates this work. Section 2.2 continues by introducing ODML, providing a formal description of the structure and built-in functionality. A detailed example of its use modeling the distributed sensor network system is given, along with empirical tests validating the predictions made by the model.

Chapter 3 describes and validates a detailed model of the information retrieval domain. This is used as a further example of ODML’s flexible yet precise nature. In particular, the mathematics involved in correctly modeling the system are more sophisticated than those used in the distributed sensor network domain. Section 3.3 details these formulations, and shows how the predictions they make can be used to find the optimal organization as environmental conditions change. The search space in this domain is large, which motivates the development of search techniques able to exploit the underlying structure.

Chapter 4 demonstrates how ODML can be used as the basis for designing organizations, using the DSN and IR models as examples. The complexity of the valid instance search is proved in Section 4.1.1. Section 4.1 describes the design search process itself, and details the algorithmic and modeling techniques that I have created to assist with that search. The chapter concludes with Section 4.5, which outlines how ODML can be used to address the online adaptation of organizations.

Chapter 5 provides additional examples of ODML’s use across a range of features and organizational paradigms. In particular, Section 5.1 shows how the distributed sensor network model from Chapter 2 can be enhanced to model geographic heterogeneity, temporal interactions and different levels of abstraction. Section 5.2 describes how models can be created to capture aspects of several additional organizational paradigms.

Although a brief review of related approaches to these problems is provided along with ODML’s introduction in Section 2.2.1, a more thorough description is delayed until Chapter 6. This chapter describes related representation and design research and contrasts those projects with ODML’s capabilities. Chapter 7 concludes with a summary of the work, and describes the contributions and conclusions that have been made.

There are several appendices that provide additional detail and context. Appendix A describes how parts of ODML model and search space can be translated for use in a general mathematical solver. Appendix B is a survey of common organizational paradigms used in multi-agent systems, which is used as the basis for the discussion in Section 5.2. The remaining Appendices C, D, E and F contain the complete textual source descriptions of several of the ODML models described in the dissertation.



## CHAPTER 2

### REPRESENTING ORGANIZATIONS

#### 2.1 Organizational Effects

This chapter starts with the presentation of a detailed, real-world example in section 2.1.1. This is done to better characterize the types of organizational structures I am concerned with, and how those structures impact the performance of a system. The intricacies and characteristics of this design demonstrate why this is an important and challenging problem to address, and both motivate and ground the discussions given in later sections.

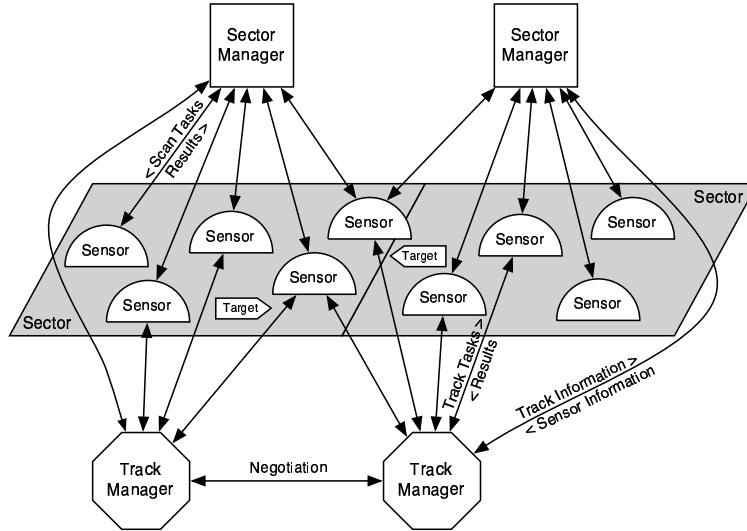
Recall the distributed sensor network (DSN) example given earlier. This was a simplified view of a DSN problem domain presented in DARPA's Autonomous Negotiating Teams project (ANTs) [110]. This section describes a system created to address the needs of that environment in three ways: by exploiting individual agent autonomy to quickly make decisions, by using negotiation to effectively allocate resources, and by using organizations to balance and reduce agents' computational and communication burdens [86]. This architecture serves as a working example of a complex, organizationally-driven multi-agent system.

Section 2.2 introduces the ODML representation, beginning with a formal, domain-independent description of the language. This is followed by a description of a complete ODML model of an operational system that has been constructed for the ANTs DSN domain. A set of quantitative predictions derived from that model are compared against a set of existing empirical results to validate the accuracy of the model.

##### 2.1.1 The Distributed Sensor Network Domain

The goal of a distributed sensor network is most generally to employ a population of sensors to obtain information about an environment. I will focus on using such a network to track one or more targets that move along arbitrary paths in an area. A collection of three-head, Moving Target Indicator (MTI) Doppler radars make up the sensor network. They are each fixed in position and have a wired power source. Each sensor is equipped with a processor, on which is run a single agent process that controls the sensor. The sensors are connected with a FM-based wireless network, which is divided into eight communication channels. Each channel has limited capacity, and agents may communicate over only one channel at a time.

Individual sensors can return only simple amplitude and frequency values, so a sensor is incapable of determining the absolute position of a target by itself. In

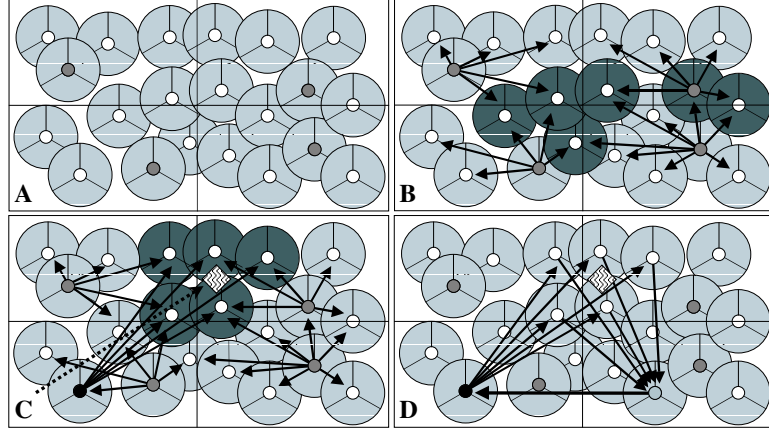


**Figure 2.1.** Organization-centric view of the DSN architecture.

addition, because only one of a sensor’s three heads may be in use at a time, each sensor’s scanning policy must be adapted based on current needs. To track under these conditions, the sensors must be organized and coordinated in a manner that permits their measurements to be used for triangulation, and geographically distinct groups of such coordinated sensors used to produce a continuous track as the target moves. More measurements, and particularly more measurements taken in groups in the same area at approximately the same time, will lead to better triangulation and a higher resolution track.

A system was created by myself and several other researchers to address this problem [86, 119]. The system’s architecture employs closed-loop control; the measurements and estimated target locations are used by the sensor agents to evaluate and adapt the network’s subsequent scanning strategies. Consequently, any processing, decisions making and communication that occurs to enact this control has to take place in real time, or the target may be lost. Additional hurdles include a lack of reliable communication, the need to scale to hundreds or thousands of sensor platforms over a wide area, and an uncertain, noisy operating environment. A more detailed description of the entire framework and the environment it operates in can be found in [110].

As mentioned above, an explicit organizational design has been employed in an effort to reduce overhead without negatively impacting performance. There are three types of responsibilities, or *roles*, that agents may take on: *sector manager*, *track manager* and *sensor manager*. They are related to each other and to the environment as shown in Figure 2.1. Sector managers are created for each sector in the environment, and serve as intermediaries for much of the local activity. For example, they generate and distribute plans needed to scan for new targets, store and provide local sensor information as part of a directory service, and assign track managers.



**Figure 2.2.** The DSN architecture in four phases. A: sectorization of the environment, B: distribution of the scan schedule, C: negotiation over tracking measurements, D: tracking data fusion.

Each detected target has such a track manager that is responsible for identifying the sensors needed to gather target information, gathering the resulting data, and fusing it into a continuous track. Track managers obtain some information from their originating sector manager, but can also interact directly with other sector and track managers. The sensor manager role controls how the local sensor is used. In response to sector or track manager requests, it takes measurements at specified times and places, and reports back the resulting data. Agents may work concurrently on one or more of these roles, so a viable organizational design must ensure that each agent has sufficient resources to meet the combined demands of the roles it is assigned.

Some aspects of this design are static, such as the partitioning and sector manager assignment, and defined as the sensors are deployed in the environment. Other aspects are dynamic, such as the track manager assignment and sensor selection, requiring the agents to self-organize in response to new events. This blend of styles takes advantage of characteristics of the environment that are invariant, without giving up the ability to react appropriately as conditions change.

To see how the organization works in practice, consider the sequence of behaviors shown in Figure 2.2. The environment is first divided by the agents into a series of sectors, each a non-overlapping, identically sized, rectangular portion of the available area as shown in Figure 2.2A. The intent of these divisions is to limit the interactions needed between sensors, to reduce and distribute the overall communication load. As shown in Section 2.1.2, this strategy does not always have the desired effect.

Each sensor has a local agent that takes on a sensor manager role. A single agent in each sector also takes on the sector manager role, represented by shaded inner circles in Figure 2.2A. Sensor managers begin their existence by finding their local sector manager, and sending it a description of the sensor’s capabilities. These include the sensor’s position, range, orientation and preferred communication channel. When completed, the sector manager will possess a complete picture of the sensing

capabilities within its sector, which it offers to other agents in the form of a directory service. The sector manager also uses this information to generate a scanning schedule for detecting new targets that it disseminates to the local sensors in Figure 2.2B.

Once the scan is in progress, individual sensors report positive detection measurements to their sector manager. The sector manager, through interactions with nearby track managers, maintains a list of targets currently close to or within its sector. By comparing the measurement with that target list, the sector manager can determine if a new target was found, or if it is more likely the measurement was of an existing target. If it determines a new target was found, the manager selects an agent from its sector to be the track manager for that target. Not all agents are equally qualified for this role, and an uninformed choice can lead to very poor tracking behavior if the selected agent is already busy or shares communication bandwidth with garrulous agents. For example, if we simply collocated the track manager and sector manager roles at the same agent, the combined communication load will generally exceed capacity. Conversely, if an agent who has previously acted as a track manager is chosen, some of the environmental state that agent had accumulated may be reused, which reduces its communication needs. Therefore, in making this selection, the sector manager considers each of its agents' estimated load, communication channel assignment, geographic location and history. Recognizing such ramifications of role assignment is an important aspect of the model presented in Section 2.2.2.

The track manager role, depicted in Figure 2.2C with a blackened inner circle, is responsible for tracking the target assigned to it. To do this, it first discovers sensors capable of detecting the target, and then negotiates with members of that group to gather the necessary data. Discovery is done using the directory service provided by the sector managers. As the target approaches a previously unknown area, the track manager will query the appropriate sector manager to determine the available local sensing capabilities. The track manager uses this knowledge to determine from where and when the data should be collected, and sends measurement requests to the sensor managers it selects (see Figure 2.2C). Because those sensors may be servicing tasks from other sector or track managers, conflicts can arise between the new task and previously existing commitments. The sensor agent will address such conflicts as best it can locally by using priorities to devise a round-robin schedule, but will also notify the conflicting managers of the problem. Because these managers have a more global view of the situation, they are in a more suitable position to resolve it. For example, they may negotiate with other track managers to use other sensing resources, or offer concessions in the form of reduced quality [121].

The data produced by the sensors is collected and analyzed (see Figure 2.2D). Although this activity is logically a separate role, it is a relatively lightweight process, and as a simplification the organizational design implicitly incorporates it into the track manager's responsibilities. Once the track manager has received the measurements, the data are fused in a triangulation process. Amplitude and frequency values can place the target's location and heading relative to their source sensor, and several of these relative values can be combined to derive an absolute position. The data point is then added to the track, which is used to predict the target's future

location. It is also used to periodically notify nearby sector managers of the target's location.

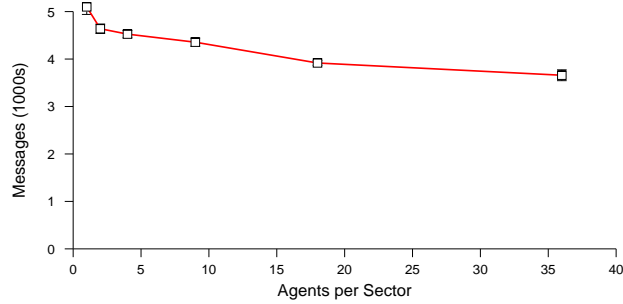
At this point the track manager must again decide which sensors are needed and where they should take measurements. Under most situations, the process described above is simply repeated. However, if the target has moved far from where the track manager is, the track managing role may be *migrated* to a new agent in a different sector. This is done to avoid penalties associated with long-distance wireless communication, which may cause unwanted latency or unreliability transferring information. This technique is covered in more detail in Section 2.1.5.

## 2.1.2 Empirical Demonstration of Organizational Effects

The two primary organizational features used by this system can be thought of as *geographic coalitions* and *functional differentiation*. The first describes the partitioning process, while the second is a result of the heterogeneous assignment of roles to agents. An integral part of each is the notion of locality. Information propagates and is made available to only the agents that have need of it. In some cases, such as with the environmental sectorization, artificial boundaries are created to encourage locality at the expense of time or flexibility. In other cases, as with the target tracking role, locality is exhibited naturally through the domain. Other, lower-level organizational structures are embedded within these, such as the peer-to-peer negotiation network that forms between track managers and the manager-worker relationships used to obtain measurements.

There are many data flows and interactions that are encouraged and restricted as a result of this design. As I will demonstrate, these characteristics affect the quantitative performance of both individuals and the system as a whole in a variety of ways. I will informally describe these effects below and show how they may be more concretely defined in Section 2.2.2.

In the following sections, I show empirical evidence exhibiting these characteristics, and I explain how they drive the selection of an organizational design. The experimental scenario consisted of a group of 36 sensors and 4 mobile targets. Differently sized sectors were tested in this scenario, ranging from 36 very small sectors each containing just one sensor to a single sector encompassing the entire area. All sectors shared the same size within any given test. The sensors were arranged in a grid pattern and the targets' locations and movement spread evenly through the environment to normalize results and simplify analysis. Targets moved with constant speed. The results were then observed over 10 runs per configuration in a simulation environment called Radsim, which closely models the performance of the physical MTI sensors [106]. The same agent code was used for both these simulation runs and actual hardware tests performed elsewhere. Each run lasted approximately 140 seconds.



**Figure 2.3.** Effect of sector size on messaging.

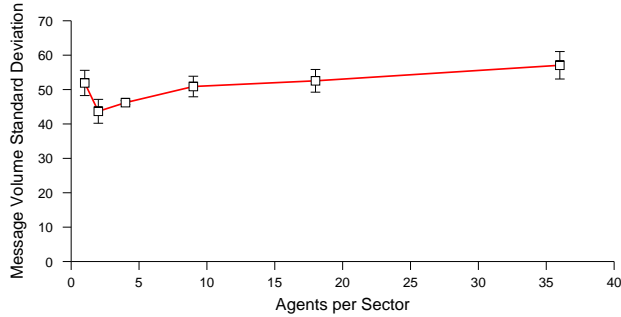
### 2.1.3 Geographic Coalitions

The first organizational effect I will explore is the total amount of communication that occurs in the system. Figure 2.3 shows that as the number of agents in each sector increases, and there are correspondingly fewer sectors overall, the amount of communication traffic decreases. Because each sector requires a certain amount of control messages, the total number of messages is reduced as the number of sectors decreases. A more detailed view of the effects this change has on messaging will be shown later in Figure 2.5.

Recall that the initial intent behind creating these sectors was to reduce the communication burden. The results in Figure 2.3 are in some sense contradictory of this goal, because they show that the unpartitioned environment had the lowest communication overhead. This aggregate metric obfuscates certain details, however, by hiding the potentially worse performance of key organizational participants. To see this, recognize that the partitioning process described in Section 2.1.1 results in the creation of loose coalitions of sensors based on geographic location. Sector directory information, new target scan schedules, discovery measurements and certain tracking control messages are all contained within or directed to these coalitions.

Because the manner in which this information being communicated is determined by the sectors, the sectors' average size and shape has a tangible effect on some aspects of the system's performance. If the sector is too large, and contains many sensors, then the communication channel used by the sector manager may become saturated. If the sector is too small, then track managers may expend a great deal of time and bandwidth updating sector managers as its target moves through the environment. So, although these results show that large sectors have lower total overhead, the individual picture is not so straightforward. This is covered in more detail below.

Although not shown in this figure, partitioning can also affect reactivity, because time may need to be expended to discover sector information. A track manager, for example, must perform queries to obtain sensor information as its target moves to new sectors. Smaller, more numerous sectors will result in delays caused by the additional queries that ultimately affects the number of measurements it receives. This delay will be revisited in Section 2.3.7.



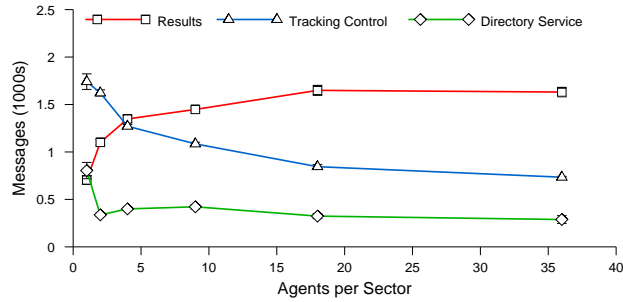
**Figure 2.4.** Messaging disparity vs. sector size.

### 2.1.4 Functional Differentiation

The varied assignment of roles in the system comprises a different organizational design based on the capabilities and responsibilities of the participants, also known as a functional organization [61]. Agents specialize their functionality in order to restrict the type of interactions that must take place between agents. For example, to obtain information about available sensors, a track manager must only contact the relevant sector managers, instead of blindly broadcasting to all sensors [184]. Concentrating the track management functionality into individual agents serves a similar role, by limiting the number of interactions necessary to resolve conflicts in sensor usage.

Although this type of functional decomposition does reduce the total number of interactions an agent might need to make, it can also increase that number for particular individuals in the environment. For example, we have seen how the sector manager is responsible for disbursing information about the sensors in its sector, which facilitates the track manager’s discovery process. However, by serving in this capacity, it also makes itself a center of attention, which can result in unreasonable load when demand is high.

Figure 2.4 shows how much agents in the population differ from one another in their communication habits as the sector size changes. This notion is captured by measuring the standard deviation in communication activity (total messages sent) exhibited by individual agents. If all agents are roughly the same they will have a low deviation, while a population that has a handful of outlier agents with significantly higher message traffic will have a high deviation. As the number of agents in each sector increases, this graph shows an increase in disparity, because a few agents are communicating more than their peers. Since the environment and target spacing are uniform, the differences can be attributed to the roles those agents take on. For example, the increasing trend as the sector size grows can be attributed to the increased concentration of communication caused by the correspondingly reduced number of sector managers that must handle the demand produced by the agent population. The rise in deviation when there is a single agent per sector represents the coexistence of the sector and track manager roles, because all agents act as sector



**Figure 2.5.** Message types vs. sector size.

managers when there is only one agent in each sector. This trend demonstrates that as the sector size grows, specialized agents such as sector and track managers can become hotspots of activity. In a bounded environment with unreliable communication this concentration of activity could lead to reduced performance and data loss if the communication channel becomes overloaded.

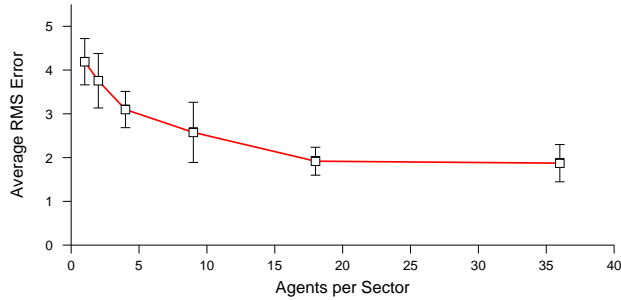
A growing tension between sector sizes is made apparent by these results: Figure 2.3 shows that smaller sectors lead to increased message traffic, and while Figure 2.4 shows that larger sectors imbalance load in the population. Although not shown, similar trends were observed in agents' local workload levels, which track the number of non-communicative actions being performed. Both characteristics are bad, so a compromise must be sought between them in the selected organizational design.

### 2.1.5 Organizational Maintenance

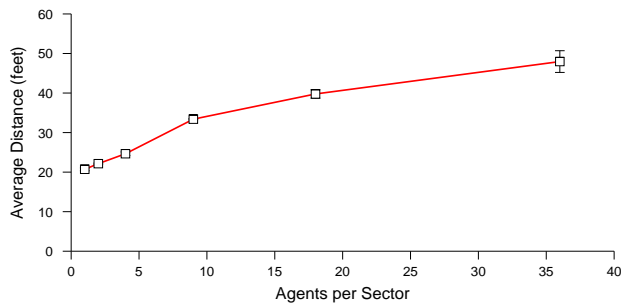
There are costs associated with creating and maintaining the organizations employed by this design. The most frequently updated aspect of the organization is the relationship formed between track and sector managers because the sectors that track managers interact with change as the target moves. This results in a class of control messages dependent on sector size. For example, as the target moves into part of the environment the track manager is not familiar with, the manager must query the sector manager of that area to discover local sensors. Once those sensors are known, data collection commitments can be established. As the target is tracked, the nearby sector managers must also be periodically notified of the target's estimated position.

Figure 2.5 provides a quantitative view of this overhead. As sector size increases, fewer directory and tracking control messages are necessary, because there are a fewer sectors to interact with as the target moves. In addition, the number of measurements increases as the sector size increases, because the reduced time spent by the track manager interacting with the additional sector managers allows more time to be spent requesting data. More measurements results in a lower root-mean-squared (RMS) error between the measured and actual track, as seen in Figure 2.6. RMS error is a key measure of overall system performance in this domain.





**Figure 2.6.** Effect of sector size on RMS error.



**Figure 2.7.** Average communication distance.

The technique of migrating the tracking responsibility through the agent population as the target moves is another example of how locality can be exploited. Signal attenuation conspires to make communication less reliable as distance increases. Multi-hop protocols can maintain reliability, but will increase end-to-end latency at each hop. Lacking the capacity for movement, the initial manager selected to track a target will therefore become less effective as the target moves away from it. By migrating this task to follow the target, the organization is able to retain locality despite the fact that the sensors themselves are immobile. This results in a reduction in the average distance that messages must travel.

Figure 2.7 shows the effect track manager migration has on the average distance of communication. Because migration is triggered by sector boundaries, the tracking task will migrate less frequently when sectors are large, simply because they cover more area. Conversely, a lower average communication distance is observed when sectors are smaller. The lower migration rates also contribute to the increased measurement totals from Figure 2.5. Each migration interrupts the collection process as the role is moved from one agent to another, so the more frequently this transfer takes place, the more the average overall collection rate will be reduced.

A different approach that estimated this distance directly instead of indirectly through sector boundaries is certainly possible. Decoupling the two characteristics

would likely avoid the trend shown by Figure 2.7. The original decision to employ this technique was made because it was assumed that other pressures (like those shown in Figures 2.4 and 2.6) would keep the sector size within an acceptable range. Figure 2.7 shows a side effect of that decision, and is an example of the type of interactions that an appropriate model can help uncover. If our assumptions had been incorrect, for example, or if the capacity of the wireless medium was increased without a corresponding increase in range, subsequent design changes could fail to take these interactions into account. An explicit model like that shown later in Section 2.3 can help avoid such mistakes, as well as provide guidance over which technique is most appropriate.

More generally, these results show how different characteristics contribute to the organizational tension. Large sectors improve the system’s RMS error rate, while smaller sectors exhibit better communication locality.

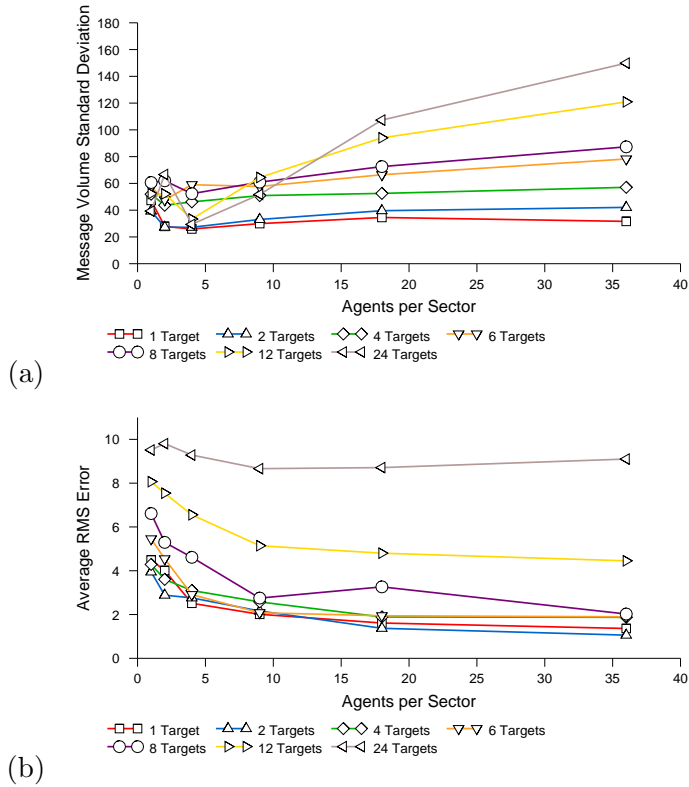
### 2.1.6 Generality of Effects

To explore the generality of these trends, I performed an additional set of experiments that varied numbers of targets. Each test contained between 1 and 24 equally distributed targets, all of which moved concurrently through the environment for the duration of the experiment. The scenario was otherwise identical to those in Section 2.1.2. Figure 2.8a shows that the original communication disparity profile from Figure 2.4 is maintained as the target density is varied, and the amount of disparity increases with the number of targets. Intuitively, this is because the amount of work particular agents are performing is tied to the number of targets in the environment. The communication load of the sector managers, for example, is directly proportional to the number of track managers it must interact with. This is particularly true as the sector size increases – in the most extreme case a single sector manager must support all 24 track managers.

Similarly consistent results are seen for the systems RMS error in Figure 2.8b. The RMS error profile is maintained, although the baseline RMS error increases because the bounded sensing capabilities result in fewer average measurements per target. Notice how the RMS value for 6 and fewer targets are clustered together, while those with 8 or more become progressively worse. This is caused by a phase transition that occurs between 6 and 8 targets, when the number available sensors is no longer sufficient to meet demand. The inevitable reduction in the number of measurements track managers receive leads to an increase in RMS error.

Additional tests were performed which also varied the number of sensors in the environment, using six different configuration with between 9 and 81 sensors [84]. Results from those experiments concur with the trends outlined above.

The conclusion that may be drawn from these experiments is that a tradeoff exists between the overall volume of message traffic and its distribution over the agent population. Message volume decreases when there are more agents per sector because fewer interactions are needed to obtain information, as shown in Figure 2.3. However, this shift can cause individual agents to incur a disproportionate communication burden, as shown in Figures 2.4 and 2.8a. Figures 2.5, 2.6, and 2.7 show that organizational



**Figure 2.8.** Communication disparity (a) and RMS error differences (b) with varied sector sizes and target densities.

maintenance causes a similar tradeoff - larger sectors have lower overhead and better RMS error, while more track migration in smaller sectors increases communication reliability.

More generally, these results also demonstrate the tangible effects that organizational design can have on system performance. Recognizing and understanding these effects is the first step in determining what the most appropriate design should be. In this instance, I explored the effects of just one aspect of the organization under a relatively small set of environmental conditions. However, the consequences of this and other organizational decisions depends on a number of factors not addressed here, including the density of the sensors, sensor range, communication medium characteristics and target speed, among other things. Finding the appropriate organization for any given instance of these factors cannot be accomplished with just a small set of experiments. The following section will show how explicit organizational models created with ODML can be used to address this need.

## 2.2 Organizational Representation

In Sections 1.1 and 1.2, I described the theoretical basis for agent organizations, by alluding to the possible benefits and drawbacks they can provide. The previous section provided concrete evidence of this, by demonstrating that different organizational designs can have tangible effects on a range of performance characteristics. In this section I show how it is possible to capture those effects in a predictive model. Such a model will be used in Section 4 to support an automated organizational design process.

I have previously created a set of discrete analytic equations to describe and predict the distributed sensor network behaviors outlined in the previous section [84]. Although individually precise, such models lack the cohesion necessary to create a complete prediction of system performance. There is no strong notion that particular and distinct entities exist with associated characteristics. There is no well-defined way of specifying what decisions must be made, what values must be optimized over, or what constraints must be respected. Instead, such individual expressions provide performance characteristics piecemeal, and comparative analysis of the entire system is performed later in a relatively ad hoc manner. Finally, I believe a single, static set of equations will be unable to represent all the alternative ways that a structure might be created in a concrete and accurate manner. For example, consider if there were a choice of the type of sensor or agent available for use in the environment, different tracking techniques that might be employed, or optional information aggregation hierarchy of arbitrary height and width. While one could create individual models for each dimension, combining them together in a coherent and expressive way would be challenging. It is for this reason that I view tools that operate principally on such representations, such as nonlinear solvers and queuing networks, as too limiting to address the general organizational design problem (although I believe they can play an important role in certain aspects of modeling and design evaluation). Section 4.2.3 highlights the difference between the capabilities of these general techniques and the problem addressed by the work presented here.

To address this deficiency, I have developed a representation designed to capture such organizational information into a single, unified structure, along with a set of techniques to analyze and search the space of organizational possibilities. The Organizational Design Modeling Language (ODML) provides domain-independent mechanisms to model, evaluate, and compare a variety of organizational styles, including the sensor network described above. ODML incorporates quantitative information in the form of mathematical expressions. These expressions are grouped into organizational constructs, connected in a graph of relationships, and ultimately used to represent and predict both the localized and global characteristics of an organization.

The immediate benefits of such a language are twofold. First, by incorporating quantitative information about the environment, resources, agents, roles, tasks, goals, or any other object relevant to the system's performance, candidate organizations may be tailored and evaluated in a context-specific way. For example, one can directly embed information about relevant characteristics such as sensor density, target velocity, communication limitations, and the like. This model can then be

used to determine the organization which is most appropriate for that context, given a particular definition of utility. By simply changing those values, a new context can be defined, potentially resulting in a new corresponding organization. Second, once a suitable model has been found, it can serve as an explicit organizational representation, guiding agents' local decisions in a manner consistent with global objectives. The longer-term benefits of the organizational model include being able to make predictions about runtime performance, which can be used to isolate and diagnose system failures and deficiencies. This same information can also be used to support adaptation of the system, by incorporating learned knowledge into the existing model and analyzing the resulting structure. These benefits will be explored in Section 4.5.

### 2.2.1 ODML

An organizational model serves in several different capacities. At design time, it should be possible to use the structure to create and evaluate not just a single organizational instance, but an entire family of organizational possibilities. At runtime, it should accurately describe the current organization. In both cases, the model must be sufficiently descriptive and quantitative that one can evaluate the organization's effectiveness and rank alternatives according to some specified criteria. Below, I enumerate the desired capabilities and characteristics the modeling language should possess to satisfy these requirements:

1. It should be able to represent a particular organizational structure. This would include roles, interactions and associations (e.g., coalitions or teams). Different flows in the organization, such as communication and resources, should be representable.
2. It should be able to represent the range of organizational possibilities, by identifying general classes of organizations and the parameters which influence their behavior. Different elements should be able to be modeled at different levels of abstraction. Identify which characteristics are under deliberate control, and which are derived from external factors.
3. It should enable concrete performance predictions and allow deductive analysis by quantitatively describing the relevant characteristics exhibited by the structure, the manner in which those characteristics interact, and the constraints they are affected by. For example, both communication overhead and the effect that overhead has on work load should be representable.

A representation that meets these requirements is sufficient to support the automated organizational design process, by first representing the space of alternatives and then using quantitative predictions to evaluate and rank competing designs in that space. Many different organizational representation schemes have been developed by researchers [187, 109, 129, 171, 50, 59, 122, 88, 174]. Nearly all these representations can satisfy the first two points, but none are able to incorporate quantitative knowledge in such a way that concrete predictions along multiple, interdependent

dimensions can be made directly from the model itself. Without this ability, such representations must rely on qualitative comparisons, which can lack important contextual detail, or on external heuristic or simulation-based evaluations, which are typically more time consuming.

Most existing representations fall into one of two categories: either they represent a wide range of organizational characteristics abstractly, or they can capture a smaller set of characteristics concretely. The former are usually good at representing what entities or relationships exist or could exist, but cannot compare alternatives in a quantitative way. The latter may contain quantitative knowledge, but have difficulty relating that knowledge to specific organizational concepts, either because the quantitative information is only indirectly related to the organizational structure or because a separate simulation is required to elucidate the actual performance. This mitigates their usefulness if one is hoping to understand the effects a particular organizational design will have, particularly in response to the needs of a dynamic environment.

For example, OMNI [50] and  $\mathcal{M}\mathcal{O}\mathcal{I}\mathcal{S}\mathcal{E}^+$ [88] can each capture a greater variety of explicit organizational concepts than ODML, but do so in a largely qualitative way. For example, they have concrete notions of norms, ontologies and plans, but no way to directly relate the organizational decisions that define those features to the qualitative effects they have on performance. Work by Matson and DeLoach [129] does dynamically compute the quantitative utility of an organization, but does so using only a single approximate quality statistic. The previous section showed how many different interacting features affect the utility of an organizational design. Conversely, both SADDE [171] and MIT's Process Handbook [122] can incorporate arbitrary quantitative information, but neither couples this information with the organizational structure in a way that enables one to deduce how the characteristics of one aspect of the design affect another. The representation created by Sims [174] does incorporate quantitative information into a structured organizational model, but I believe ODML's more flexible design can model more situations at different levels of abstraction. For example, although one can model individual agents and roles in ODML, the representation does not require that such elements exist. By modeling these concepts only abstractly or not at all, one can potentially create models of much larger systems without the associated high combinatorics. At the same time, this flexibility can make the design search itself more difficult. These differences will be expanded upon in Section 6.1. Each representation has its strengths and ODML's goal is not to supplant these works, but to demonstrate another approach that makes different tradeoffs. As shown in the following section, ODML does so by incorporating a concrete but flexible set of primitives that can model a range of organizational constructs along with the quantitative characteristics that differentiate them.

I believe the principal benefit of using a quantitative representation is the ability to make rapid but precise predictions about organizational performance. If one views the process of organizational design as a search through the potentially very large space of possible organizational structures, a critical part of that search is the ability to evaluate alternative designs with respect to an expected operational context. I further believe that the utility of an organizational structure should not be restricted to just a single metric, but can be based on many characteristics that may be tightly

coupled with one another and the structure itself. In this section I will show how ODML models are able to capture both the space of alternatives and the complex characteristics they exhibit in a representation that can make predictions much faster than is possible through simulation. If used as part of a design process, this advantage can allow the evaluation of a greater number of alternatives, which will increase the chance that the most suitable design will be found.

Conceptually, ODML models exist in two distinct forms that share a common representational definition. The first acts as a *template*, that expresses a range of organizational possibilities by explicitly encoding the organizational decisions that must be made. The second is an organizational *instance*, created from the template by making specific choices for those decisions.

Formally, an ODML template specification  $\mathcal{O}$  is defined as follows:

$$\begin{aligned}\mathcal{O} &= \{\mathcal{N}, H, C, K, M, V\} \\ \mathcal{N} &= \{N_0, N_1, \dots, N_n\} \\ N_i &= \{t, \ell, \bar{p}, I, H, C, K, M, V\}\end{aligned}\tag{2.1}$$

Section 2.2.2 gives examples of how these features are used in practice. The bulk of the ODML template specification is made up of the set  $\mathcal{N}$  of *nodes*, each of which corresponds to a particular physical or logical entity that might exist in the organization. For example, in the sensor network scenario there would be nodes corresponding to sectors, managers, relationships, agents and the environment, among other things. Each node  $N_i$  contains a number of elements, defined below:

$t$  The node's *type*. This label must be unique within the set of template nodes that make up the organization.

$$\begin{aligned}N.t &= \langle symbol \rangle \\ \forall N, M \in \mathcal{N}, N.t = M.t &\Leftrightarrow N = M\end{aligned}$$

$\ell$  The node's instance *limit*. This specifies the maximum number of instances of the node type permitted in a valid organizational instance.

$$N.\ell \in \{\mathbb{Z}^+ \cup \infty\}$$

$\bar{p}$  An ordered list of *parameters* that must be passed to the node's template when an instance of the node is created. These are analogous to the parameters one might pass to an object constructor. Each parameter is specified with a type and local name.

$$N.\bar{p} = [\langle symbol, type \rangle, \dots]$$

$I$  The set of node types that this node has an *is-a* relation with using conventional object-oriented inheritance semantics. If we assume that a node's  $I = \{a, b\}$ , an instance of the node will also be an instance of  $a$  and  $b$ , possessing the characteristics of all three node types (e.g., if  $a$  has a constant  $x$ , then the related node will have the same  $x$  unless it locally overrides it). Is-a relationships cannot be cyclic, i.e.,  $N$  cannot have itself as a decedent.

$$N.I = \{\langle type \rangle, \dots\}$$

$$\forall i \in N.I, N \neq i \wedge N \notin i.I \wedge \dots$$

*H* The set of node types that this node has a *has-a* relation with. If we assume that  $H = \{a, b\}$ , an instance of the node will possess some number of instances of both *a* and *b*. It is through this type of relationship that the primary organizational decomposition is formed. Each has-a has a magnitude that specifies the number of instances connected by the relationship.

$$N.H = \{\langle symbol, type, magnitude \rangle, \dots\}$$

$$magnitude = \langle symbol \rangle$$

*C* A set of *constants* that represent quantified characteristics associated with the node. Constants may be defined with numeric constants (e.g., 42), or mathematical expressions (e.g.,  $x + y$ ).

$$N.C = \{\langle symbol, expression \rangle, \dots\}$$

*K* A set of *constraints*. Also defined with expressions, an organization is considered valid if all of its constraints are satisfied.

$$N.K = \{\langle symbol, op, expression \rangle, \dots\}$$

$$op \in \{<, >, \leq, \geq, =, \neq\}$$

*M* A set of *modifiers* that can affect (e.g., mathematically change) a value contained by a node. Multiple modifiers may affect the same value. Modifiers model flows and interactions by allowing the characteristics and decisions made in one node to affect those of another.

$$N.M = \{\langle symbol, op, expression \rangle, \dots\}$$

$$op \in \{+, -, \times, \div\}$$

*V* A set of *variables*, representing decisions that must be made when the node is instantiated. Each variable is associated with a range of values it can take on. For example, a node might have a variable *x* that could take any one value in the set  $[2.7, y^2, \pi z]$ .

$$N.V = \{\langle symbol, \{expression, \dots\} \rangle, \dots\}$$

*symbol* refers to a user-defined string, similar to a variable name in a conventional programming language. These typically describe or refer to a particular characteristic. *type* is the type name of some defined node, so  $\exists N \in \mathcal{N}$  such that  $N.t = type$ . *expression* is an arbitrary algebraic expression, possibly referencing constants, symbols and function calls.

The top-level organization node  $\mathcal{O}$  also contains the elements *H, C, K, M, V*, providing a location for the designer to embed additional global information and constraints.

I refer to *C, K, M, V* collectively as a node's *fields*, and the quantitative state of a field as its *value*. For example, the constant field *total\_load* might be defined with



the expression  $total\_load = work\_load + communication\_load$  and have a value of 0.9 for an agent in a particular organization. Note that the use of the term “constant” may initially be misleading. While the expression defining  $total\_load$  is fixed, the value for  $total\_load$  produced by that equation may change through the application of modifiers, or due to changes in fields or values that the expression is dependent on.

At first glance, the ODML language may appear to be devoid of almost all the organizational concepts that are provided by typical organizational representations. This is partially true, and it is by design. Instead of directly incorporating the usual high-level organizational components, such as hierarchies, roles, agents, etc., ODML provides a set of relatively low-level primitives by which such structures can be defined. For example, a node with the user-defined type *manager*, having a has-a relationship with another node of type *agent* could embody a role-agent relationship. A sequence of has-a relationships between nodes could indicate a hierarchy. Although the high-level semantics for these nodes may be implicit, the concrete characteristics and design ramifications are still directly and quantitatively captured by the nodes’ fields.

My opinion is that this approach can lead to an increased diversity of representable structures, by avoiding the assumptions and inevitable restrictions that typically accompany high-level structures. The remainder of this chapter as well as Chapter 5 and Section 3.1 will demonstrate this flexibility by showing how many different types of organizations and organizational effects can be successfully modeled using ODML. Section 4 will also demonstrate that this approach simplifies some aspects of the organizational search process, by unifying the various ways that design alternatives can be specified into two well-defined template characteristics (has-a relationships and variables).

ODML instances are quite similar to templates. The difference is that where a template is a description of what *could be*, an instance is a description of what *is*. Where a template might specify that a *manager* role can be assigned to a single *agent* or distributed across multiple *agent* nodes, an instance would indicate that *manager\_1* is distributed across *agent\_5* and *agent\_7*, and so on. Once instantiated, the expressions defined by the fields, the data passed in through parameters, and the interactions caused by relationships can all be used to predict values for an individual node’s characteristics.

The formal definition of an instance is nearly identical to that given in Equation 2.1, so I will not repeat it here. The differences principally relate to the replacement of node types in the template with instances of those nodes in the organizational instance. Thus, the set  $\mathcal{N}$  is the set of node instances, whose individual types no longer need be unique. So, where there might be just a single *manager* type in the template, there can be an arbitrary number of *manager* instances in the instance. Both is-a ( $N.I$ ) and has-a ( $N.H$ ) relationships no longer reference node types, but particular node instances in  $\mathcal{N}$ . Finally, the set  $\bar{p}$  is filled with appropriate values from each node’s parent, and the variable set  $V$  for each node is replaced by a single item from that variable’s range. Because a common syntax is shared between the two forms, for the remainder of this document I will indicate where necessary which is being considered.

```

get_value(symbol s)
  r ← null
  if (s is of the form s1.s2)      -- Dereference dot-notation symbols
    n ← get_value(s1)
    r ← n.get_value(s2)
  else if (∃ c ∈ C | c.symbol = s) r ← evaluate(c.expression)  -- Constants
  else if (∃ h ∈ H | h.symbol = s) r ← h                    -- Has-a members
  else if (∃ v ∈ V | v.symbol = s) r ← evaluate(v.expression)  -- Variables
  else if (∃ p ∈ p̄ | p.symbol = s) r ← p                    -- Parameters
  else forall i ∈ I      -- Handle is-a relationships
    r ← i.get_value(s)
    if (r ≠ null) break
  forall m ∈ M      -- Add local modifier effects
    if (m.symbol = s)
      r ← r m.op evaluate(m.expression)
  forall n ∈ N      -- Add remote modifier effects
    forall m ∈ n.M
      if (m.symbol is of the form s1.s2) ∧ (s1 = N) ∧ (s2 = s)
        r ← r m.op n.evaluate(m.expression)
  return r

evaluate(expression e)
  forall s ∈ { non-function symbols referenced by e }
    vs ← get_value(s)
    substitute all occurrences of s ∈ e with vs
  r ← mathematical result of e
  return r

```

**Figure 2.9.** Pseudocode for the `get_value` function of a node  $N$ . This function is used to quantify the characteristics of instance nodes.

As mentioned above, it is the ability to use an ODML model to deduce quantitative values for specific characteristics that sets it apart from other representations. The manner in which these values are determined for an instance node’s characteristics is defined by the pseudocode in Figure 2.9, that outlines the `get_value` function for computing the value of a symbol. Note that some aspects of `get_value`’s behavior, such as the manipulation of list-based data, have been omitted for clarity. This function shows how various sources of information, non-local data and node interrelationships all interact to describe the features of a particular node. It is through the execution of this function on a particular symbol that predictions are made of the design’s performance. For example, `agent.get_value(total_load)` would return a prediction of `agent’s total_load`.

As will be shown in Section 4, the process of finding an appropriate organization revolves around first finding the set of valid designs, and selecting from that set the

one that is most desirable. The validity of a particular organizational instance  $\mathcal{O}$  is defined as:

$$\begin{aligned}
\mathcal{O} \text{ is valid iff } & \forall N \in \mathcal{O}.\mathcal{N}, N \text{ is valid} \\
N \text{ is valid iff } & \sum_{M \in \mathcal{O}.\mathcal{N} | M.t=N.t} 1 \leq N.\ell \\
& \wedge \forall k \in N.K, (N.get\_value(k.symbol) \ k.op \ k.expression) = true
\end{aligned}
\tag{2.2}$$

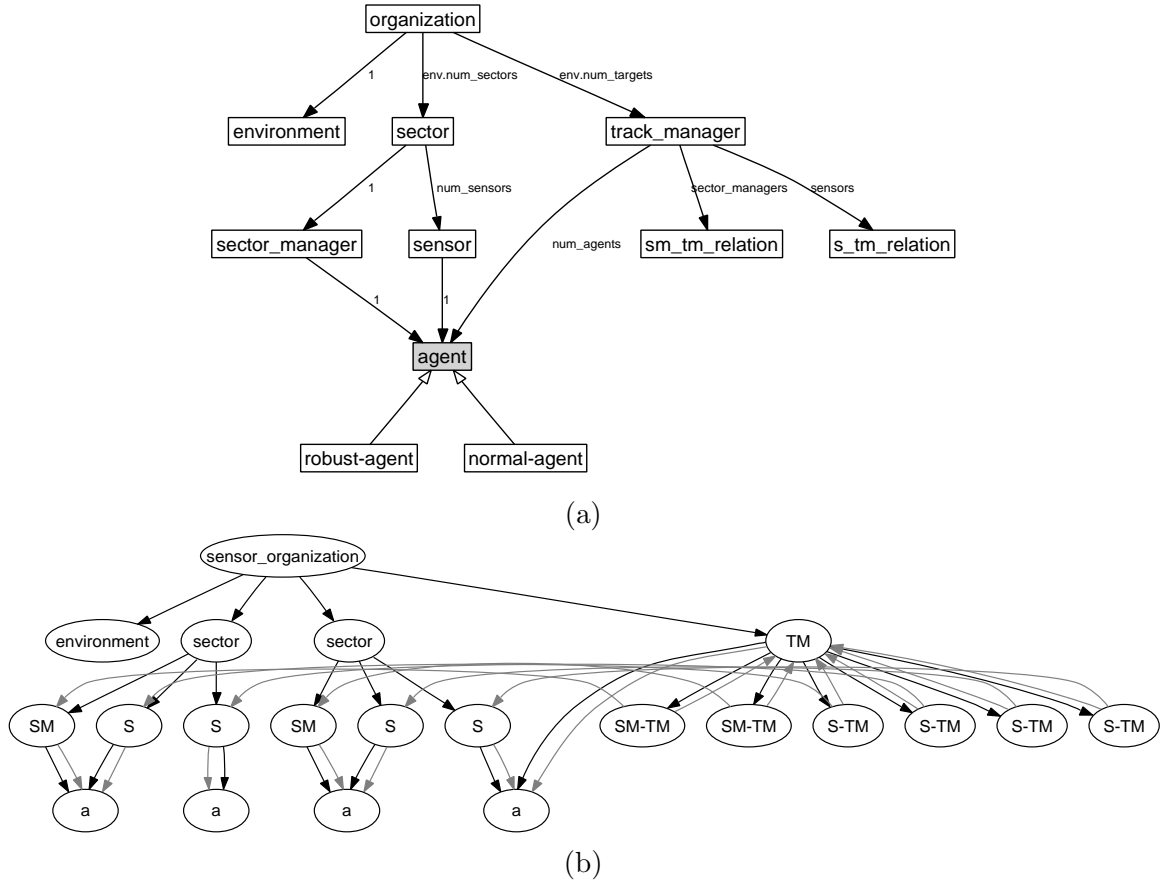
The “desirability” of instance  $\mathcal{O}$  can be quantified by defining a *utility* characteristic in the organization. This can then be computed using the existing machinery by calling  $\mathcal{O}.get\_value(utility)$ . Once such a value has been computed for all candidate organizational designs, they may be ranked and the best selected. Examples of utility definitions be given in Sections 2.3.8 and 3.3.

## 2.2.2 Distributed Sensor Network Model

The comparison to related work given in Section 6.1 shows that ODML’s method of defining organizations is more free-form than most other existing organizational representations. On one hand, this characteristic can offer a great deal of flexibility in how and what organizations are represented, and this capability is exploited repeatedly in the techniques presented in this work. On the other hand, this approach has relatively few high-level guiding structures, which can make it difficult to initially grasp how ODML models should be created and how the various pieces of a model really interrelate.

The capabilities of ODML are best explained through an example. The distributed sensor network (DSN) framework described in Section 2.1 was designed, developed and analyzed prior to the existence of ODML, making it an ideal platform to gauge ODML’s ability to accurately depict the characteristics of a sophisticated, real-world architecture. This section will use the DSN system to motivate and demonstrate ODML’s capabilities. Section 3.1 provides further demonstration of ODML by modeling a second domain, a distributed information retrieval (IR) system. The flexible, hierarchical structure of the IR architecture offers a vast range of organizational possibilities, creating a challenging representation and design problem.

I will proceed with an overview of how an ODML model was produced for the DSN organization described in Section 2.1.1. For clarity, the names of particular nodes or fields that reside in nodes are represented in italics. Recall that ODML models are divided into templates and instances, the latter being derived from the former. A graph showing the high-level structure of the sensor network’s template can be seen in Figure 2.10a. This graph is derived from a much more detailed specification that can be found in Appendix C. This complete description is roughly 280 lines long, so a smaller but representative portion is provided in Figure 2.11. The Extensible Markup Language (XML) [14] was selected as the basis for ODML’s syntax, to leverage the abundance of tools that already exist to parse and manipulate such data.



**Figure 2.10.** Example ODML (a) template and (b) instance structures for the sensor network organization.

### 2.2.2.1 Has-A Relationships

Vertices in the template graph, such as *sector* and *sensor*, correspond to nodes in the ODML model. Nodes can represent both tangible (e.g., *agent*) and intangible (e.g., *sector*) entities. At the root there exists an *organization* node. This is typical of all designs created in ODML. The organization node serves as a container, the common root, of all other nodes in the structure. This is true even if multiple, otherwise independent structures exist in a design, because conceptually they still operate within a common environment (the agent world), which implicitly relates them even if no other aspects do. For example, the *sector* nodes are related in a peer-to-peer fashion, no concrete entity exists above to manage or control them. Thus, although the graph-inspired design used by ODML facilitates the modeling of decomposable organizations, it is not limited to depicting this class.

Edges in the graph represent different types of relations between nodes. Directed edges with a solid arrow represent has-a relations, and the corresponding label indicates the magnitude of that relation. For example, each *track\_manager* node has a

```

1 <node type="track_manager">
2   <param>organization:org,environment:env,[sector]:sectors</param>
3   <is-a>entity</is-a>
4   <has-a name="agent" size="num_agents">agent(env)</has-a>
5   <has-a name="sm_relations">forall(sm, sector_managers):sm_tm_relation(org, this
6     , sm)</has-a>
7   <has-a name="s_relations">forall(s, sensors):s_tm_relation(org, this, s)</has-a
8     >
9   <!-- Determine how frequently the role will migrate -->
10  <constant name="num_agents">max(1, min(max_agents, migration_rate * env.time))<
11    /constant>
12  <constant name="migration_rate">1 / ((2 * average_sector_area)^0.5 / velocity)<
13    /constant>
14  <!-- Determine target bounds -->
15  <constant name="uncertainty_radius">5</constant>
16  <constant name="influence_radius">uncertainty_radius + 10</constant>
17  <constant name="target_area">3.14 * influence_radius^2</constant>
18  <!-- Calculate requested measurement rate -->
19  <constant name="desired_sensors">3</constant>
20  <constant name="sensor_density">forallavg(sectors.sensor_density)</constant>
21  <constant name="actual_sensors_available">target_area * sensor_density</
22    constant>
23  <constant name="requested_sensors">min(desired_sensors,
24    actual_sensors_available)</constant>
25  ...
26 </node>
27
28 <node type="s_tm_relation">
29   <param>organization:org,track_manager:tm,sensor:s</param>
30   <!-- Calculate actual measurement rate -->
31   <constant name="requested_sensor_rate">tm.requested_sensors / org.total_sensors
32     </constant>
33   <constant name="requested_measurement_rate">tm.requested_measurement_rate *
34     requested_sensor_rate</constant>
35   <modifier name="s.requested_measurement_rate" op="+">requested_measurement_rate
36     </modifier>
37   <!-- Assign measurement communication load -->
38   <constant name="actual_measurement_rate">requested_measurement_rate * s.
39     actual_measurement_ratio</constant>
40   <modifier name="tm.actual_measurement_rate" op="+">actual_measurement_rate</
41     modifier>
42   <modifier name="s.message_rr" op="+">actual_measurement_rate</modifier>
43   ...
44 </node>

```

**Figure 2.11.** A portion of the ODMML specification for the *track\_manager* and *s\_tm\_relation* nodes. The structural has-a relations are defined in *track\_manager* first, using an estimate of the role's migration rate and the number of extant *sensor* and *sector\_manager* nodes. An estimate of the target bounds is expressed next, and that bound used with the environmental sensor density to estimate the number of sensors that will be contacted for data. The *sm\_tm\_relation* shows how modifiers are used to first propagate that demand to a *sensor*, and later to inform the track manager of the the resulting actual measurement rate.

number of *agents* defined by the field *num\_agents*. The corresponding definition is shown in line 4 of Figure 2.11. Because the has-a’s magnitude may reference a field, the actual number of relationships that are created can be dependent on other organizational factors. In this particular case, *num\_agents* is a constant that is used here to control how the *track\_manager* role is distributed. It depends on number of other factors, including the role migration rate, sector size and estimated target velocity, and thus may have a different value in different conditions. This is covered in more detail in Section 2.3.5. In another example, *organization*’s has-a relationship with *sector* has magnitude *num\_sectors*, which is defined with a variable field. This allows the number of sectors used in the design to be an explicit organizational decision.

These examples demonstrate how the structure of the organization, as embodied in part by these has-a relations, can be affected by characteristics or decisions elsewhere in the model. This is an important aspect of how ODML templates are used to define contextually-appropriate structures, and are revisited when I discuss the organizational design process in Section 4.1. Although not shown in the sensor network model, has-a relationships in the template may also be recursive or self-referential. This facilitates the modeling of self-similar organizations such as hierarchies or holarchies, and are demonstrated in Section 3.1 (see Figure 3.2a).

### 2.2.2.2 Is-A Relationships

In Figure 2.10a, a hollow-arrow edge represents an is-a relation, so *normal\_agent* is an instance of *agent*. Shaded nodes, such as *agent* are abstract, and cannot be directly instantiated. Thus, any node with a has-a relation with *agent* can instead substitute *normal\_agent*. This level of indirection allows this model to represent and easily use agents with different capabilities. For example, suppose there were two types of agents available for use: a normal agent, and a “robust” agent that had better communication capabilities but a higher cost. To model this, a *robust\_agent* node can be created that also has an is-a relation with *agent*, and can be substituted for *agent* in the same way. A similar arrangement could model a range of alternative roles that had different characteristics but could serve overlapping purposes.

This aspect of the is-a relationship shows how organizational decisions may be encoded (e.g., the choice of which type of agent to use). This will be covered in more detail in Chapter 4. Is-a relationships are also used to allow the characteristics of one node to be available in another. For example, the base *agent* node defines a range of characteristics shared by *normal\_agent* and *robust\_agent*. The underlying ODML code is shown in Figure 2.12. *agent* provides default values for the *sensors\_controlled* and *roles* fields, as well as for a number of message types. It also specifies a constraint over the number of sensors controlled, and defines the equation used to calculate *communication\_load*. This allows *normal\_agent* and *robust\_agent* to share these common features. They may of course specify new characteristics in their individual node definitions. In this case their different allowable communication load and cost are encoded in the constraints and constants they define.

```

1 <node type="agent" abstract="true">
2   <param>environment:env</param>
3   <constant name="sensors_controlled">0</constant>
4   <constant name="roles">0</constant>
5   <constraint name="sensors_controlled" op="<=">1</constraint>
6
7   <constant name="message_rr">0</constant>
8   <constant name="message_tb">0</constant>
9   <constant name="message_rb">0</constant>
10  <constant name="message_drq">0</constant>
11  <constant name="message_drr">0</constant>
12  <constant name="communication_load">message_rr + message_tb + message_rb +
    message_drr + message_drq</constant>
13 </node>
14
15 <node type="normal-agent">
16   <param>environment:env</param>
17   <is-a>agent(env)</is-a>
18   <constraint name="communication_load" op="<=">1</constraint>
19   <constant name="cost">5</constant>
20 </node>
21 <node type="robust-agent">
22   <param>environment:env</param>
23   <is-a>agent(env)</is-a>
24   <constraint name="communication_load" op="<=">5</constraint>
25   <constant name="cost">10</constant>
26 </node>

```

**Figure 2.12.** An ODML specification for the agent types in the DSN domain showing how is-a relationships allow characteristics to be shared among derived nodes, and how those derived nodes can be differentiated through their additional local definitions.

### 2.2.2.3 Templates and Instances

Figure 2.10b shows a particular instance of the template from Figures 2.11 and 2.10a. Vertices in the instance graph represent nodes, and a gray directed edge indicates the existence of a non-local modifier from the source node to a field in the target node. Black directed edges represent has-a relationships, but unlike the template they have no labels. Because this is a particular instance of the sensor network organization, the decision points present in the template have all been decided. Therefore, where *sector* might have the *num\_sensors* label on its sensor relationship in the template, a discrete value of two has been chosen for that field in this particular instance. Because of this, each sector in the instance has two sensors (S). Normal agents (a), sector managers (SM), track managers (TM), and two kinds of track manager relations (SM-TM and S-TM), are also present.

We can relate this model directly to the organizational structures discussed in Sections 2.1.3 and 2.1.4. Geographic coalitions are embodied in the *sector* node. The size of the has-a relation *sector* has with the *sensor* node reflects the chosen sector size, and the sector manager is specified with the *sector\_manager* node. The functional differentiation aspect is modeled directly by the *sector\_manager*, *track\_manager* and *sensor* nodes. Each represents a role that can be assigned. This assignment is represented with the *agent* has-a relationship each node possesses. The particular in-

**Table 2.1.** ODML’s built-in functional operators. By convention,  $x$  and  $y$  are simple numeric values,  $\bar{x}$  is a list, and  $D$  is a distribution.

Function	Returns
$min(x_1, \dots, x_n)$	$x_i$ s.t. $\forall j, x_i \leq x_j$ .
$max(x_1, \dots, x_n)$	$x_i$ s.t. $\forall j, x_i \geq x_j$ .
$abs(x)$	The absolute value $ x $ .
$choose(x, y)$	$\binom{x}{y}$ .
$sqrt(x)$	$\sqrt{x}$ .
$round(x)$	$x$ rounded to the nearest whole number.
$map(x, y_1^v, y_1^e, y_2^v, y_2^e, \dots, *, *^e)$	$y_i^e$ s.t. $y_i^v = x$ (see also Section 2.3.1).
$list(x_1, \dots, x_n)$	$\bar{x} = \{x_1, \dots, x_n\}$ .
$listitem(\bar{x}, i)$	$x_i$ , the $i$ th element of $\bar{x}$ .
$size(\bar{x})$	$ \bar{x} $ , the length of list $\bar{x}$ .
$unique(\bar{x})$	$\bar{x}' = \{\forall x \in \bar{x} x\}$ s.t. $\forall (i, j), i \neq j \Rightarrow x'_i \neq x'_j$ .
$forall(y, \bar{x}, e)$	$\bar{e} = \{e_1, \dots, e_n\}$ s.t. $e_i = e$ , and $\forall_{y \in e_i} y = x_i$ .
$forrange(y, x_l, x_h, e)$	$\bar{e} = \{e_l, \dots, e_h\}$ s.t. $e_i = e$ , and $\forall_{y \in e_i} y = x_i$ .
$forallsum(\bar{x})$	$\sum_{x \in \bar{x}} x$ .
$forallprod(\bar{x})$	$\prod_{x \in \bar{x}} x$ .
$forallavg(\bar{x})$	$size(\bar{x})^{-1} \sum_{x \in \bar{x}} x$ .
$forallstddev(\bar{x})$	$\sqrt{size(\bar{x})^{-1} \sum_{x \in \bar{x}} (x - forallavg(\bar{x}))^2}$ .
$E(D)$	The expected value of distribution $D$ .
$V(D)$	The variance of distribution $D$ .
$Pr(D, op, x)$	$\sum_{y \in D} \Pr[y]$ if $(y op x)$ , $op \in \{<, >, \leq, \geq, \neq, =\}$ .
$mc(D, seed)$	$y, y \in D$ , weighted random sample based on $seed$ .

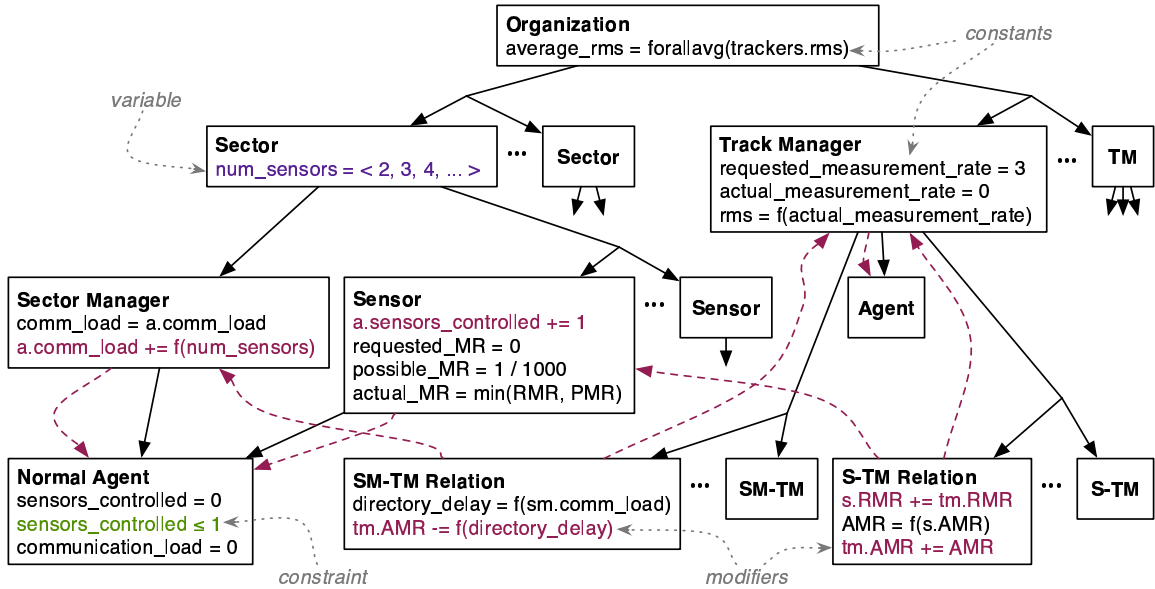
stance of agent node associated with a role node corresponds to the particular agent assigned to that role.

#### 2.2.2.4 Quantitative Expressions

The heart of any ODML model exists in the expressions encoded within nodes’ fields, which provide a way for the designer to represent how different characteristics of the node may be computed. Each field may contain an arbitrary mathematical equation, combining local and nonlocal information to calculate new local values. Expressions consists of a sequence of standard mathematical operations (e.g.,  $+$ ,  $\div$ ,  $x^y$ , etc.) and a limited number of predefined functions (e.g.,  $min$ ,  $max$ ,  $sqrt$ ,  $round$ ,  $forallavg$ , etc.). A list of these predefined functions can be seen in Table 2.1; they consist of a mixture of operators providing new capabilities to the language and convenience functions to simplify the modeling process. The data types supported by ODML are covered in Section 2.2.3.

Figure 2.13 shows another view of some of the information contained within an instance of the ODML DSN model from Figure 2.10b. As before, this is an abstraction





**Figure 2.13.** A graph showing some of the equations and equation-based interactions between nodes that are used to predict RMS tracking error in the DSN model. Solid arrows represent has-a relationships, while dashed arrows between nodes indicate where a modifier is used to propagate values between nodes.

that omits details present in the real model from Appendix C for the purposes of clarity. This figure improves upon those shown previously by depicting some of the quantitative expressions used in the model and how they interact. In particular, it shows many of the hierarchical and lateral data flows that are used to predict the *average\_rms* tracking error of the organization as a whole. I will outline the way this value is computed below, and Section 2.3 returns to several of the ideas that are mentioned in more detail.

The process begins within *organization*, where the *average\_rms* is defined to be *forallavg(trackers.rms)* – the average of the *rms* values for each of its *trackers*. Within the track manager, the *rms* is defined to be some function of its expected *actual\_measurement\_rate* (AMR). This function is well defined in the real model, but omitted for clarity here. The AMR is initially unknown and set to zero, because it depends on other entities in the environment. This initial value will be affected by the relationships the track manager forms, which are discussed below. What is known is the track manager’s *requested\_measurement\_rate* (RMR), which specifies the measurement rate it would prefer to receive.

The RMR is referenced by a modifier in the sensor-track manager relationship that passes this demand level to the sensor. Because there may be many such track managers, each sensor can have a corresponding number of affecting modifiers, which are combined into a single aggregate *requested\_measurement\_rate* within the sensor. Like the AMR of the track manager, the sensor’s RMR is also initially unknown and

set to zero. The sensor itself is a bounded resource, and therefore may not be able to meet all these needs. It therefore also specifies a maximum *possible\_measurement\_rate* (PMR) that is used to determine the *actual\_measurement\_rate* that it will exhibit. An appropriate fraction of this AMR value is delivered back to the track manager by a modifier in the S-TM relationship. Because the track manager may have many such sensor relationships, its local AMR will ultimately reflect the aggregate level of service it can expect.

A similar relationship allows the load of the sector manager to affect the track manager's performance. Recall that the track manager uses the sector manager as an information source. If the sector manager is overloaded, this information may be delayed, which can affect the rate at which tracking measurements may be requested. The SM-TM relation computes a *directory\_delay* based on the sector manager's *communication\_load*, which is used by a modifier to degrade the track manager's AMR. Once these flows have been established, the expected AMR in the track manager may be used to determine its *rms* error. This can then be used to determine the *average\_rms* within the *organization* node.

As another example, Figure 2.11 shows the *track\_manager* and *s\_tm\_relation* nodes, which contain the expressions used to calculate the track manager's logical footprint (area) of a target as it moves through the environment. This is used to reflect the number and location of sensors that might be called on to sense the target. The size of this area will depend on the amount of uncertainty the manager has in the target's location, along with a factor modeling the target's "area of influence" that relates to the area visible to all sensors in a sector, rather than the size of the sector itself. In our model, this area will be a circle; line 15 shows how the *target\_area* of a track manager is derived from the target's *influence\_radius*. The number of sensors presumed capable of sensing the target is the average number of sensor which lie within the target area. Therefore, although the number of *desired\_sensors* is independent of the environment, the *actual\_sensors\_available* to the manager will depend indirectly on the *target\_area* and *sensor\_density*, as shown in line 20. The *requested\_sensors* will be the minimum of the desired and available.

In this way, the characteristics of one node may affect or be affected by those of another. The resulting web of equations allows one to model important concepts such as information flow, control flow, and the effects of interactions. By propagating data through these expressions, the model can predict the characteristics of both individual nodes and the organization as a whole. Perhaps more importantly, it also allows the model to predict characteristics not necessarily envisioned or considered by the designer, as the results of expressions can flow through the graph in unanticipated ways. It is this automatic propagation which differentiates an ODML model from a simple set of equations, by creating a unified view of the complete working organization. A more detailed description of the implemented value calculation process is provided in Section 4.4.1.

### 2.2.3 Supported Data Types

ODML supports three types of data in its expressions, as shown in Table 2.2.

**Table 2.2.** ODML’s built-in data types.

Type	Examples
Float	$x = 1.2$ $y = 5.2e-4$
List	$\bar{x} = [1.1, 3.75, 5.0, 2x + y]$ $\bar{y} = [[x, y], [x + 1, y + 1]]$
Distribution	$d = [\{4.0, 0.3\}, \{-1.0, 0.2\}, \{10.0, 0.5\}]$ . $f = [\{x, 0.25\}, \{x + y, 0.5\}, \{2y, 0.25\}]$ .

### 2.2.3.1 Numeric Data

The simplest and most common type is the numeric floating point value. Most of the expressions shown so far use these single numbers as their primary component.

### 2.2.3.2 Lists

In some contexts, arrays or lists of values or symbols may be used. These are typically operated on using the *forall\**-style functions shown in Figure 2.11 and Table 2.1. Lists are useful for representing, extracting and manipulating sets of similar characteristics. For example, recall that the DSN organization has several *track\_manager* nodes. These are stored in *organization* using a has-a relationship named *track\_managers*. The reference *track\_managers.rms\_error* would create a list containing the *rms\_error* values for each track manager. The function *forallavg(track\_managers.rms\_error)* would calculate the average RMS error value among them.

Lists can be nested, and can contain references to other symbols or built-in functions. The *list*, *listitem*, *size* and *unique* functions in Table 2.1 exist to support list manipulation.

### 2.2.3.3 Distributions

While these two data types are sufficient for most quantitative characteristics, it can be too imprecise to model the uncertainty and variations that exist under realistic conditions. Consider the case where we wish to represent how many targets exist in the environment at a given time. One approach, used in the existing DSN model, is to represent the average case with a single numeric value. However, if there is actually a range of different possibilities at runtime this single number will not capture that fact in a manner that facilitates design evaluation in those varied circumstances. In place of a single value, one could model this condition with a *probabilistic distribution*. This is the third data type ODML supports — a discrete list of values, each with an associated probability. One could then explicitly specify that there was some chance of  $x$  targets and some other chance there would be  $y$ . Section 5.1 will provide additional details on how this feature may be used to improve the level of detail in the

DSN model. Distributions may be manipulated with the common binary operators ( $+$ ,  $-$ ,  $\times$ ,  $\div$ ), and there are also usable by several probabilistic functions, also listed in Table 2.1.

The value provided by this new data type becomes even more apparent when we consider the functions provided by ODML to manipulate and analyze distributions. For example, one can identify best or worst cases with the *min()* and *max()* functions. The expected value of a distribution can be deduced with the *E()* function, allowing one to describe average case performance. The variance of a distribution, computed with *V()*, can quantify how consistent a particular characteristic is. Using this, one might explicitly trade off the potential quality of an organization with its reliability. Finally, one can use the *Pr()* function to determine the aggregate probability of a characteristic compared to a threshold. With this one could create a constraint specifying, for instance, that the probability that characteristic  $x > 10$  must be less than 0.05.

A more difficult problem than syntactically incorporating the distribution data type is how to correctly utilize and propagate distribution information throughout the ODML model. As mentioned above, most of the primitive operators and several functions are able to directly manipulate distributions. This is the ideal case, because the uncertainty represented by the distribution is precisely maintained throughout the calculation process, producing characteristics that are an accurate reflection of the underlying dynamics and uncertainty.

In some cases, however, such direct manipulation of distributions is impossible, because operators and functions do exist in ODML which do not or can not support this type of data. A different strategy is needed to make use of distributions in these situations. One approach supported by ODML is to use weighted sampling to generate individual values from the distribution. The *mc* function in Table 2.1 serves this purpose. The single value resulting from this function may then be propagated as normal numeric data.

Individually, such point values do not capture the original complexity of the distribution. To compensate for this loss, an iterative sample and calculate process is used to approximate the affects of the original distribution. By keeping track of the predicted organizational characteristics during this Monte Carlo (MC) style analysis, one can generate distribution estimates for fields that are unable to use or produce distributions directly.

From a certain perspective, this capability can be seen as a compromise between the strict predictive models described in earlier sections and a true simulation. Each MC trial is analogous to a simulation trial, except that because it is “run” using the ODML model rather than using real agents it can complete in much less time. Evaluating a non-MC model uses the least amount of time, although it can be difficult to capture the variety that will actually be experienced at runtime. A simulation can capture this variety at an extreme level of detail, but typically does so using much more time. The MC approach provides a middle ground between these two, drawing upon the strengths of each to perform a more detailed evaluation in a reasonable amount of time.

For example, consider the situation presented above, where we wish to explicitly capture the fact that different numbers of targets may exist in the environment. The number of targets affects the number of track manager roles that will be created, which will impact the structure of the organizational instance (there will be different numbers of “TM” nodes in Figure 2.10b). Has-a relationships, like that *organization* has with *track\_manager*, require their magnitude be specified as a single numeric value, and do not support the use of the distribution information. Because of this, simply setting the magnitude to a distribution will not suffice. Instead, we may set the magnitude to be  $mc(env.num\_targets)$ , where  $env.num\_targets$  is now a distribution. Because  $mc$  returns a single sample from the distribution, the result can be used as a has-a magnitude. During the evaluation phase, this part of the instance will be repeatedly reformed as different samples are made to determine how the organization will behave under those different conditions.

The interdependent nature of an ODML model makes performing the Monte Carlo (MC) analysis somewhat more complicated. For example, it is possible for the contents of one distribution to be influenced by those of another. It is also possible that a MC sampling exists as part of that interaction, meaning that it is really a sample from the second distribution that influences the first. If a new MC sample is performed on the first distribution, it should be clear that the results of this new sample are dependent on the results of the other. An example of this relationship is captured by the equations below.  $D_2$  is defined using a sample of  $D_1$ , while  $x$  is defined with a sample of  $D_2$ .

$$\begin{aligned} D_1 &= \{0, 0.5\}, \{1, 0.5\} \\ D_2 &= \{0, 0.5\}, \{mc(D_1, S_1), 0.5\} \\ x &= mc(D_2, S_2) \end{aligned}$$

If not correctly compensated for, this dependence can affect the precision obtained by the MC analysis. For example, assume that  $n$  samples are needed to accurately estimate a single independent distribution such as  $D_1$ . A distribution such as  $D_2$  is based on a MC sample of  $D_1$ , and therefore the contents of  $D_2$  will change depending on the outcome of that sample. In this case,  $n$  samples are not sufficient to estimate  $x$ , because the distribution  $D_2$  has an additional degree of freedom imparted by the sample  $mc(D_1, S_1)$  it is dependent on. To compensate for this, each sample of  $D_2$  made for  $x$  must also be evaluated in the context of a representative number of samples of  $D_1$ , the distribution it is dependent on. This so-called hierarchical Monte Carlo approach [198] compensates for the dependence relationship, by ensuring that a sufficient number of trials are performed for the effect of  $D_1$  on  $D_2$  to be captured.

In ODML’s evaluation process, this sampling technique is implemented by manipulating the *seed* argument to the  $mc()$  function (i.e.,  $S_1$  and  $S_2$ ). Pseudocode for this process is shown in Figure 2.14. The *seed\_names* parameter consists of a list of the seeds that must be manipulated. If seeds remain, the procedure creates and saves a random number generated based on that seed, and uses it to produce a seed for

```

perform_mc(organization o, int trials, list seed_names, long seed)
  if (|seed_names| = 0)
    evaluate(o)
  else
    random ← create_random_number_generator(seed)
    seed_name ← seed_names[0]
    next_seed ← random.next_long
    for (trials)
      set_seed(seed_name, random.next_long)
      perform_mc(o, trials, seed_names + 1, next_seed)

evaluate(organization o)
  i ← o.instantiate()
  i.check_constraints()
  i.get_value(utility)

```

**Figure 2.14.** Pseudocode for the hierarchical Monte Carlo trial procedure.

the subsequent recursion. This forces those recursive trials to be consistent for each sample made at the parent level. Within the iterative loop, the seed that will be used by the *mc* function is set repeatedly, after which the function recurses. The recursion terminates when no seeds remain, at which point the organization’s characteristics are evaluated. During the evaluation, a new organizational instance must be created, because the *mc* samples may have caused the shape of the organization to change by impacting has-a decisions. The validity and utility of the instance are then determined and saved (not shown). Upon completion, a total of  $trials^{|seed\_names|}$  sampling trials will have been performed. The accumulated results saved during evaluation will be used to calculate the overall utility and chance of validity for the organization.

## 2.2.4 Limitations and Tradeoffs

As should be apparent, ODML makes heavy use of expressions and equations to model the characteristics of organizations. This provides the important ability to make accurate, quantitative predictions about organizational characteristics. The drawback to this approach is that one is limited to creating models of only those (aspects of) organizations that can be mathematically captured by closed-form equations. If important aspects of a design cannot be directly modeled in this way, and a acceptably precise mathematical approximation cannot be found, it may not be appropriate to use ODML for that design. In the general case, a simulation may be required to accurately predict organizational performance, although doing so typically requires significantly more time and resources than evaluating an ODML instance (see also Section 6). This technique can also be viewed as complementary, where a simulation can be used to refine or validate performance predictions made by an ODML model.

Related to this issue is the potential brittleness of the model itself. For example, it is not uncommon for mathematical models to function with adequate precision on only a range of values, outside of which the predictions no longer hold. This is a real concern, and something that must be taken into account when shaping the space of possible organizations. This can be addressed in part through additional constraints, or by restricting the range of variable fields to those values for which the expressions are known to hold.

ODML's use of an interconnected web of equations also leaves open the possibility for oscillations and infinite recursion when computing values for fields in instances. Such conditions are currently avoided by allowing only acyclic equation relations. This approach has been sufficient for the domains that have been analyzed thus far, although it is not clear that this will be true of all domains. The structure of the ODML representation itself does not preclude cycles from forming, it is the evaluation of values contained along a cycle that creates problems. To support such flows, one would need to specify or find an appropriate way to bound such a computation, so that the affected values may be correctly used elsewhere in the structure. This remains a potential area of future research.

A pair of tradeoffs have been made to reduce the size of the data needed to produce an ODML model and the complexity of the model itself. The first is the lack of a well-defined notion of time, and the second is the absence of a goal tree or task structure. Although ODML models are shown in this work that can and do capture rates of change and expected value, there is no explicit representation of a varying timeline or change points present in ODML. This makes ODML a more tractable structure to work with by dramatically reducing the search space of organizations, but can also make representing some characteristics more difficult. The absence of time means that ODML instances generally represent a snapshot of a running system, an averaging of effects as they would occur over some span of time, or a distribution of possible behaviors that take place. They are more concerned with the steady-state behavior of the system, and less with the transient performance at particular points in time.

If dynamic elements exist in the source environment or system, they may be represented in that same manner, and can still be used to guide organizational decisions. For example, one might identify that sensor resources must be available to satisfy the needs of dynamically discovered targets. By using a worst-case value or a distribution to quantify this need, the organization can be designed appropriately even though the specific situation is uncertain at design time. Section 2.3.5 provides more details on how to represent dynamism under these conditions. Section 5.1.2 also describes an approach that uses ODML's existing syntax to approximate a finite-horizon timeline capable of bridging this gap in some circumstances. Section 4.5 discusses the related topic of dynamic organizational adaptation to new conditions.

ODML also does not have a formal notion of a task environment or goal tree. The approach taken in existing models is to derive suitable abstractions of key characteristics from the task environment, and use those to affect organizational decisions. For example, the task environment for the sensor network broadly consists of a number of targets that must be tracked. Section 2.3.1 will show how this information (target

number and speed) is incorporated into the environmental specification of an ODML model, and later sections demonstrate how it affects organizational decisions. In general, I feel that many high-level task characteristics, such as goal arrival rates and goal-specific features, can be adequately represented in this manner. It is also the case that different sensors may provide different capabilities in service of that task, or that a task may be completed in different ways. ODML models these aspects using has-a and is-a relationships, along with appropriate data flows to determine efficacy. The normal and robust agent variants mentioned previously are one example of this. In general, many of the goal and task characteristics that might normally be represented by formal task decomposition tree can be extracted, analyzed and used to guide the construction of an ODML template, despite the absence of formal support for such constructs in the language. Section 5.2.7 revisits this issue, demonstrating how hierarchies of nodes can be used to approximate goal trees. These can then be used to drive organizational design by using has-a relationships to define task allocations and capture resource conflicts.

## 2.3 Modeling Characteristics in the DSN Domain

Ultimately, the efficacy of any representation lies in its ability to correctly capture the object of its attention. The mechanisms provided by ODML's primitives allows one to model a range of common, organizationally-influenced system characteristics. This section will describe several such characteristics in the context of the sensor network organization, and demonstrate how the interplay between such elements results in a coherent, unified model. A second detailed model will be given for a different domain in Section 3.3 and a number of additional examples described in Chapter 5.

### 2.3.1 Environmental and System Constants

Incorporating numeric constants within an ODML structure, a crucial element of any realistic model, can be at once simple to accomplish and difficult to complete successfully. The definition itself, comprised of a straightforward constant field, is trivial to create. Determining what value to place within this field can be an entirely different matter, just as with the values used in the discrete analytic models.

For example, the *desired\_sensors* constant at Figure 2.11 line 18 is a known quantity that can be extracted directly from agent code or a software engineering specification. On the other hand, the *uncertainty\_radius* on line 13, which represents the expected radius of the target's uncertainty bound, can be more difficult to determine directly. Assuming for the moment that this value does not depend on other characteristics (such as the target's velocity), one could first specify a rough estimate, and later revise that estimate if contradictory empirical evidence is observed in practice or a more accurate value is devised. In practice, most of the numeric constants in the sensor network model were derived through a combination of known system parameters, estimation based on domain expert knowledge, and in some cases, instrumentation of a running system or prototype.



The specification of expression-based constants can be accomplished in a similar fashion, although these are more frequently determined based on knowledge of the system in question. An example of this is the track manager’s *requested\_sensors* in line 21 of Figure 2.11. This represents the number of sensors that manager will actually ask for, which may be less than *desired\_sensors* in the case where there is insufficient sensor density in the environment. It is sometimes the case, however, that a simple closed-form solution is either difficult to derive or not possible. In the former case, I have used curve-fitting techniques to obtain approximate expressions from empirical data. This technique was used to find a predictive expression for RMS error, based on the number of received measurements. For the latter case, when a closed-form solution does not exist or cannot be found, ODML supports a “mapping” function, which allows one to define a function correlating a discrete input value with an arbitrary expression. The general form of this function, which can be specified directly in the ODML node definition is:

$$\text{map}(x, y_1^v, y_1^e, y_2^v, y_2^e, \dots, *, *^e),$$

where  $x$  is the logical input and  $\{y_i^v, y_i^e\}$  is the  $i$ th possible mapping value and expression. The literal  $*$  represents a default or fall-back case. The value of a map function is then defined as:

$$\text{map}(x, y_1^v, y_1^e, \dots, *, *^e) = \begin{cases} y_i^e & \text{if } \exists i \text{ s.t. } y_i^v = x \\ *^e & \text{otherwise} \end{cases}$$

The structure is roughly equivalent to the case or switch structures found in many programming languages. With this, one may define a conceptual function  $f$  such that, for example,  $f(1) = x + \sqrt{y}$ ,  $f(2) = 32z^3$ , etc. The average *effective\_area* of the *sector* nodes used such a function. This characteristic represents the average area covered by the sensors in each sector, which is typically larger than the area of the sector itself. A mapping function was used to effectively create a look-up table, which associates an appropriate expression calculating the effective area for each sector size. The exact specification used for this field can be found within the *sector* node definition in Appendix C.

### 2.3.2 Local Characteristics

Typically, a large portion of the expressions in a given node are devoted to modeling or representing the behaviors that node will exhibit at runtime. This is true of physical components such as *agent*, role or behavioral representations such as *track\_manager* or *s\_tm\_relation*, and the less tangible logical or conceptual nodes such as *sector* or *organization*. In each case, a set of fields will be defined to capture the relevant characteristics of the node they inhabit. For example the *agent* node has a *communication\_load* constant that represents the amount of communication that agent will experience. It also has a *communication\_messages* constant, based on *communication\_load* and *time* that determines the total number of messages the load will produce during the specified period of time.

The set of characteristics defined in a node is arbitrary and left to the discretion of the designer. There are no predefined characteristics that are assumed to exist. For instance, *agent* need not have a *communication\_load*, and could just as easily omit it, or have a fine-grained *message\_byte\_total* or a vague *verbosity* field in its stead. In the DSN model, the *agent* node also includes *role* and *sensors\_controlled* constants. The former is used to track how many roles an agent is assigned, while the latter tracks how many sensors are under its control. It should be clear from this example that both domain-independent and domain-specific characteristics are present. Section 5.2 shows how inheritance can be used to create domain-independent templates or libraries that can be augmented with domain-specific information, which can be used to leverage a set of common models across a range of domains.

Notice how many of these “local” characteristics are in fact determined almost exclusively by remote entities. For example, because an *agent* is initially unbound, the *role* and *sensors\_controlled* constants are both initially set to zero. The same is true of the *actual\_measurement\_rate* values for *track\_manager* and *sensor* from Section 2.2.2.4. They only take on non-zero values when the agent is incorporated into an organizational context that affects them (e.g., it is assigned a role or forms a relationship with another entity). Characteristics can be top-driven, as when a role is assigned to *agent*, laterally-driven, as when a *sensor* is used by a *track\_manager* to collect data, or bottom-driven, as when the characteristics of *sector* are determined by its contents. In each case, local definitions provide a baseline expression, and modifiers, parameters or remote values are used to affect the value that expression produces.

### 2.3.3 Entity Interactions

The manner in which entities interact is perhaps the most visible and defining characteristic of multi-agent systems. It generally plays a crucial role in determining how information flows through the system, how load is distributed, how efficient operations are, and ultimately the effectiveness of individual agents and the system as a whole. In our sensor network model, these interactions are defined in two different ways. In the first, entities simply model the effects of interactions internally. For example, the *sensor* node calculates the directory service messages it sends to its sector manager, and uses a modifier to add a corresponding number of messages to the *agent* it is bound to.

The second technique employs a more explicit representation by creating an independent, external node to model the interaction. An example is *s\_tm\_relation*, the sensor-to-track manager relation, shown in Figure 2.13 and line 25 of Figures 2.11. This node models the interactions that take place between a track manager and a sensor, which include determining the rate at which task requests are generated by the track manager, the rate at which measurements are taken in response to those requests, and the rate at which corresponding results are sent back to the track manager. Each of these values is calculated using a combination of information from each entity, and the results applied back to the appropriate node. For example, the *ac-*

*tual\_measurement\_rate* is used to increment the sensor's *messages\_rr* constant, which tracks the number of result messages that are sent.

Choosing how to model an interaction depends on a number of factors. Simple interactions are typically embedded, thus avoiding the additional overhead associated with node creation. I have found that there are several ways that more complex interactions benefit from an explicit, separate model. By separating and encapsulating the interaction, its effects can be made more transparent and the model more understandable. In the case where there is a one-to-many type of relation, as with a track manager and the sensors it uses, this type of separation also facilitates the expression writing process by limiting the scope that individual equations must cover. Instead of using a single complicated expression in the source node, for example, that model defines a single simple expression in the relationship node that is replicated for each target. In this case, the track manager-sensor relationship simplifies the process of propagating sensing demands and responses between the two entities.

In the case where several alternative interaction styles are available, the explicit representation also allows the designer to use variables or inheritance to model and reason about such choices. For example, if our track managers had two different ways of requesting measurements from a sensor, those alternative interactions could be modeled as *s\_tm\_relation1* and *s\_tm\_relation2*, each defined as an instance of *s\_tm\_relation* with an is-a relationship. When an instance of *track\_manager* is created, one of those two alternatives would be selected for each *sensor*, and the corresponding effects incorporated appropriately. In this way, in addition to representing the quantitative effects of interactions, the selection of agent interaction or coordination mechanisms may be cast as an organizational decision in ODML.

### 2.3.4 Multiple Role Assignments

In human organizations, individuals frequently act in many different capacities, serving different needs and exhibiting different behaviors depending on the working context. In some complex multi-agent systems, similar phenomena may be observed, where individual agents take on multiple roles that dictate the various responsibilities, capabilities and activities it is associated with. Because the assignment of these roles to individual agents is an organizational decision, it is important to be able to both represent the assignment itself and the cumulative effects of that decision.

As mentioned earlier, there are three roles in our distributed sensor network organization: the sector manager, the track manager and the sensor. These are represented by the *sector\_manager*, *track\_manager*, and *sensor* nodes, respectively. Role assignment is modeled through the use of a has-a relationship. Specifically, each of these role nodes has an *agent*, as shown in Figure 2.10a, that represents the particular agent that role is assigned to. During instantiation, has-a relations may be fulfilled in two different ways. Either a new instance of the target node is created to satisfy the relationship, or an existing instance of the node is used in the same way. An example of the latter can be seen in Figure 2.13, where the normal agent on the bottom left is owned by both the sector manager and sensor, indicating that particular agent has been assigned to two roles.

```

1 <node type="sensor">
2   ...
3   <modifier name="agent.message_rr" op="+">message_rr</modifier>
4   <modifier name="agent.message_tb" op="+">message_tb</modifier>
5   <modifier name="agent.message_rb" op="+">message_rb</modifier>
6   <modifier name="agent.message_drr" op="+">message_drr</modifier>
7   <modifier name="agent.message_drq" op="+">message_drq</modifier>
8
9   <modifier name="agent.sensors_controlled" op="+">1</modifier>
10  <modifier name="agent.roles" op="+">1</modifier>
11 </node>

```

**Figure 2.15.** A portion of the ODML specification for the sensor node showing how message load values, number of sensors controlled and role indicators are passed to the *agent* with modifiers.

Most of the detailed characteristics in this model are computed within the role nodes. Therefore, important aspects such as load and resource usage are inherently separate and role-specific. To capture the effects of multiple role assignments, these individual characteristics are first propagated into their relevant *agent* using modifiers. Each agent can then predict the cumulative effects of its roles. A natural example of this in this is the propagation of communication effects. Each role has an expected communication load that is determined within the role node. These are maintained as individual statistics for each message type. These values are then routed to the appropriate *agent* with set of modifiers. This is shown in Figure 2.15 in the set of the *message\_\** modifiers. Figure 2.12 shows how those individual values are used to calculate the *agent's* overall *communication\_load*.

Referring back to Figure 2.13, one can now see that the *communication\_load* of the agent will be affected by both the sector manager and sensor roles it has been assigned. Therefore, when the SM-TM relation uses the sector manager's communication load to calculate the track manager's *directory\_delay*, the potentially negative consequences of this role coexistence will be correctly accounted for. This is an example of how the consequences of messaging disparity from Section 2.1.4 can be modeled.

This confluence within the *agent* node also provides a useful place to incorporate constraints. For example, a design assumption in the original system said that each agent would be associated with a single sensor. In the ODML model, each *sensor* role uses a modifier to increment the *sensors\_controlled* field of its *agent*. The design assumption may be embodied by placing a constraint in the agent, specifying that *sensors\_controlled* must be equal to 1, which guarantees that all agents in a valid organization will control exactly one sensor. This is shown in line 9 of Figure 2.15, as well as in the corresponding nodes in Figure 2.13. A *roles* count is maintained in a similar way, so that an analogous constraint on the *roles* constant inside the *agent* node could be put used to control how many roles an agent can be assigned. A similar approach could assign a constraint on maximum communication or processing load, which tie role assignment to a more concrete metric. Conversely, by adding a constraint defining a lower bound on load, we can make the selection process more conservative by ensuring all created agents see a certain minimum level of work.

### 2.3.5 Dynamic Behaviors

In reactive or adaptive systems, roles are frequently created dynamically in response to emerging phenomena. Such is the case with the track manager role, which is assigned only when a new target has been detected in the environment. Although ODML does not have a strong notion of time, it is still possible to represent behaviors or characteristics that occur dynamically. For example, although at any given point in time there may be many or few targets in the environment, there will be some expected number of targets that represents a statistically average value. This number would then be used to estimate the “normal” situation, and be reflected in the model accordingly. If one is interested in worst-case scenarios, other assumptions could be made to model such conditions. In both cases, some discrete value may be used to instantiate a set of nodes that represent the results of dynamic events. The *num\_targets* field in *environment* serves this purpose in the DSN model, by governing how many track managers should be represented in an organizational instance.

Additional role-based dynamism is present in the sensor network example, due to the migration of the track manager role as described in Section 2.1.1. When this role moves to maintain locality with its target, the effects of that role are effectively spread over multiple different agents. To represent this effect, the model uses the target’s velocity and the sectors’ sizes to first estimate how frequently that role will migrate. Because this is a rate, it must be combined with the duration of the scenario to determine the number of agents that role will be assigned to. This number is then used to calculate *num\_agents*, which is used to specify the size of the track manager’s *agent* has-a relationship (line 9 of Figure 2.11). So, if the model predicts that the track manager role will be created and then migrate twice, the *num\_agents* field in *track\_manager* will be set to three, and three *agent* nodes will be created or selected to satisfy those relationships. The role’s relevant characteristics are divided and distributed evenly among those three agents using modifiers as described in the previous section. The principal difference being that the values propagated by the modifiers shown for the *sensor* role in Figure 2.15 are each divided by *num\_agents* in the *track\_manager* role.

When the role is migrated, a small gap in the stream of measurements will be experienced because of the time it takes to make the hand off. This *migration\_delay* is used to influence the *rms\_error* experienced by the track manager, by reducing the *requested\_measurement\_rate* by an appropriate amount.

The dynamics of the agent’s actions or environmental conditions can also dictate organizational performance. For example, the velocity of the target also affects the *uncertainty\_radius* of the track manager’s location estimate. This uncertainty is similarly influenced by the quality of the measurement fusion and interpretation process. The faster the target is moving, or the more unreliable the fusion process is, the larger the uncertainty bound will be. I mentioned earlier how uncertainty affects the target’s “area of influence”, determining how many sensors would be considered viable at a particular instance in time. This will decide how many sector managers must be notified of the target’s position as it moves, affecting the communication load of both the track and sector managers.

### 2.3.6 Heterogeneity

An important advantage that ODML offers over simple analytic models is that heterogeneity is more readily representable. For example, an equation was created in [84] to calculate the total number of measurements that would be produced for a track, but this assumed that all sensors would produce measurements at equal rates. Similarly, a different estimation of a sector manager's load assumed that all targets moved with equal velocity. Neither of these simplifying assumptions are likely to be true in practice, so to the degree the unified model can represent such additional information, it will have a decisive advantage in accuracy.

ODML's ability to model heterogeneity is derived from the the node-based representation of entities. Because each role, agent or other structure is defined as a separate node in the organizational instance, entities that share a common type may still contain different values or be affected by different organizational pressures and flows. We have seen examples of this in the previous two sections. Because agents may be assigned single or multiple roles, the resulting agent population has the potential to be heterogeneous in the final organization. These variations may then propagate through the organizational instance to create differences elsewhere in the model. For example, communication load can be tied to the agent's capacity to perform work, which could affect the number of measurements its sensor role could take, which would affect the RMS error of the track manager using that sensor.

Inheritance provides an additional mechanism to represent heterogeneity. For example, to describe and use the more capable but also more costly agent mentioned earlier, the model contains a *robust\_agent* node that has an is-a relationship with *agent*. This effectively creates two different classes of agents that can be employed, each with potentially different capabilities and costs.

### 2.3.7 Conflicts, Constraints and Resolution

Many of the more interesting aspects of organizational models revolve around the limits or constraints that are imposed on the system, and what happens when those limits are approached or exceeded. ODML models can represent both hard and soft constraints. The former include conditions which the designer has deemed untenable, while the latter are usually characteristics that degrade more gradually, and may be tolerated by the system.

Hard constraints may be modeled using constraint fields. A constraint is defined with a target, a relational operator, and an expression. To verify the constraint, both the target and the expression are evaluated to produce numeric values, which are then compared with the provided operator. The constraint is considered satisfied if the resulting relation is true, and unsatisfied if otherwise. Because a valid organization must contain only satisfied constraints, these constraints are considered hard, or strict conditions that must be met. I mentioned earlier how a constraint on the *sensors\_controlled* field in the agent ensured a one-to-one mapping between sensors and agents. Similar constraints could be added to set an upper bound on average expected RMS, a limit on local work load, or a maximum number of agents in the

```

1 <node type="track_manager">
2   ...
3   <constant name="directory_delay">forallavg(sm_relations.directory_delay)</
   constant>
4   <constant name="requested_measurement_rate">env.measurement_rate * 4.0^min(0,
   velocity * 1000 * ( (env.num_sensors / (requested_sensors * org.
   total_targets)) - 1.9))> (1 - (velocity / average_sector_path) * (
   directory_delay + migration_delay/2))</constant>
5 </node>
6
7 <node type="sm_tm_relation">
8   <param>organization:org,track_manager:tm,sector_manager:sm</param>
9   ...
10  <!-- Directory Delay -->
11  <constant name="directory_delay">3000</constant>
12  <constant name="per_message_delay">0</constant>
13  <modifier name="directory_delay" op="+">sm.communication_load * 1000 *
   per_message_delay</modifier>
14 </node>

```

**Figure 2.16.** A portion of the ODML specification that estimates the delay in directory responses that is experienced by a track manager. The *sm\_tm\_relation* nodes determine the individual delays for each sector manager, while the *track\_manager* uses these values to estimate a worst case delay that is used to reduce the requested measurement rate.

organization. Although our sensors were hard-wired, a battery-driven sensor network could also be modeled by adding a suitable constraint to the agent. In this case, communication rates, action rates or the passage of time could decrement a battery constant. A constraint placing a lower bound on the battery would ensure that the unit met an expected minimum performance.

Soft constraints have a more subtle effect on the system. They are not explicitly modeled using the constraint field. Instead, we represent them using equations that affect performance in response to other attributes. For example, in the sector manager, excessive communication load can delay directory service responses. This is modeled with a *directory\_delay* field, which is then used to determine the values for delays incurred by sector directory queries and tracking task migration. Increases in those values will eventually increase the RMS error by slowing the rate at which the track manager acquires new sensor information.

The portions of model code which reflect this are shown in Figure 2.16. This is represented in the model by using a modifier to increase the *directory\_delay* in each *sm\_tm\_relation* by a value proportional to the sector manager's *communication\_load* (line 13). The maximum directory delay observed by the track manager is then used to calculate the *requested\_measurements\_rate*. Therefore, although there is no fixed, arbitrary limit on sector manager load, excessive load will still degrade the system's performance. A breaking point, at which the performance level has become untenable, can still be modeled using a hard constraint governing the value in question.

Soft constraints are also used to model the competition for sensors by track managers, as outlined in Section 2.2.2.4. The *s\_tm\_relation* in Figure 2.11 notifies each sensor of the *requested\_measurement\_rate* that will be asked of it. If the requested

rate exceeds the sensor’s capabilities, the *actual\_measurement\_rate* given to the manager will be correspondingly limited. This will negatively affect the expected RMS error. Modifiers from multiple, competing managers requesting a sensor’s time will each increment the *requested\_measurement\_rate*. If the sensor’s measurement rate is exceeded, each manager’s *actual\_measurement\_rate* will be reduced to some fraction of the possible rate proportional to what was asked for. This is accomplished as follows:

$$\begin{aligned}
s\_tm.RM\_rate &= tm.RM\_rate \\
s.RM\_rate &\stackrel{+}{\leftarrow} s\_tm.RM\_rate \text{ (modifier)} \\
s.AM\_rate &= \min(s.RM\_rate, env.MD^{-1}) \\
s.AM\_ratio &= s.AM\_rate/s.RM\_rate \\
s\_tm.AM\_rate &= s\_tm.RM\_rate \times s.AM\_ratio \\
tm.AM\_rate &= s\_tm.AM\_rate
\end{aligned}$$

*RM* refers to requested measurement, *AM* to actual measurement, and *MD* to the expected duration of a measurement. So, although there is no set limit on the number of track managers, a “tragedy of the commons”-like degradation in quality can be predicted from overuse [192]. This same scheme can also be used to model notions of authority, autonomy and priority [6]. If multiple agents have disparate authority over another, then those differences can be reflected in the amount of utility they derive from that relationship. For example, if multiple track managers each had different authority over a particular sensor, then their relative levels of authority would determine the proportion of time that sensor would spend on their tracking tasks:

$$\begin{aligned}
s\_tm.RM\_rate &= tm.RM\_rate \\
s.AM\_priority &\stackrel{+}{\leftarrow} s\_tm.RM\_priority \text{ (modifier)} \\
s.AM\_rate &= env.MD^{-1} \\
s\_tm.AM\_rate &= \min(s\_tm.RM\_priority \times \\
&\quad s.AM\_rate/s.AM\_priority, \\
&\quad s\_tm.RM\_rate) \\
tm.AM\_rate &= s\_tm.AM\_rate
\end{aligned}$$

In the current model, all track managers implicitly have the same priority or level of authority, but as was discussed in Section 2.3.6, adding this type of heterogeneity among managers would can be accomplished in ODML.

### 2.3.8 Organizational Utility

Early on, an assumption was made that different organizations can induce different behaviors and characteristics in a working system. Empirical evidence of this was

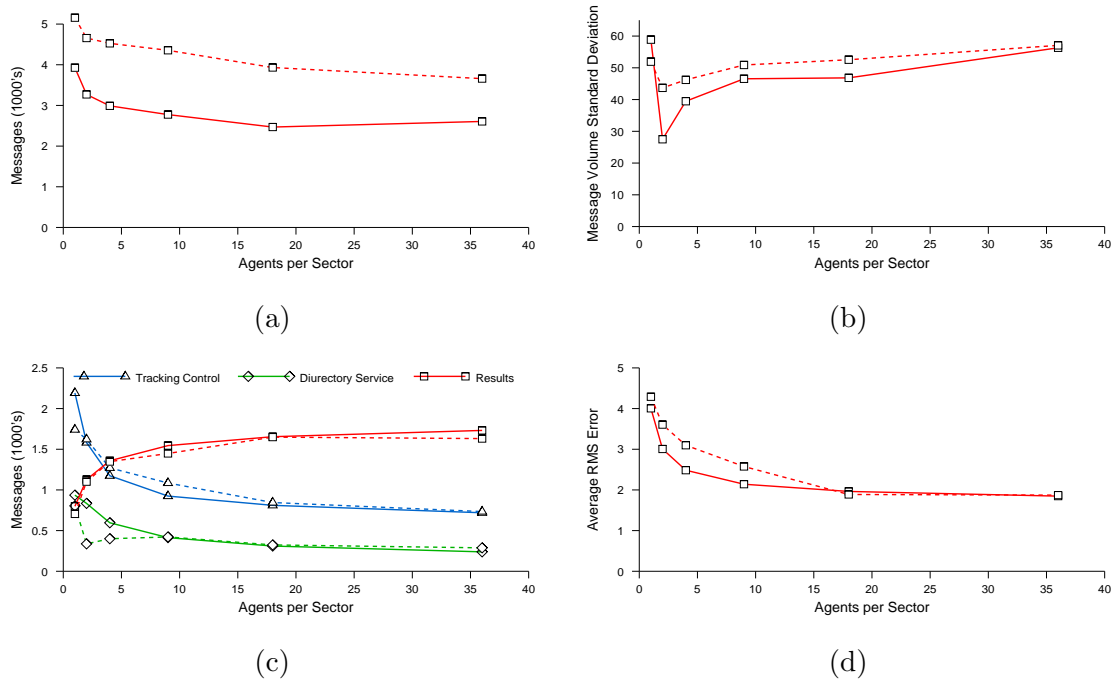


presented in Section 2.1, and it will be shown in the following section that the ODML model reflects this as well. Because the characteristics vary, it is natural to assume that some characteristics will be preferred over others. There will therefore be a comparative ranking of organizational designs, dependent on how these characteristics are affected by the environment, available resources and high-level objectives.

The expressions embedded in an ODML model provide a straightforward way of representing the relevant characteristics. Earlier sections outlined how features such as individual agent load, delays incurred by organizational maintenance and RMS error can all be represented. Evaluating the utility of an organization can be accomplished by creating an expression that compares these characteristics in a suitable way. For example, in the sensor network model, the primary concern is to minimize the RMS error associated with tracking targets, while respecting any constraints that exist in the agents or environmental resources. The previous section described how constraints can be represented and satisfied, so a particular DSN design can be rated by the expected RMS error the track managers will exhibit. This is done by creating a *utility* constant. Like any field, *utility* can contain an arbitrary expression. In this case, it is defined to be  $-1 \times average\_rms$ , where *average\_rms* is the average expected RMS error for each track manager. Because the value for this expression will be negative, lower RMS values will lead to greater utility. If we wanted to rank it based on the weakest track manager, it could instead be defined as  $-1 \times minimum\_rms$ , where *minimum\_rms* is defined as the minimum expected RMS among the track managers. The utility of the organization can then be determined from the value of the *utility* field, which automatically takes into account all the relevant organizational characteristics and decisions that were made.

Earlier it was mentioned that one might want to consider some notion of organizational cost. For example, a *robust\_agent* might cost more to deploy than a *normal\_agent*, because of the increased local processing resources that are needed. To model this, a *cost* constant could be included in each agent type reflecting this information. These costs could then be propagated up through the organization in the same way RMS values are, summed, and added to the utility calculation. If RMS were paramount and cost secondary, a weight could be simply be added to the utility function to reflect this. For example, *utility* could be defined as  $-1 \times (average\_rms + 0.01 \times total\_cost)$ . In this case, organizations that performed similarly in terms of RMS would be ranked according to their cost.

Although it has yet to be tested, I believe ODML also has the ability to determine less tangible measures of utility, such as the ability to handle faults or deviations from expected environmental characteristics. For example, because it is possible to model an upper bound on an agent's computing power as well as its expected computing load, one can also compute the difference in these two values. By aggregating these values from all agents, one can measure the excess computing power, which will influence how well the system might tolerate more demanding operating conditions. Similarly, one could determine the excess number of agents that have particularly critical capabilities, which reflects how well the system might be able to adapt to failures in those critical agents. It is also possible to use an external process or even domain-specific expert knowledge to grade and evaluate the utility of candidate



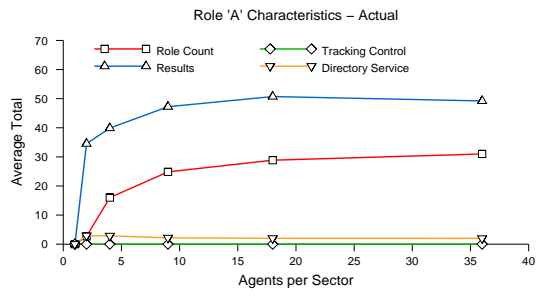
**Figure 2.17.** Performance predicted by the ODML sensor network model versus empirical observations for a) Total messaging, b) Messaging disparity, c) Message type totals and d) RMS error. Predicted lines are solid, empirical are dashed.

organizations, although I will not explore their use in this thesis. More information can be found in Chapter 5, which will present additional ways to model utility in different circumstances.

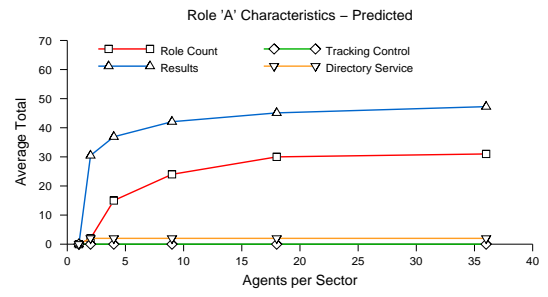
### 2.3.9 Evaluation of Representation

To gauge the efficacy of the ODML representation, I have constructed the model described in the previous section, used it to create organizational instances that match the prior test runs, and compared the predicted characteristics against the empirical results from Section 2.1.2. Because time-based characteristics in the ODML model (such as communication load) are computed as rates rather than totals, the values are not directly comparable. However, cumulative totals may easily be computed by multiplying the relevant rate by the length of the prior simulation run. Ideally, the empirical and modeled results will match, demonstrating that the model captures the complexity present in the system and that predictions derived from the model are accurate.

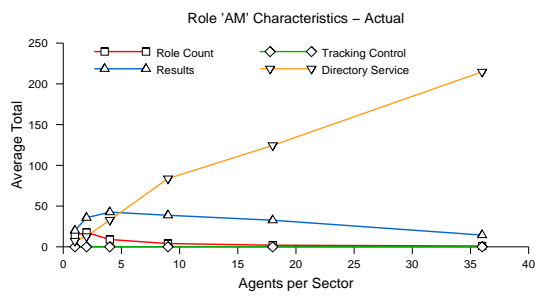
A relatively high-level set of comparative results are shown in Figure 2.17 that contrast the predicted results against some of the actual, empirical results shown earlier in the paper. Solid lines represent the values predicted by the ODML model, while dashed are those obtained through empirical testing. Although there are a few



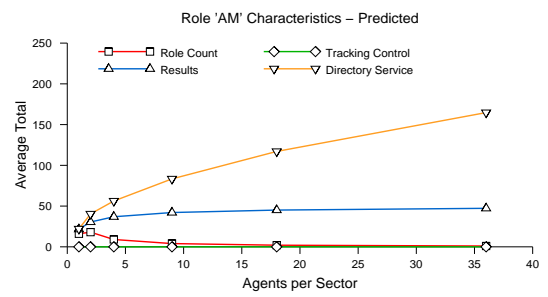
(a)



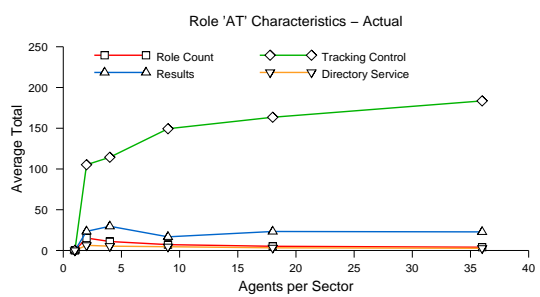
(b)



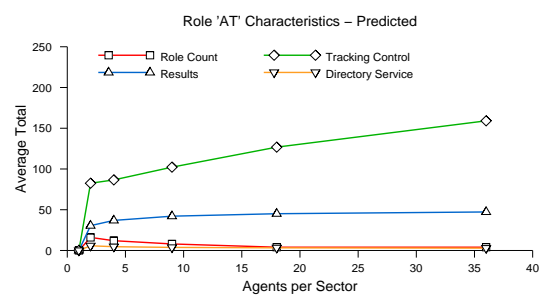
(c)



(d)

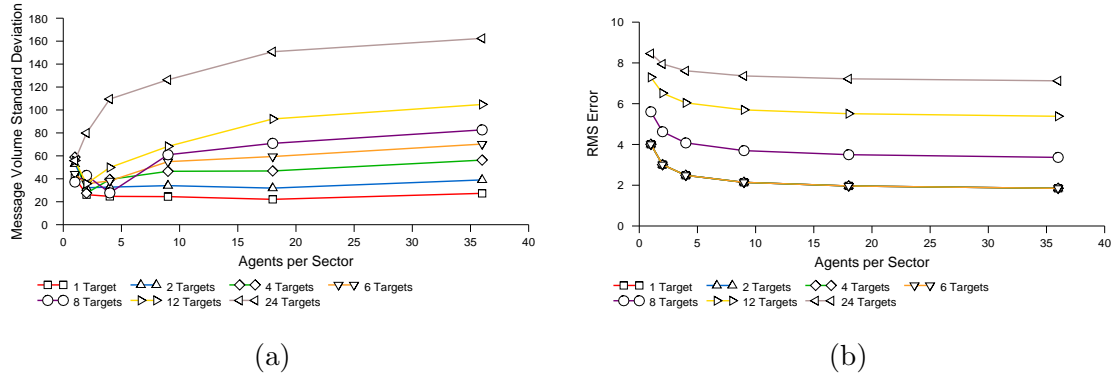


(e)



(f)

**Figure 2.18.** A comparison of the average actual and model-predicted characteristics by role, for agents operating in the distributed sensor network.



**Figure 2.19.** Performance predicted by the ODML model as the number of targets is varied for a) Messaging disparity, and b) RMS error.

significant points of difference, in most cases the model does a good job predicting performance.

One difference can be seen in Figure 2.17b, where the predicted standard deviation underestimates the actual performance in most cases. This is a byproduct of our assumption that all sensors were equally used. In the running system, sensors in the center of the environment are used more than those at the edges, and will have different communication profiles because of it. This model does not capture these geographic differences, and will therefore generally have a lower estimated deviation. To accomplish this, individual sensors would need to be differentiated by their place in the environment, using an internal location constant or through specification by their containing sector. The location information could then be used to estimate the type of geographic-specific behavior observed in the real system. The resulting model would be more accurate, but it would also be more complex and difficult to create, because of the need to correctly propagate this new information to all affected areas of the model. The decision to omit location information was an intentional tradeoff made to simplify the model. This issue will be revisited in Chapter 5.

A more obvious difference can be seen between the overall message totals, in Figure 2.17a. This difference can be attributed to the fact that the empirical values included all 24 message types that occurred in the system, while the model only tracks the five most significant message types. Combined, those five types constituted roughly 80% of the communication volume on average. The remaining 19 were uncommon; no individual type accounted for more than 3% of the total. As can be seen in Figure 2.17a, the difference between predicted and empirical remains relatively constant with sector size, and could be accounted for by adding a suitable constant to the model. The exclusion of these message types was a conscious choice on our part. It is an example of trading off the complexity of the model with its fidelity.

Recall that one of our initial goals was to predict organization-level characteristics of the system. The metrics I have shown so far accomplish this, but primarily

on a global, aggregate level. To evaluate how our model predicts finer-grained details, I produced a separate set of graphs that show communication profiles by role, rather than the system-wide totals seen in Figure 2.17c. The actual and predicted role-specific graphs can be seen side-by-side in Figure 2.18. In addition to the communication totals I have discussed, these graphs also include role counts, indicating how many agents did or would take on that role in the environment. 'A' represents the *sensor* role, 'M' is the *sector\_manager*, while 'T' is the *track\_manager*. The role 'AT' describes agents acting as both sensors and track managers. These results are also encouraging.

By and large, the model's predictions are similar to prior observations. Some of the differences, such as the result totals for some sector and track managers, can be attributed to geographic variances in a small sample size. For example, the 36- and 18-size sector scenarios had only one and two sector managers, respectively. Their individual geographic locations would certainly affect the actual averages in Figure 2.18c, and these variations are not reflected in the counterpart predicted values in Figure 2.18d.

The last set of ODML predictions are shown in Figure 2.19, which examines messaging disparity and RMS error as both the sector size and the number of targets are varied. Figure 2.19a corresponds to the empirical results shown in Figure 2.8a, while 2.19b corresponds to Figure 2.8b. The trends shown in those earlier figures are similar to those predicted by ODML, including the same general profiles and the RMS phase transition.

## 2.4 Conclusions

This chapter began with a description of an existing and functional distributed sensor network system. Through a series of empirical tests and analyses, I showed how the organizational design employed by the system has a measurable impact on performance. Results such as these and others that are shown in later chapters motivate the organizational design problem this dissertation is addressing. The ODML language defined in Section 2.2.1 takes a fundamentally different approach to solving this problem by offering a simple but quantitatively rich framework in which organizational characteristics can be modeled. Unlike previous representations, ODML eschews predefined structures and assumptions in favor of a general mathematical syntax. I believe this approach leads to an increased diversity of representable situations as well as an increased level of predictive detail. The model of the distributed sensor network system described and validated in Section 2.2.2 shows how this has been successfully used.

The result of the DSN modeling exercise is to demonstrate that it is possible to create quantitative organizational models in ODML that accurately predict large and small scale performance. Such models can be used to find and evaluate candidate organizations or identify design weaknesses. More generally, this chapter demonstrates that the flexible and quantitative approach ODML employs can be an effective way to capture the behaviors of a realistic organization in a concrete and detailed way.

The following chapter will show how these same capabilities are true for a different domain that allows a greater diversity of organizational designs and requires more sophisticated techniques to accurately model. Chapter 4 will then show how these modeling capabilities can be used as the foundation for techniques that automatically design organizations for agent systems.

## CHAPTER 3

### MODELING AN INFORMATION RETRIEVAL SYSTEM

The distributed sensor network (DSN) model presented in Section 2.2.2 is an example showing how ODML has been used to capture the behaviors of a complex, working multi-agent system. What the DSN model lacks, however, is the vast space of possible organization that one can find in less constrained designs. In particular, the actual DSN system has relatively few degrees of freedom (sector size, role assignment, etc.), resulting in a relatively small space of organizational possibilities. This chapter presents a detailed model of a second domain that permits a much greater variety of organizational structures. This is used to ground the discussion of automated organizational design that follows in Chapter 4. It also complements the DSN model by providing a second detailed example of ODML being used to effectively and accurately capture different types of organizational characteristics.

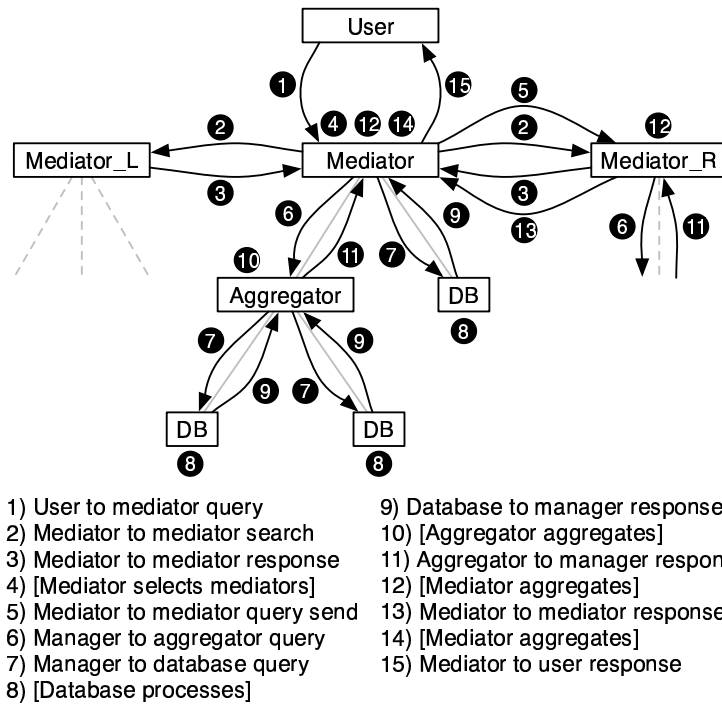
Section 3.1 describes how ODML is used to create a model of an information retrieval (IR) system. This includes a description of the problem domain and the class of organizations that are proposed to address that problem. A sample trace through the system's behaviors is given.

Section 3.2 describes the simulation environment created to model this domain. It is used to empirically evaluate the approaches and assumptions used in the ODML model.

A description of these characteristics and the methods used to model them is given in Section 3.3. These include a probabilistic model of different search and query effects (Section 3.3.3), a detailed model of response time based on queuing theory (Section 3.3.4) and examples of how a utility metric is used to guide the selection of an appropriate design (Section 3.3.6).

### 3.1 Information Retrieval Model

The IR model presented in this chapter is inspired by work by Zhang et al. [214, 215]. A general peer-to-peer information retrieval system is composed of a number of interconnected databases, controlled by a set of (agent) entities. Queries are first received by individual members of the network. An appropriate set of information sources must then be discovered that can address the query, after which the query is routed and processed to produce a response for the user. The information necessary for responding to a particular query may be distributed across the network, which can cause an undirected retrieval process to be time consuming, costly, or ineffective, particularly when the number of sources is large.



**Figure 3.1.** The control and communication sequence involved in handling a query in the information retrieval organization. Steps in the numbered trace correspond to messaging events, except for bracketed elements that indicate local processing.

Zhang proposes that a structured, hierarchical organization can be used to address this problem. Content in the network is arranged in hierarchies, allowing queries to quickly propagate to data sources, and results be efficiently routed and incrementally aggregated back to a single agent in the network. At the top level of the hierarchy are a set of mediators. Each mediator is responsible for providing a concise and accurate description, known as a collection signature, of the data available in its hierarchy. A hierarchy forms below each mediator, which contains and manages a collection of information sources. An information source may be an individual database, or an aggregator agent which manages other sources. This self-similar relationship allows the hierarchy to grow arbitrarily wide and tall. Mediators are responsible for handling the user queries, by first using the collection signatures of other mediators to compare data sources, then routing the query to those mediators that seem most appropriate, and finally collecting and delivering the resulting data. This model slightly diverges from Zhang's in that it takes into account the query and response aggregation work load and omits any lateral connections between aggregators and databases.

Figure 3.1 shows an example trace of how a particular organization using this approach processes a single query. The process begins when a user query arrives at a mediator (1). The mediator then queries a number of other mediators to determine if they are appropriate to handle the user query (2). After the responses are sent (3) and collected, a subset of those searched are selected based on their reported



collection signatures (4) and notified that they should handle the query (5). In this case, Mediator\_R was selected, while Mediator\_L was not. The user query is then propagated in parallel down all branches of the mediators' hierarchies (6, 7), until the terminal databases at the leaves are reached. Each database processes the query (8) and reports it back to its immediate manager (9)<sup>1</sup>. Intermediate aggregators will wait until all subordinates have responded, and then consolidate the results (10) before delivering the information up the next level in the hierarchy (11). Mediators perform a similar consolidation step (12). Any mediators that were selected to handle the query report their results back to the originating mediator (13), which performs a final consolidation step (14) before delivering the final response to the user (15).

This organizational design provides several advantages. The use of collection signatures to model the contents of a number of individual sources can dramatically reduce the number of agents that must be searched and queried. The use of hierarchies introduces an element of parallelism into the query distribution process. These same hierarchies also distribute the communication and processing load of the response through the use of information aggregation and consolidation.

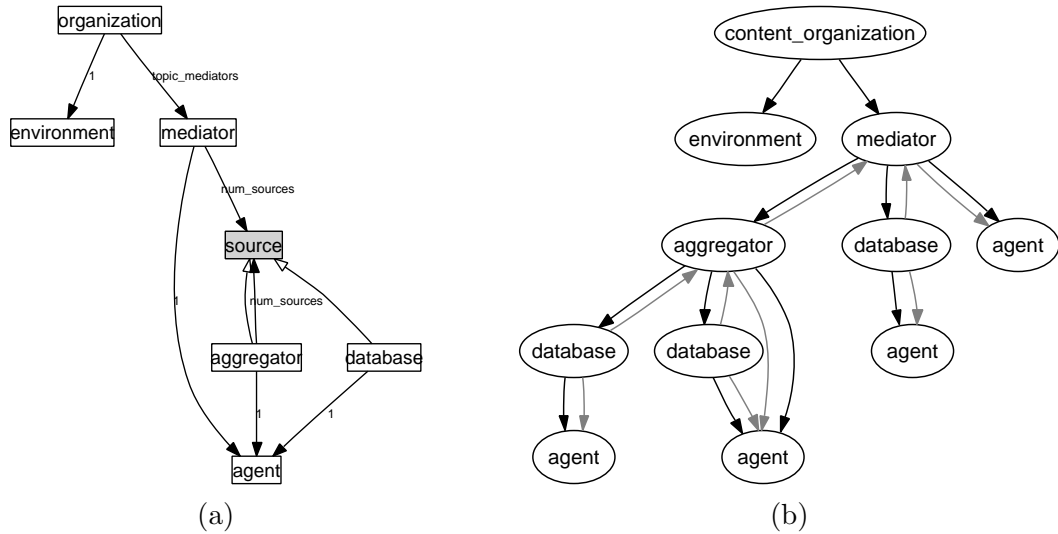
At the same time, if the structures are poorly designed they can lead to inefficiencies. A single collection signature, which must be bounded by size to be efficiently used, can become unacceptably imprecise if the set of sources it models is large or extremely diverse. This can cause data sources to be overlooked, potentially reducing the response quality. If the data sources are distributed across many different mediators it may require a more extensive search and query process to obtain a high quality result. Whenever a hierarchy is used, there also exists a tension between the width and height of the structure. Very wide structures can lead to bottlenecks, as particular individuals with high in-degree may become overwhelmed by the number of interactions. Very tall structures can be slow or unresponsive, as the long path length from root to leaf increases latency. The collection signature generation process may also be affected by the tree height, as when abstraction is used at intermediate nodes, causing the signatures of tall hierarchies to incur additional imprecision.

An ODML template for this domain can be seen in Figure 3.2a, and a particular instance derived from that template is shown in Figure 3.2b. Like the DSN model, this uses notions of roles, a task environment, performance constraints and utility functions. However, other underlying phenomena that must be captured are significantly different, and drive the shape of the organization in different directions. These include statistically predicting the results of the source discovery process, determining how the information contents of a hierarchy affect its expected load and approximating the effects of increased signature uncertainty caused by summarization [82].

Additional constraints and characteristics exist in the system that exist independent of the organization that is employed, but are relevant to the organization selection process. The communication and processing loads of individual agents are

---

<sup>1</sup>Note that despite the numbering differences, activity down the tree branches occurs in parallel, so that the mediator message to the aggregator (6) will happen concurrently with the mediator's message to the DB node (7) that exists at the same level.



**Figure 3.2.** a) An ODML template structure for the information retrieval domain. b) A small organizational instance produced from that template.

bounded. There may be quality or response time constraints imposed at a high level by the designer. Queries may arrive at regular rate, or at least be probabilistically predictable. Individual databases will vary in size, scope and content. Each of these aspects may affect performance in a non-trivial way.

The problem then, is to determine the most appropriate organization of agents and databases, given the desired characteristics of the system, the provided characteristics of the environment and the tradeoffs we have presented here. For example, how tall should the aggregation hierarchies be? How many nodes should be searched to answer a query? How many mediators should be created? How should these various roles be mapped to actual agents? In the remainder of this section, I show how these questions can be answered by embedding the relationships described above in an ODML model. That model is used in Section 4.1 as the basis for a search process in the space of possible organizational designs.

The inclusion of this domain serves two purposes. First, it demonstrates that ODML can be used to model a different domain than the distributed sensor network, which lends credibility to the argument that ODML can be used to capture a range of different multi-agent system designs. Second, it provides a rich space of organizational alternatives, because one must decide how tall aggregator hierarchies should be, how many mediators to use, and where databases are integrated, among other things. In comparison, the distributed sensor network organization has relatively few organizational variants, and does not pose suitably challenging space of possibilities. Because of this, the information retrieval domain will be used to ground much of the structure analysis and search strategy research presented in this chapter.

## 3.2 IR Simulation

A simulation environment has been created of the information retrieval domain to permit the empirical evaluation of different organizational strategies. The testbed was created using the Farm [85, 79], a distributed simulation framework designed to capture the real-time performance of a multi-agent system.

A Farm simulation consists of a number of specialized plugins connected via a central core, each of which is responsible for a particular aspect of the simulation. The information retrieval domain makes use of several generic plugins for time management, visualization and logging. Three domain-specific plugins were also created, as described below.

**IR Driver** The driver component is responsible for creating the scenario that is to be tested. It takes in an ODML instance (described in detail in Section 3.3, which it uses to first determine the assignment of roles to agents. Certain parameters are also used from the template to determine, for example, the rate that queries should be injected into the system and the base rate at which individual entities should operate. These values are inserted into a global state repository for other plugins to use.

**IR Analysis** This plugin is responsible for aggregating, processing and storing statistical data. It keeps track of the average response recall and response time, the number of total and extant queries, and the average amount of utility that is obtained.

**IR MetAgent** This plugin is responsible for managing the agents that are created in the scenario. Using the configuration data written earlier by the driver, the metagent begins by spawning an appropriate number of agents. Using that same data, each agent is also told what roles it should take on, what parameters it should use for operation, and the relationships it should have with other entities.

It is within the metagent that the individual domain agents exist. Each agent is initially an empty shell, containing only rudimentary message and control routing functions. The actual domain functionalities are implemented as separate role objects (e.g., MediatorRole, DatabaseRole, etc.), where a single agent may take on multiple such roles. While the scenario is running, the agent routes messages to their appropriate role destination, and provides execution time to each role in a round-robin fashion. The roles act in accordance with the behaviors outlined in the previous section, and will be covered in more detail in Section 3.3.

Part of the intent behind creating this environment is to correctly simulate the amount of time that will pass while a particular query is being answered. In particular, the amount of time required for communication and local processing should be captured, as well as the deleterious effects that congestion and latency have on those values. The Farm framework is ideal for this purpose, as agents are provided a specific quanta of CPU time in which to run and are blocked at all other times. This will allow some of the positive and negative effects of different organizations to be directly observable in the response times exhibited by the system. Other characteristics, such as response recall and utility, can be captured through appropriate instrumentation.

### 3.3 Representing IR Characteristics

I will proceed with a description of the characteristics captured by this model. Like the working system it represents, there are many facets to the model. Although each can be modeled as a particular, distinct characteristic of the system, they may interact through coexistence in nodes' fields. The tensions that arise in the resulting object embody the tradeoffs and decisions that must be made when designing the organization.

The interesting aspects of the model are discussed in detail below, concentrating on those characteristics that are significantly different than were seen in the DSN model. This includes how roles are represented, how the collection information signatures are generated, a description of the query and response propagation model, a detailed model of the system's response time, how constraints are used, and how all these features are combined into a utility value. The complete ODML specification for this model can be found in Appendix D.

Together, this and the the DSN model described in Section 2.3 represent the most complete ODML models produced to date. This model also demonstrates a greater level of mathematical sophistication than was present in the DSN model, and can be seen as a successful integration of modeling techniques from related domains (notably probability theory and queuing theory) into an ODML model.

#### 3.3.1 Roles

The main portion of the organization is divided into *mediator*, *aggregator* and *database* nodes. As with the DSN model, these nodes do not represent particular agents by themselves. Instead, each represents a role that may exist in the organization, that is assigned a particular agent through a has-a relationship. Separating these two concepts allows the creation of more complex organizations, where agents may be assigned multiple roles, possessing the capabilities, constraints and responsibilities of each.

#### 3.3.2 Data Sources and Collection Signatures

To correctly estimate work levels, we must first know the type and quantity of information that a source may provide. Different organizations may be necessary if the available information for that topic is concentrated in one spot, or if it is distributed across many separate sources. In this model, we are concerned with a single class or topic of information. This is modeled by specifying the total amount of information owned by a source, along with the fraction of which that is relevant to the topic. This value is constant for databases. Aggregators and mediators, which have no information of their own, derive these values as the sum their respective sources' total and topical information. Ultimately, this is used to calculate the *actual\_response\_size* of the mediator, the total amount of relevant information that will be searched while processing a query.

This raw information is only half the picture, however, as the number of queries that a mediator receives is not dependent on the actual amount of data it manages, but on the data that others *perceive* it to manage. Recall that query processing has two phases, a search phase identifying appropriate mediators, and a query processing phase where a subset of those mediators is selected. The selection process is based on the mediator's collection signature, which is generated from the information in its hierarchy. Ideally, this would be a perfect, correct synopsis. In practice, the signature's accuracy may be skewed by the technique used to generate it, or because of abstraction inherent in the aggregation process. This factor is taken into account in the simple recursive calculation of *perceived\_response\_size* for mediators (*m.prs*), aggregators (*a.prs*) and databases (*d.prs*).

$$m.prs = \sum_{s \in m.sources} s.prs * aggregation\_factor \quad (3.1)$$

$$a.prs = \sum_{s \in a.sources} s.prs * aggregation\_factor \quad (3.2)$$

$$d.prs = response\_size \quad (3.3)$$

The *aggregation\_factor* ( $\leq 1$ ) is used to model the loss of information that occurs at each aggregation point as the contents of subordinates are approximated by a single aggregate signature.

### 3.3.3 Probabilistic Search and Query

Ultimately, the query load incurred by a mediator, and by relation any sources beneath it, will be dependent on the number of queries that mediator is asked to service. This value depends on a number of factors, including the mediator's perceived value, the average number of queries arriving in the system, the number and value of competing mediators, and how many mediators are used to answer the query. To estimate this, we must first determine the relative rank ordering  $M_r$  of the mediator in question  $M$ , and the number of mediators  $R_r$  that share that ranking.

$$M_r = 1 + \left( \sum_{k \in \mathcal{O}.mediators} 0^{\max(M.prs - k.prs, 0)} - 0^{\text{abs}(M.prs - k.prs)} \right) \quad (3.4)$$

$$R_r = \sum_{k \in \mathcal{O}.mediators} 0^{\text{abs}(M_r - k_r)} \quad (3.5)$$

Where *prs* is the *perceived\_response\_size* of the respective mediator. The summation term will equate to 1 when the competing size is higher, and 0 when lower. Thus, the highest ranked mediator will be 1, followed by 2, and so forth. Mediators with the same value will have the same ranking. Using this information, it is possible to compute the probability  $P(M|Q)$  that mediator  $M$  will be selected to service query  $Q$ .

$$\begin{aligned}
P(M|Q) = \frac{s}{|M|} \frac{1}{\binom{|M|-1}{s-1}} & \left( \sum_{i=0}^{q-1} \sum_{j=0}^{\min(s, R_r)-1} \binom{|M| - M_r - R_r + 1}{s - i - j - 1} \right) \times \\
& \binom{M_r - 1}{i} \binom{R_r - 1}{j} \min \left( 1, \frac{q - i}{j + 1} \right)
\end{aligned} \tag{3.6}$$

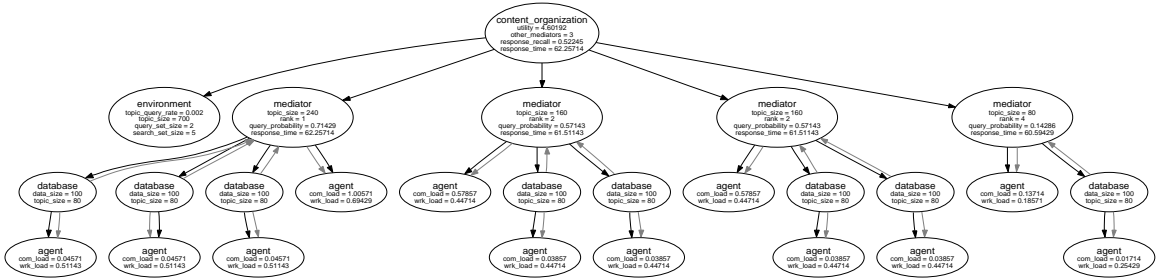
Where  $|M|$  is the total number of mediators,  $s$  is the number of mediators that will be searched and compared, and  $q$  is the number of mediators that will be given the query. Equation (3.6) models the search process and subsequent mediator selection that will take place when a query is received by the system. In this particular domain, some subset of the available mediators will be searched, and ranked based on their collection signatures. Using these ranks, a subset of those searched will actually be selected to service the query. This is a common strategy employed by agent systems, so it is worth discussing the equation in greater detail. The accompanying sidebar provides further insight into the origin of Equation 3.6.

First, assume that all mediators may be initially searched with equal probability, and that selection from a set of equally-ranked mediators is done uniformly. The probability that mediator  $M$  is searched, which depends on the total number searched and the total number of mediators, is simply  $\frac{s}{|M|}$ . Given that  $M$  will be searched, the nested summations count the total number of sets of remaining mediators that both could be searched and would result in  $M$  receiving the query. A ratio of this total to the number of unrestricted mediator combinations that are possible from the search  $\binom{|M|-1}{s-1}$  will provide the final desired probability. The summations work by iterating over the various ways in which the mediator search set might be composed. On each loop, a value is selected for the number  $i$  of higher ranked mediators and  $j$  of equally ranked mediators that will exist in the set, the remainder being made up of lower ranked mediators. Since  $i < q$ , then there will be at least one spot for a mediator ranked  $r$ . There are  $\binom{R_r-1}{j}$  equal valued mediators competing for the available query slots, and the final ratio is calculates the fraction of those that might contain  $M$ . The model in Appendix D uses these equations to determine the final topic query rate for a particular mediator, specifically in the mediator node's *rank*, *rank\_ties*, *query\_probability* and *query\_rate* fields.

An example organization showing the effects of this formulation is shown in Figure 3.3. In this instance, there are four mediators, one with three sources, two with two sources each, and one with a single source. All databases in this model have an equal amount of topic data, so a ranking of  $\{1, 2, 2, 4\}$  can be determined among the mediators respectively, as shown in the model. In addition, there are three other mediators in the organization that contain an insignificant amount of topic data and are not graphically shown. These “other” mediators are significant because they can potentially distract the search process, resulting in a decrease in expected utility. The *environment* node shows that the *search\_set\_size* in this instance is set to 5, indicating that the collection signatures of five other mediators will be searched. The *query\_set\_size*, the number of mediators from the search set that will actually be queried, is set to 3. Therefore, as the number of “other” mediators grows, the

*Sidebar - Abstracting the Selection/Query Problem*

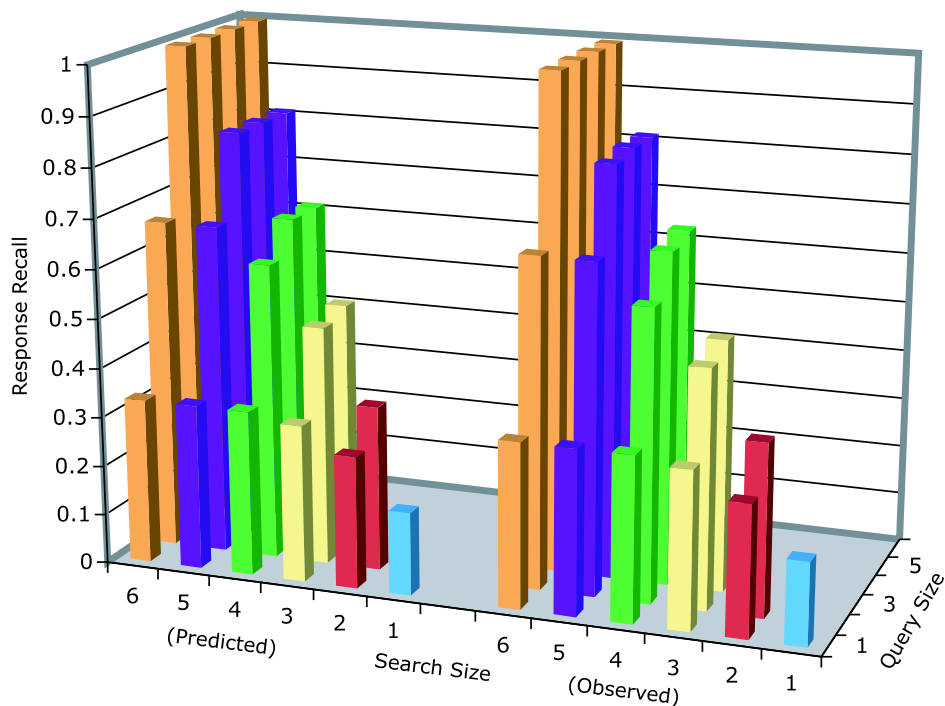
It is illustrative to ignore the domain characteristics of the search and selection process and focus on the underlying counting and probability problem. I will use the familiar ball-and-bag metaphor. You are given a bag containing a set of black, green and red balls of known size, named  $B$ ,  $G$  and  $R$  respectively. One green ball  $g^* \in G$  is distinguished from the others. Select  $s$  balls from the bag to form set  $S$ . Choose  $q$  balls from  $S$  in order of preference red, green, black, i.e., a green ball will only be selected if no reds remain. Call this new set  $Q$ . I wish to find the probability that  $g^* \in Q$ . In this abstraction,  $g^*$  is the mediator in question  $M$ .  $R$  represents those mediators higher ranked than  $M$ ,  $G$  is the mediators of equal rank, and  $B$  those of lower rank.  $S$  is the set of mediators that are searched, and  $Q$  those that ultimately receive the query.



**Figure 3.3.** An information retrieval instance with variously ranked mediators.

chance that one of the relevant mediators will be found and subsequently queried decreases. The value of  $|M|$  in Equation (3.6) is the sum of the relevant mediators and these other mediators. The culmination of these data occurs in the calculation of  $\Pr(M|Q)$ , shown in the *query\_probability* for each mediator. These are used to compute the organization’s *response\_recall*, and ultimately affect the utility of the organizational structure.

To test this formulation, a set of simulation trials were performed, and the observed response recall compared to the predicted value for each scenario. The environment consisted of six mediators and nine databases, and each trial consisted of 100 queries from a simulated user to a random mediator in the organization. The first mediator had four of the databases below it, the second had three and the third had two. The remaining three mediators had no databases, and therefore could not provide value to queries, although their presence made it more difficult to find the actual sources of data because they increased the size of the population to be searched. The *perceived\_response\_size* and *actual\_response\_size* for each mediator was proportional to the number of databases it had access to. In the trials, both the number of mediators that were searched for, *search\_set\_size*, and the number of mediators that were queried, *query\_set\_size* ranged from 1 to 6, producing 36 possible experiments. In practice, only 21 of these were valid, because *query\_set\_size* must be greater than or equal to *search\_set\_size*. A graph comparing the values predicted by the ODML model and the empirical results are shown in Figure 3.4. As expected, when the search size



**Figure 3.4.** A comparison of the predicted and empirical response recall values across a range of search and query size parameters.

is small, the recall suffers, because it is less likely a good information source will be found. The *query\_set\_size* has a similar but lesser effect. For clarity, the relative error between the predicted and observed values are given in Figure 3.1. This shows that the predictions were quite accurate in most cases, with a maximum relative error of 5.9% in one case and an average of 0.9% error over all cases.

The relationships described here are a good example of how changes to the organization can indirectly affect the characteristics of many, potentially distant parts of the structure. In this case, the perceived, relative quality of a mediator, which is based on the sources under its control, affects the ranking of all other mediators in the organization. These rankings affects query load, which affects the load imposed on the agents, which can affect both the constraints on those agents and the response time of a mediator’s hierarchy as a whole. Thus it is possible for a single source added to some segment of the organization to dramatically affect nodes with which it does not obviously interact. These effects can be subtle yet important, motivating the need for a model such as ODMML capable of representing them. It is also shown in Section 4.2.1 how this type of indirect interrelationship can make it particularly difficult to determine either the validity of an organizational instance prior to its complete construction. I will return to this problem in Chapter 4, which discusses how the organizational design problem can be framed as a search for the most appropriate valid instance.



**Table 3.1.** The relative error (i.e.,  $(observed - predicted)/observed$ ) between the predicted and empirical response recall values from Figure 3.4.

		Query Size					
		1	2	3	4	5	6
Search Size	1	-0.059					
	2	-0.011	0.013				
	3	0.008	-0.024	-0.003			
	4	-0.005	0.005	0.005	-0.002		
	5	-0.004	-0.006	-0.011	0.005	-0.012	
	6	-0.004	-0.003	-0.002	-0.001	-0.003	-0.003

### 3.3.4 Query Response Time

The response time of the information retrieval hierarchy is the amount of time that elapses between a user query and the system response. This particular characteristic is clearly important from an evaluation standpoint, as it captures an easily observable phenomena that is important to the end user. Like the probabilistic query model, the response time is intimately tied to the structure and characteristics of the organization.

The response time characteristic is also more complicated than those discussed so far, because it cannot be accurately captured with simple average-case statistics. The solution to this problem is fairly complicated, drawing upon techniques from probability and queuing theory. I will begin by describing why simpler techniques are insufficient and progress with the actual solution in stages, gradually incorporating new elements into the solution as deficiencies are recognized.

To better understand the various components that affect the response time, it is helpful to follow the lifecycle of a particular query. Consider the simple organization shown in Figure 3.2b. A query will first arrive at the mediator, who will begin by searching for an appropriate set of mediators to answer the query. After eliciting a response from some subset of candidate mediators, the original mediator will select from among them the set that looks most promising, and provide them with the query. In this example, there is only one mediator, and it will progress by sending copies of the query to all the information sources below it. The two databases below the mediator will immediately begin processing the query, while the aggregator will pass the query down to its two information sources. At this point both the mediator and aggregator must wait for their respective information sources to formulate replies before progressing. When the aggregator's databases have completed, the aggregator will combine their responses and send a reply back to the mediator, while the remaining two databases also report directly to the mediator. Finally the mediator will combine all the results it has received and report back to the user.

There are several aspects of this flow that directly affect the response time. Each communication event incurs some latency because of the message transit time. The

query processing by the databases, and the aggregation performed by both the aggregator and mediator will take some amount of time. The latter two entities must also wait until the slowest of their information sources to respond before they can themselves respond. Finally, because multiple queries can exist simultaneously in the network, additional delays at individual nodes can be incurred when a query must wait for existing processing to complete.

I will begin with the *query\_rate* characteristic of the mediator, which predicts the arrival rate of queries to that mediator based on the probabilistic model given above. One can infer that responses will, on average, be returned back to the user at this same rate. The response rate cannot be faster, because the system would eventually run out of queries to process. If the response rate were slower, the number of queries in the system (along with the expected response time) would tend to infinity as new queries encounter an ever lengthening queue of existing queries upon arrival. Of course, this is not an impossible situation, just undesirable, so we must include constraints in the model that specify that the possible rate at which tasks are serviced must be greater than or equal to the rate at which they arrive. Given these constraints, we can assume that the query rate will equal the response rate. We can furthermore assume this is the case for all nodes in the hierarchy by analogous reasoning, after observing that the arrival and response rates of one node dictate the complementary rates of the other nodes they are attached to.

More concretely, the existing model assumes that queries arrive to the mediator with a Poisson arrival distribution and mean rate *query\_rate*. The model is further abstracted by saying that tasks arrive with rate *arrival\_rate*, where each task is a query and *arrival\_rate* = *query\_rate* in this instance. This means that the amount of time between subsequent tasks will be a random value sampled from an exponential distribution with parameter *arrival\_rate* — on average, a new query will arrive at the mediator every *arrival\_rate*<sup>-1</sup> milliseconds.

After some amount of time elapses, during which the query makes its way down through any aggregators, the databases themselves will receive the query. By the logic given above, we can assume that they arrive at rate *arrival\_rate*. Each database is also associated with a *service\_rate*, which reflects its ability to complete queries given to it. When this new query arrives there may be previously received queries currently being processed or waiting to be processed by the database. Because we assume first-in, first-out processing, the amount of time the new query must wait before being addressed will depend in part on these existing queries.

We can exploit existing techniques from the field of queuing theory [101, 151] to help analyze how long the wait will be, an approach that has recently proved successful in other MAS models [68, 176]. For example, the database node as it has been defined can be modeled with a *M/M/1* queue. This model assumes a Poisson task arrival rate (i.e., *arrival\_rate*) and service rate (i.e., *service\_rate*), and a single processor (i.e., the agent performing the database role and the resources under its control). From this information, one may immediately determine the expected *service\_time* of the database, using the formula [151]:

$$service\_time = \frac{1}{service\_rate - arrival\_rate}$$

However, this single expected value is not what we actually need to compute at this point in the process, for reasons that will become clear below. Instead, the model uses this same basic information to compute approximations of the probability density function (pdf) and cumulative distribution function (cdf) of the service time characteristic. These functions represent richer forms of the same waiting time knowledge, because they preserve the statistical character of the phenomena, rather than just a sample mean. The pdf  $f_M(x, \lambda)$  and cdf  $F_M(x, \lambda)$  of the  $M/M/1$  queue's waiting time distribution are [47]:

$$\begin{aligned} f_M(x, \lambda) &= \lambda e^{-\lambda x} \\ F_M(x, \lambda) &= 1 - e^{-\lambda x}, \end{aligned}$$

where  $x \geq 0$  and  $\lambda = \text{service\_rate} - \text{arrival\_rate}$ . The model maintains this information as a discrete list of sampled points, which are calculated dynamically from the two underlying functions. In particular, it defines the following two lists:

$$\text{pdf\_list} = [f_M(x_0 \times \text{dist\_step}, \lambda), \dots, f_M(x_n \times \text{dist\_step}, \lambda)] \quad (3.7)$$

$$\text{cdf\_list} = [F_M(x_0 \times \text{dist\_step}, \lambda), \dots, F_M(x_n \times \text{dist\_step}, \lambda)], \quad (3.8)$$

where  $x_n = n$  and  $(0 \leq n < \frac{\text{dist\_range}}{\text{dist\_step}})$ .  $\text{dist\_range}$  represents the upper bound on the sampled points, while  $\text{dist\_step}$  is the stride length between points. Lines 45 and 48 in Figure 3.5 show the corresponding actual ODML definitions, which are initially stored as *local\_pdf\_list* and *local\_cdf\_list*. Sample curves from the data produced by these two fields are shown in Figure 3.6, with a *arrival\_rate* of 0.002 and *service\_rate* of 0.005. Note that, if we wish, we can compute an expected *service\_time* for the database from this data, using the conventional definition of expected value:

$$\text{service\_time} = \sum_{x=1}^{\text{dist\_range}/\text{dist\_step}} (x \times \text{dist\_step})(\text{pdf\_list}[x] \times \text{dist\_step}) \quad (3.9)$$

The first multiplicative term restores  $x$  to the value it would have after the stride (i.e., the appropriate duration), while the second term accounts for the probabilities lost by the discrete nature of the the sample distribution. The fact that a simple product is used to recover those missing points from the distribution implies that the resulting data are only an approximation of the true function. The precision of the approximation increases as  $\text{dist\_step} \rightarrow 1$ , as do the time and space needed for the computation.

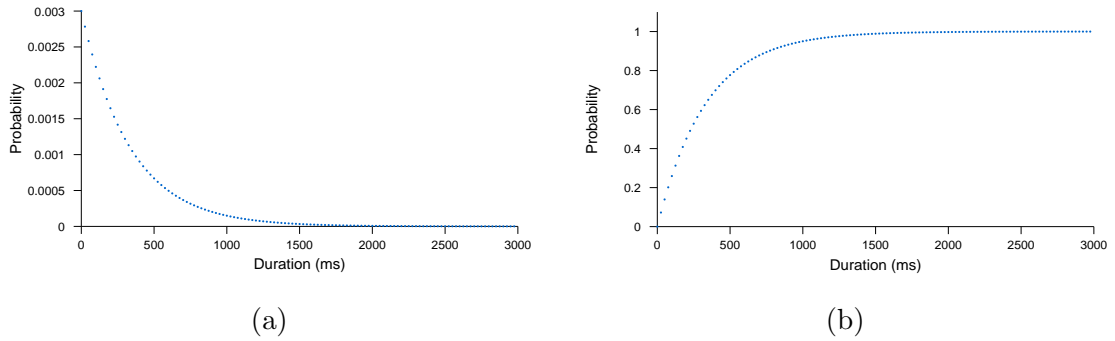
A similar summation is used in the model to calculate the total density captured by the distribution. A constraint over this total ensures that a sufficiently complete view of the behavior is retained, by causing any situation where a significant number of time points have exceeded the distribution's window to be recognized and marked invalid.

```

1 <node type="mediator">
2   <constant name="arrival_rate">query_probability * env.topic_query_rate</
   constant>
3   <constant name="service_rate">env.response_service_rate / num_sources</
   constant>
4   <constant name="poisson_rate">service_rate - arrival_rate</constant>
5
6   <constant name="local_pdf_list">forrange(x, 0, (dist_range / dist_step),
7     poisson_rate * e^(- poisson_rate * x * dist_step) /* Exp pdf f(x) */
8   )</constant>
9   <constant name="local_cdf_list">forrange(x, 0, (dist_range / dist_step),
10    1 - e^(- poisson_rate * x * dist_step) /* Exp cdf F(x) */
11  )</constant>
12
13  <constant name="source_pdf_list">forrange(x, 0, (dist_range / dist_step),
14    forallprod(forall(s, sources, listitem(s.cdf_list, x))) * forallsum(forall(
15      s, sources, listitem(s.pdf_list, x) / listitem(s.cdf_list, x)))
16  )</constant>
17  <constant name="source_cdf_list">forrange(x, 0, (dist_range / dist_step),
18    forallprod(forall(s, sources, listitem(s.cdf_list, x)))
19  )</constant>
20  <constant name="pdf_list">forrange(x, 0, (dist_range / dist_step),
21    forallsum(forrange(i, 0, x, listitem(local_pdf_list, i) * listitem(
22      source_pdf_list, x - i) * dist_step
23    )))</constant>
24  <constant name="cdf_list">forrange(x, 0, (dist_range / dist_step),
25    forallsum(forrange(i, 0, x, listitem(local_pdf_list, i) * listitem(
26      source_cdf_list, x - i) * dist_step
27    )))</constant>
28  <constant name="service_time">forallsum(forrange(x, 1, (dist_range / dist_step
29    ), (x * dist_step) * listitem(pdf_list, x) * dist_step
30  )</constant>
31  <constant name="overhead_time">
32    2 * env.message_latency /* Query to/from user */
33    + 2 * env.message_latency /* Search to/from mediators */
34    + 2 * env.message_latency /* Query to/from mediators */
35    + env.message_latency /* Query down to sources */
36    + max(sources.overhead_time) /* Subordinate overhead */
37  </constant>
38  <constant name="response_time">overhead_time + source_service_time</constant>
39  ...
40 </node>
41
42 <node type="database">
43   <constant name="query_rate">manager.query_rate</constant>
44   <constant name="service_rate">env.process_service_rate</constant>
45   <constant name="poisson_rate">service_rate - query_rate</constant>
46
47   <constant name="local_pdf_list">forrange(x, 0, (dist_range / dist_step),
48     poisson_rate * e^(- poisson_rate * x * dist_step) /* Exp pdf f(x) */
49   )</constant>
50   <constant name="local_cdf_list">forrange(x, 0, (dist_range / dist_step),
51     1 - e^(- poisson_rate * x * dist_step) /* Exp cdf F(x) */
52   )</constant>
53   <constant name="pdf_list">local_pdf_list</constant>
54   <constant name="cdf_list">local_cdf_list</constant>
55
56   <constant name="overhead_time">env.message_latency /* Response to manager */
57   </constant>
58   ...
59 </node>

```

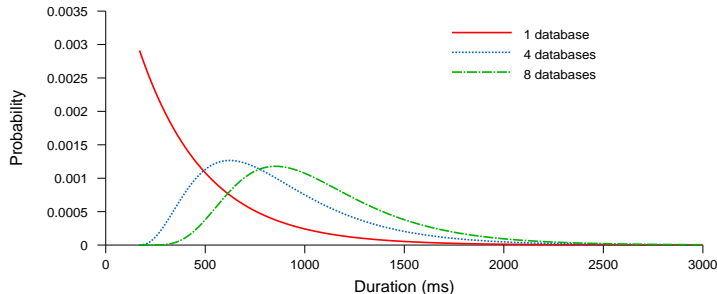
**Figure 3.5.** A portion of the ODMML specification for the *mediator* and *database* nodes detailing the fields used to estimate response time.



**Figure 3.6.** Sample (a) pdf and (b) cdf profiles for the database waiting time distribution.

Let us ignore the aggregator for the moment, and focus our attention on the mediator. Recall that it must wait for responses from all its information sources before progressing, which directly ties its *service\_time* to those of the information sources below it. A simple approach would find the average such *service\_time*, and use that to estimate that component of the mediator’s service time. However, this metric is actually a biased approximation of the service time, for reasons similar to those presented in Jensen and Cohen’s discussion of multiple comparison procedures [95]. Recall that databases are modeled as Poisson processes. Although they do have a mean service time, most of the time a greater or lesser value will be observed in practice. If there is just one database to wait for, the sample mean would be a sufficient and accurate characterization. However, if there are two or more databases to consider, multiple samples will be made, effectively increasing the chance that a greater-than-mean value will be observed. This skews the response time distribution so that the aggregate waiting time is no longer a simple Poisson. This can be seen in Figure 3.7, which shows how the waiting time distribution of a mediator changes with the number of databases below it. Notice that as the number of databases increases the distribution shifts to the right, which is consistent with our intuition.

The underlying issue is that the mediator must wait for the *slowest* responder, which means that its service time will be dependent on the *maximum* service time of those below it, not simply the average. This is the motivation for explicitly representing the pdf and cdf of the database above. Given this more detailed information, it is possible to determine what the actual distribution is of the time the mediator must wait for its responders. A branch of probability theory known as *order statistics* is useful here. Assume that we have  $n$  samples from some distribution  $X$ ,  $(X_1, \dots, X_n)$ .  $X_{(k)}$ , the  $k$ th smallest sample from this set, is known as the  $k$ th order statistic [36, 148]. The  $n$ th or *maximum order statistic* of the waiting distribution faced by the mediator is the expected maximum value in the set of samples. This corresponds to the amount of time the mediator is expected to wait before all responses have been received.



**Figure 3.7.** A comparison of the waiting time distributions for differently sized sets of databases.

The example order statistic given above is known as the independent, identically distributed (iid) case, because all  $n$  samples are from the same random variable  $X$ . Unfortunately, this is not the case seen in the information retrieval system. First, we do not assume that all databases necessarily have the same service rate. Second, different aggregation structures with different heights and widths will also produce different waiting time distributions. The model does make a simplifying assumption, however, that the various samples are independent. Together, this is known as the independent, non-identically distributed (inid) case.

After determining the appropriate case, and finding the appropriate inid expectation function, we could progress straight to the goal by finding the expected *service\_time* mean for the mediator. Instead, as before, the model generates the more informative pdf  $f_{(n)}$  and cdf  $F_{(n)}$  sample distributions of the  $n$ th order statistic for the source service time, using the following functions from [36] and [148]:

$$f_{(n)}(x) = \left[ \prod_{i=1}^n F_i(x) \right] \sum_{i=1}^n \left( \frac{f_i(x)}{F_i(x)} \right) \quad (3.10)$$

$$F_{(n)}(x) = \prod_{i=1}^n F_i(x), \quad (3.11)$$

where  $f_i$  and  $F_i$  represent the pdf and cdf of the  $i$ th sample, respectively (i.e., the service time distribution of the  $i$ th information source). Sample lists are generated for these two distributions in the same manner shown in equations 3.7 and 3.8, which are reflected in lines 13 and 16 of Figure 3.5.

What has not been mentioned so far is that the mediator itself is not simply a pass-through, but must process and aggregate the resulting data as well. Just as with the processing of queries by the database, the processing of the results by the mediator also takes time, potentially causing newly arrived results to wait until the mediator can devote attention to them. Thus, the mediator can also be viewed and modeled as a queue. In this case I will assume it is also a  $M/M/1$  queue, with an arrival rate consistent with the argument presented earlier. The service rate exhibited

by the mediator depends on the number of responses it receives, which is determined by the number of information sources below it. The local pdf and cdf for the mediator can therefore be produced using the structure defined in Equations 3.7 and 3.8, with  $arrival\_rate = query\_rate$ ,  $service\_rate = response\_service\_rate/num\_sources$ , and Poisson rate  $\lambda = arrival\_rate - service\_rate$ .

At this point we have identified and described distributions for both the mediator itself and the information sources below it. To complete the flow propagation we must determine the behavior of the total service time that combines these two activities. This can be accomplished by recognizing that this service time will be the sum of the times exhibited by these two random variables, since the local processing phase takes place after all results have been received. The total service time pdf  $f_C$  and cdf  $F_C$  can then be determined by finding the convolution of the corresponding distribution functions, which has the general form:

$$f_C(x) = \sum_{i=0}^{dist\_range/dist\_step} f_s(i)f_l(x-i)dist\_step \quad (3.12)$$

$$F_C(x) = \sum_{i=0}^{dist\_range/dist\_step} f_s(i)F_l(x-i)dist\_step \quad (3.13)$$

For the mediator,  $f_s$  would be the aggregate information source pdf given in Equation 3.10, while  $f_l$  and  $F_l$  would be the pdf and cdf of the waiting time for the local  $M/M/1$  queuing process.

Both these convolution equations and those used earlier to compute the maximum order statistics make no assumptions about the underlying distributions they reference. Because of this, any other queuing model can be substituted for the  $M/M/1$  queues used in these roles, so long as it can be characterized or approximated through a closed form formula using the mathematical primitives supported by ODML. One could also directly provide a complete discrete distribution in its place.

Armed with this new information, the model can now compute the expected *service\_time* of the mediator, by using the same expression previously shown in Equation 3.9 coupled with the cumulative overhead incurred during the query and result propagation process, as shown in Lines 27-36 in Figure 3.5. This overhead, which is computed incrementally up the hierarchy, models the latency time of message transit.

Note that Equations 3.10-3.13 are recursive, in that they rely upon both the pdf and cdf distributions of the information sources below the mediator. The equations make no assumptions about the form of those distributions, which permits them to be used both when the information source is a single database or if the information source is an arbitrarily complex aggregator hierarchy — provided that root aggregator is able to correctly express its pdf and cdf. Fortunately, this same assumption also allows Equations 3.10 and 3.11 to be used to compute the pdf and cdf distributions for the aggregator itself. The recursive definition will eventually terminate in the exponential distribution exhibited by at the leaf nodes by the databases. As its name implies, the aggregator also performs a response aggregation, which can be approximated with a

suitable queuing model. The expressions given in Equations 3.12 and 3.13 are again used to combine these two characteristics to determine the aggregator’s total service time pdf and cdf distributions.

It is also worth mentioning that it is the response time distribution lists previously computed by a node’s subordinates, and not the functions they approximate, that are used to determine the response time of the node itself. So, although the pdf and cdf definitions are recursive, they only need to be computed once for each node in a bottom-up fashion if a suitable value caching system is in place. This technique is similar to those used in dynamic programming to avoid redundant computation.

The final aspect that must be taken into consideration is the effect that multiple roles have on performance. This will be approximated by weighting the *service\_rate* for each role based on the proportion of local processing time it is expected to receive. In particular, if  $\lambda_i$  and  $\mu_i$  are the original *arrival\_rate* and *service\_rate* for an agent’s *i*th role, let the *effective\_service\_rate* ( $\mu_{i_E}$ ) of that role be:

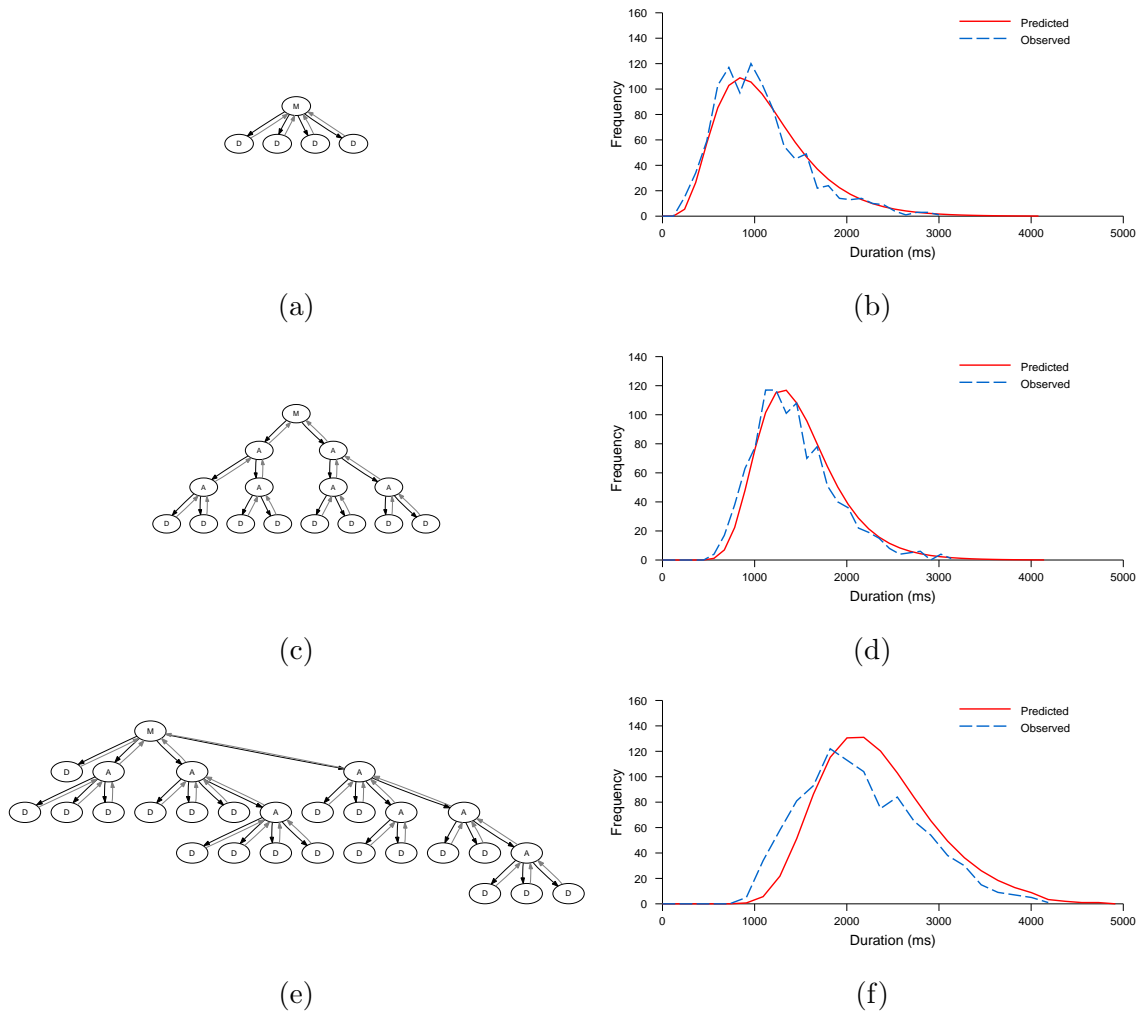
$$\mu_{i_E} = \mu_i \frac{\lambda_i / \mu_i}{\sum_{r=0}^{\#roles} \lambda_r / \mu_r} = \frac{\lambda_i}{\sum_{r=0}^{\#roles} \lambda_r / \mu_r} = \frac{\lambda_i}{agent.work\_load} \quad (3.14)$$

where *agent.work\_load* is the location in the model where the aggregate demand is stored. This expression uses each role’s arrival and service rates to first determine the expected proportion of time the agent will be busy, and then use that value to scale the individual service rates accordingly. This aspect has been omitted from Figure 3.5 for clarity, but can be found in the complete model provided in Appendix D. The role’s arrival rate remains unchanged.

The results of a several sample runs in the simulation environment described in Section 3.2 are shown in Figure 3.8. Each scenario measures the response time performance of a different IR organizational design, by submitting 1000 queries to it in a Poisson fashion as described previously. The organizational design of each scenario is depicted on the left, along with the predicted and empirical response time data on the right. The solid line represents the distribution of response times predicted by the ODML model, while the dashed line indicates the observed frequency of individual response times in the simulation. A bin width of  $W = 2(IQR)N^{-1/3}$  was used to group the empirical response times, where  $N$  is the number of trials and IQR is the interquartile range of the data (the difference between the 75th percentile and 25th percentile of the data). This is the Freeman-Diaconis rule, as discussed in Izenman’s bin width strategy analysis [92].

As can be seen in the performance graphs, the ODML model does a good job of predicting the response time distribution of the different organizational designs. Additional response time trials were performed for organizations with three agents [1M,2D], 10 agents [1M,2A,7D], and 14 agents [1M,3A,10D], with similar results. The coefficient of determination  $R^2 (= 1 - \frac{(y-\hat{y})^2}{(y-\bar{y})^2})$  was calculated for each scenario, which estimates how much of the observed behavior can be explained by the model [47].  $R^2$  was greater than 0.8 for all tested scenarios (0.96, 0.95, 0.83, 0.95, 0.89, 0.81, in order of increasing organization size), where a value of 0.7 or above is considered good for this statistic.





**Figure 3.8.** A comparison of the response time distributions predicted by the ODML model and observed in organizations with (a,b) five [1M,4D], (c,d) fifteen [1M,6A,8D], and (e,f) twenty-eight [1M,7A,20D] agents. In the designs shown in (a,c,e), node M is the mediator, A are aggregators, and D are databases.

The effort taken to preserve the underlying probability distributions in this aspect of the model has other benefits, besides being necessary to accurately model the response time behavior. This same information can be used to support high-level behavioral constraints. For example, a constraint can be defined using the mediator’s cdf that places an upper bound on the the probability that a particular response time is exceeded. The pdf can also be used to determine the average response time as shown above, which will be used to define the organizational utility in the following section. By explicitly capturing the “fuzzy” nature of the running system’s performance, these richer statistics allow the designer greater control over the evaluation and output of an organizational design process.

### 3.3.5 Constraints

The notion of bounded rationality manifests itself in this domain within the agents. Specifically, each agent has a finite amount of processor cycles and bandwidth at its disposal. There are both soft effects caused by increased load, and hard load constraints that may not be violated. An example of the former is the increased time needed to finish any individual task as the local processing load increases. The latter occurs when the agent can no longer keep up with the requests it receives. In this case, the local work queue will grow without bound, causing an untenable situation.

Both these effects are present in the model. The relevant high level metric is the mediator’s *response\_time*, which represents the average length of time from query to response as mentioned above. Each role has a set of responsibilities that affect the agent it is assigned to. The model specifies the *arrival\_rate* and *service\_time* of the role, which are propagated with a modifier to its agent to compute the agent’s aggregate *work\_load*. During instantiation, this *work\_load* will then reflect the cumulative effects of each of its roles. Equation 3.14 shows how this value is used to compute the *effective\_service\_rate* for each role. The previous section outlined the soft constraining effects of this interaction in detail. A hard constraint is also specified in each role that ensures that  $arrival\_rate \leq effective\_service\_rate$ , which prevents the local queue length from growing indefinitely.

A separate hard constraint exists at the organizational level that places a lower bound on the average response recall exhibited by the system.

### 3.3.6 Organizational Utility

A key evaluation criteria used by [215] is information recall, called *response\_recall* in the ODML model. This metric, defined as the ratio of relevant documents retrieved to the total number of relevant documents available, objectively quantifies the quality of the query response. The mediators’ *query\_probability* and *actual\_response\_size*, along with the total amount of relevant information in the environment, can be used to determine the average information recall for the organization. A secondary metric, the *response\_time*, gives the expected average amount of time the system requires to answer a query as explained above.

These two metrics are combined by the *organization* node in its *utility* field, which is typically used to compare and rank candidate instances. In this case, recall is more important than response time, so a multiplicative factor is applied to the recall value, after which the response time is subtracted out:

$$utility = response\_recall * 1000 - response\_time/10$$

Recall is the proportion of the possible information that was reported ( $[0 \dots 1]$ ), while time is measured in milliseconds ( $[0 \dots \infty)$ , generally in the thousands). For example, a system with *response\_recall* of 0.8 and *response\_time* of 1050 ms will have a utility of 695. The normalization terms cause this formulation to generally favor quality over speed, and instances with equal recall will be differentiated by their response time. An arbitrary utility function could be substituted here as needs dictate.










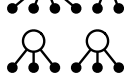

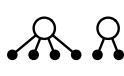





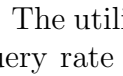
Figure 3.9 shows how utility is affected by the expected user query rate. Optimal utilities for each rate are shown in bold. This figure shows the full range of possible organizations that are possible in a six database environment with a maximum height of three and a minimum of two subordinates per node (there is no need for an aggregator with only one subordinate). The *search\_set\_size* and *query\_set\_size* were set to be the total number of mediators for each organization. The response recall is therefore identical in all cases, and the utility is determined by the response time. Organizations with zero utility were not valid at the given query rate because the arrival rate exceeds the service rate at one or more points, resulting in an infinite queue.

The single-level, single-mediator organization number 1 is predicted to be optimal when the query rate is 0.5 or less (i.e., less than one query every other second). This is intuitive, because the slow query rate avoids queuing delays, causing the response time to be dominated by the height of the organization. Note that other short hierarchies such as numbers 11 and 13 also have high utility.

As the query rate increases, first organization 11, then number 8 and finally number 9 become optimal, as the highly-connected mediator in organization 1 becomes a bottleneck. In contrast to most of the competing designs, organizations 11, 8 and 9 are all balanced (as is number 1). In an unbalanced organization, the segment with greater load tends to dominate the response time because the final result must wait for the slowest responder. These three designs avoid this by evenly spreading the load among participants.

The differences between the three revolve around where the final aggregation takes place, and how distributed the database search is. For example, 8 and 9 employ an additional mediator to aggregate the final results, while 11 does not. In 11, one of the two mediators serves this role, which will slightly unbalance the load in the organization for any given query (i.e., one of the two mediators forms the final result, while the other remains idle). Therefore, although the lack of another aggregating level saves on communication, the transient load imbalance ultimately leads to an increased chance of queuing delays. Organization 18 suffers from this same problem, although more so because one mediator must aggregate for two others.

Organization 9 outperforms 8 under the highest loads because the database aggregation is better distributed, which reduces the chance that any one node will observe

Organization	Query Rate									
	0.1	0.25	0.5	1	2	3	4	5	6	7
1. 	<b>725</b>	<b>723</b>	<b>719</b>	710	672	544	0	0	0	0
2. 	715	714	711	704	681	621	0	0	0	0
3. 	716	715	712	707	691	664	598	0	0	0
4. 	715	714	712	706	691	664	598	0	0	0
5. 	713	712	709	702	679	621	0	0	0	0
6. 	710	709	706	701	687	662	599	0	0	0
7. 	711	710	708	704	693	677	651	597	465	0
8. 	712	711	709	705	<b>695</b>	<b>680</b>	<b>657</b>	612	482	0
9. 	705	704	702	699	690	677	<b>657</b>	<b>619</b>	<b>517</b>	0
10. 	710	709	707	702	689	665	604	0	0	0
11. 	721	719	717	<b>711</b>	<b>695</b>	665	581	0	0	0
12. 	711	710	707	703	690	666	607	0	0	0
13. 	720	718	715	709	687	629	0	0	0	0
14. 	710	708	706	701	687	662	600	0	0	0
15. 	710	709	707	703	691	674	646	587	439	0
16. 	703	702	701	697	687	673	650	605	471	0
17. 	705	704	702	698	688	674	651	605	472	0
18. 	709	707	704	698	678	641	536	0	0	0

**Figure 3.9.** The utility predicted for the range of possible six-database organizations when the query rate (queries per second) is varied. Mediators and aggregators are shown as hollow circles, while the solid databases form the leaves. Higher is better, optimal values for each rate are shown in bold.

high queuing delays. By having three aggregation points of size two rather than two of size three, the range of likely durations is reduced (see Figure 3.7). The tradeoff is that the mediator has a greater load, because it now has three subordinates. Under most loads (5 queries per second or less) these two strategies perform similarly, with 8 being slightly higher because it has fewer nodes to interact with. However, under the highest viable load of 6, organization 9 outperforms 8 because only the mediator has an in-degree of three. In organization 8, there are two such nodes, which means there are two chances for a highly-loaded aggregator to slow the overall response time. Since organization 9 has only one such node, the chances of this occurring are lessened.



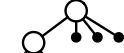
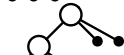
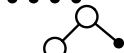









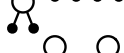



A similar set of results is shown in Figure 3.10. Unlike those in Figure 3.9, the query and search sizes are both fixed at one in these experiments. This has the effect of making response quality a differentiating factor. As shown in Section 3.3.3, the *response\_recall* of a single-mediator system will be greater than that of a multiple-mediator system when only a strict subset of available mediators are searched.

There are two characteristics that provide an interesting contrast with the previous results. The most obvious is the dramatically reduced utility in organizations 11-18 that have more than one mediator. Because at most one mediator is searched, at most one will contribute the information under its control to answer the user's query. The recall of organizations with two mediators is roughly half that of those with one, while organization 18 with three mediators has but a third of that recall. This correspondingly degrades the organization's utility.

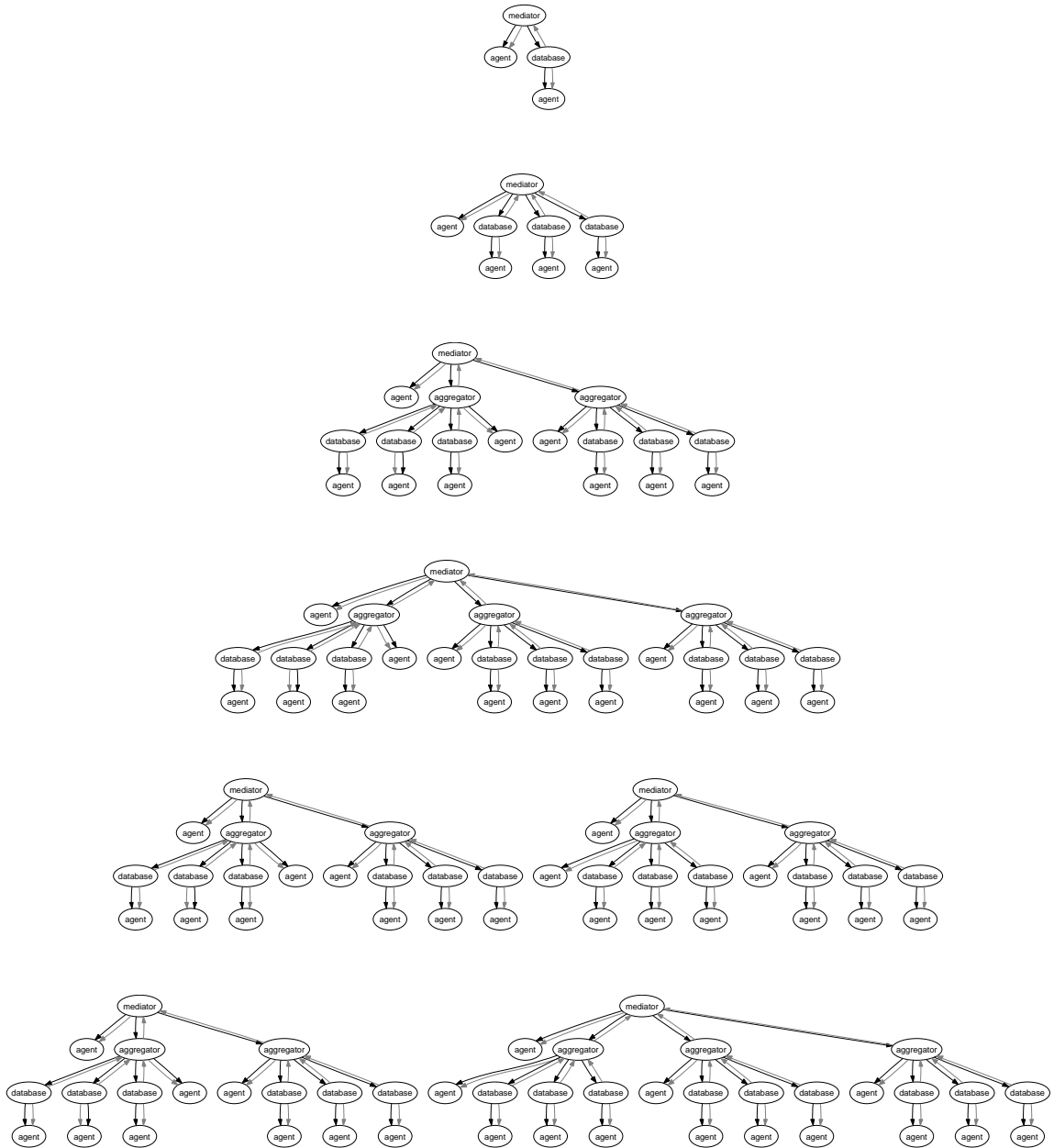
The benefit this change has is increased robustness to high work loads. Where no organization could handle more than six queries per second in the original scenario, all eight multi-mediator designs can now obtain utility with at least seven queries per second. This is because the smaller search size prevents individual queries from being handled by the entire agent population, which frees up computing resources that can then be used on subsequent queries. The aggregate demand on the system is lower, which reduces the growth rate of individual agents' queues, which allows the system as a whole to tolerate higher query rates. The remaining pressures on these more fragmented systems are the same — note the parallels between organizations 11 and 1, which are optimal at opposite ends of the spectrum.

The pressure applied to the information retrieval organization as the query rate increases is in some sense a top-down stress. An analogous bottom-up stress occurs when more databases exist in the organization. Instead of more queries existing in the system, there are more responses, but both can be addressed with similar organizational techniques.

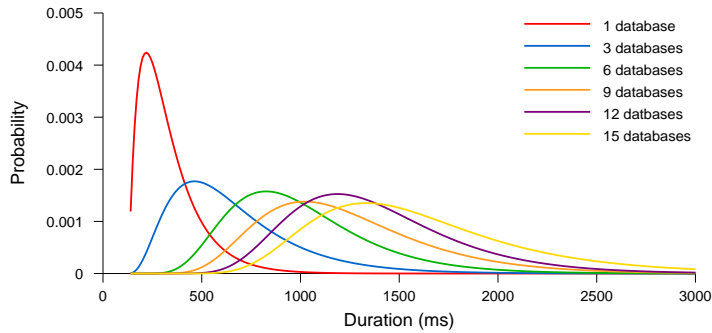
Figure 3.11 shows the optimal organizations when between 1 and 15 databases are incorporated into the system, using a constant query rate of 3 per second. Similar to the previous experiments, the search and query set sizes were set to the number of mediators in the environment. Unlike the previous experiments, different numbers of databases are available, which means the utility is based on both the organization's response recall and time. Because recall is weighted greater than time, the optimal design will integrate all available databases and organize them to minimize the expected response duration.

Organization	Query Rate									
	0.1	0.25	0.5	1	2	3	4	5	6	7
1. 	<b>725</b>	<b>723</b>	<b>719</b>	<b>710</b>	672	544	0	0	0	0
2. 	715	714	711	704	681	621	0	0	0	0
3. 	716	715	712	707	691	664	598	0	0	0
4. 	715	714	712	706	691	664	598	0	0	0
5. 	713	712	709	702	679	621	0	0	0	0
6. 	710	709	706	701	687	662	599	0	0	0
7. 	711	710	708	704	693	677	651	597	465	0
8. 	712	711	709	705	<b>695</b>	<b>680</b>	<b>657</b>	612	482	0
9. 	705	704	702	699	690	677	<b>657</b>	<b>619</b>	<b>517</b>	0
10. 	710	709	707	702	689	665	604	0	0	0
11. 	379	379	379	378	378	378	377	376	375	<b>374</b>
12. 	374	374	374	374	373	373	372	372	371	370
13. 	379	379	378	378	378	377	376	375	373	371
14. 	374	374	374	374	373	373	372	371	371	370
15. 	374	374	374	373	373	373	372	371	371	370
16. 	373	373	373	373	373	372	372	371	371	370
17. 	373	373	373	373	373	372	372	371	371	370
18. 	252	252	252	252	251	251	251	251	251	251

**Figure 3.10.** Similar to Figure 3.9, the utility predicted for the range of possible six-database organizations when the query rate (queries per second) is varied, but with the query and search sizes set to one. Note differences in organizations 11-18.



**Figure 3.11.** The range of optimal organization instances derived from the IR template, when 1,3,6,9,12 or 15 databases are available. The *environment* and *organization* nodes have been omitted for clarity.



**Figure 3.12.** A comparison of the waiting time distributions for the different optimal organizations from Figure 3.11. The distributions widen and shift right as the number of databases increase.

The shapes of these organizations change in a manner similar to the progression of optimal organizations seen in Figure 3.9. When there are few databases (e.g., 1 or 3), and correspondingly less work to be performed by the system, the optimal organization remains flat to avoid communication delays. As the number of databases increases, the load becomes more distributed. This happens first by introducing more aggregators (e.g., the 6 database case), and later by adding more mediators to break the hierarchy apart. As Figure 3.12 shows, the response time inevitably grows as more data sources are added, but this same change brings with it the desired increase in recall. These changes make tradeoffs very similar to those discussed above, by minimizing the response overhead while avoiding potential queue-related bottlenecks.

### 3.4 Conclusions

By introducing a second realistic domain, this chapter has shown that ODML is not limited to the types of organizations or problems addressed by the distributed sensor network application from Section 2.2. The information retrieval model captures several interesting and complex details, including a utility-driven search process and the consequences of queuing on response time in a distributed work flow. This demonstrates how existing modeling techniques from other disciplines can be successfully incorporated, and how the choices made during modeling may be empirically validated through simulation. Additional examples of this are shown in Chapter 5.

This chapter also briefly mentioned the potentially large space of design alternatives that can be derived from the model, and how ODML’s ability to predict organizational utility can help evaluate members of that space. The following chapter will show how it is this set that must be explored to solve the organizational design problem, and present a variety of ways this may be accomplished.



## CHAPTER 4

### DESIGNING ORGANIZATIONS

I have thus far demonstrated the measurable effect that organizations have on performance, and it is therefore useful to be able to understand those effects when designing an agent system. Section 2.2 demonstrated ODML's ability to model, predict and evaluate the characteristics of particular organizational designs used in a previously constructed, real-world application. Chapter 3 describes how ODML is used to create a model of a second domain, an information retrieval (IR) system. The range of possible organizations in these two domains along with the detailed predictions made by the ODML model naturally leads one to consider the possibility that an automated process could be used to select an appropriate organizational design. This chapter explores this possibility, by first describing how ODML can represent these alternatives, and then giving insight into how this representation enables the design of organizations.

The necessity for this technology is derived from at least three parts. The first, as shown in Section 2.1, is that the particular organization that is employed can have a significant effect on a range of important runtime characteristics. Chapter 3 showed the potential for complex interactions among these characteristics. This provides a second motivation, as design intuition can fall short when these details become simultaneously critical in importance and difficult to discern. The potential for a large or incomplete space of possible designs is the third motivating factor. For example, the initial by-hand enumeration of possible designs for the relatively simple six-database experiments in Figure 3.9 came up with 16 alternatives. It was only after the more methodical and computational search techniques described in this chapter were applied to the model that the remaining two designs were revealed (numbers 9 and 12), one of which was shown to be optimal under some conditions.

This chapter begins in Section 4.1 with a high-level discussion of the design problem and the process used by ODML to address it. The complexity analysis of the valid design search problem ODML-SAT given in Section 4.1.1 shows it to be NEXP-Complete, providing insight into the difficulty of the problem that ODML is addressing.

The techniques in Sections 4.2 and 4.3 describe suite of approaches to search and modeling designed to cope with the large search space. Section 4.2 describes several algorithmic approaches that shrink the search space, including one that uses value trends and hard constraints to bound the search and another that defines notions of equivalence to avoid redundant search. Section 4.3 takes a different approach by modifying the model itself in an attempt to reduce the search space without affecting the potential utility of the optimal design.

Section 4.4 describes more of how the design process is implemented and works in practice. This includes how candidate instances are evaluated (Section 4.4.1), where the design and construction process can take place (Section 4.4.2), and how a selected design is used in practice (Section 4.4.3).

The chapter concludes in Section 4.5, which describes the organizational adaptation process. This differs from the design process in that it must be an online, reactive or proactive process. An outline of how ODML can be used to address this problem is given.

## 4.1 Designing Organizations

Recall that ODML representations are divided into two distinct classes: *templates* and *instances*. A template encompasses the range of all possible organizations that are to be considered, while an instance is a singular, particular organization derived from a template. The key difference is that a template depicts the organizational choices that must be made, while in an instance those choices have been decided.

The process of designing an organization consists of searching the *organizational space* defined by the template and selecting an appropriate instance. This organization space is defined by decision points that exist in the template. These decision points manifested in two ways: with variables and with has-a relationships.

For example, the sensor network example explored the effects of changing the size of a sector. In the model, a *sensors\_per\_sector* variable exists in the *environment* node. This variable can take on one of several discrete numeric values that match the experimental conditions from Section 2.1 (e.g., 1, 2, 4, 9, etc.). The value of *sensors\_per\_sector* is then used to calculate *num\_sectors*, which controls the size of the has-a relationship *organization* has with *sector*. As demonstrated earlier, each of these choices results in an organization with different characteristics.

In the IR domain, the edge magnitude *topic\_mediators* shown in Figure 3.2a is a typical variable. This determines how many mediators will exist in the organization. It can be assigned different values, which will result in organizations that have different forms.

The second type of decision point revolves around how has-a relationships are satisfied. The magnitude or size of a has-a relationships can be controlled with a variable as above. The nodes that may be attached with a has-a relationship represent a more complicated space, because the relationship may be satisfied by a range of node *types*, and there may also be a number of existing instances of each type that are suitable. Consider a typical role-agent relationship, such as *sensor* has with *agent* in Figure 2.10a. Further assume that we are part way through organizational construction, and that two agents ( $a_1$  and  $a_2$ ) have already been created and assigned to one role each. In this case there are four ways to assign *sensor's agent*. A new instance of either *normal\_agent* or *robust\_agent* can be created, or it can be attached to  $a_1$  or  $a_2$ . Each will have different tradeoffs, and in some cases the decision may affect previously made decisions in other parts of the organizational structure

A similar relationship exists in the IR model. Because both *aggregator* and *database* are possible instances of *source*, the *mediator-source* has-a relationship represents the a decision point. In this case, mediators with the same number of sources can be further differentiated by the types of sources they manage.

In both of these cases, the quantitative effects of local decisions can have significant and complex non-local impact. For example, as a result of the has-a assignment above,  $a_1$  might have to divide its time between two roles. This could negatively affect the quality produced by the initial role, as well as the performance of any other structures that depend on that role. Hard constraints elsewhere in the structure that were previously satisfied may become invalid, and soft constraints may degrade. It is equally possible that all those values improve as a result of the decision, or that some local values degrade while higher achieving global utility. In general, the interdependencies between nodes and fields mean that values may be both nonlinear and non-monotonic as the structure changes.

The rationale behind creating a mechanism to define this organizational space is the fact that different organizations will be appropriate under different conditions [91, 32, 112, 24]. When the set of available resources is different, requirements and constraints shift, and capabilities and characteristics are varied in the available entities, different organizational structures will lead to different performance. Sections 2.3.8 and 3.3 outlined how the value or quality of an organization can be captured with a utility function, as embodied by a standard ODML constant *utility*. Using this, it is possible to define the set of optimal structures as those with a maximal value for *utility*. Figures 3.9 and 3.11 showed how these two concepts are brought together, by depicting the optimal organization instances derived from the information retrieval model under different conditions. The ODML template defines the organizational space using the decision points outlined above, and organizational quality or appropriateness is captured by the *utility* of each candidate instance. The search for an organization with sufficient or optimal utility is the basis of the automated organizational design process that ODML allows.

#### 4.1.1 Design Complexity

Given the two types of organizational decisions that must be made, one must next determine how best to explore the space of alternatives when searching for an appropriate organizational design. One approach is to simply generate full organizations and test them in turn. This is a perfectly valid approach, but there may be a great many such organizations to consider. A more incremental strategy is preferable, because backtracking from partially formed organizations prunes higher in the search tree, and can therefore be vastly more efficient than doing so only after a complete structure has been formed. However, because the organizational structure changes continually during such an incremental search and construction process, making correct predictions for nonlinear and non-monotonic values can be difficult. This makes it correspondingly difficult to predict the characteristics of the completed organization, so deciding when it is appropriate to backtrack is a challenging problem.

The combinatorics of the search space also conspire to complicate the search process. Implicit in the role-agent example above is the fact that if a new agent is created, the next role-agent relationship will have five choices instead of four. An entire space of role-agent relationships will also exist for each new choice of *sensors\_per\_sector*. In fact, the problem of finding even a valid organization, not necessarily the more desirable optimal or most appropriate one, will be shown below to be NEXP-Complete. This agrees with complexity results presented in related work by Nair and Tambe [138], who analyzed the complexity of the role assignment and execution problem. The high degree of complexity is due in part because an ODML template is an example of what is known as a succinct representation [140] of the organization it produces, which means that its output (an organizational instance) may be exponentially larger than its input (a template). Determining the validity of a candidate instance can only be performed by looking at each node in the instance, and can therefore require exponential time to complete.

I will refer to the process of determining if a satisfying instance exists in the space defined by an ODML template as ODML-SAT. Determining the complexity of this problem will help determine how large the search space is likely to be, and how hard it is to find solutions in that space. This will ultimately guide and constrain the development of techniques to explore the organizational space defined by an ODML model. In this section, I will first show that ODML-SAT is in the class of problems with solutions that can be verified in time that is exponential in the length of the input. This is known as the NEXP complexity class, which contains problems that can be solved nondeterministically in exponential time. I will then show that ODML-SAT is NEXP-Hard, by reducing the general form of a problem known to be NEXP-Complete to ODML-SAT. This will show that ODML-SAT is itself NEXP-Complete.

In these proofs, I will assume that the ODML structure in question does not contain recursive relationships. Structures that contain unbounded recursion have an infinite search space, and therefore the ODML-SAT problem would be undecidable in the general case. ODML does also allow a form of *bounded* recursion, where a particular node may be revisited only a specified number of times along any root to leaf has-a path in a valid organizational instance. In this case, the non-recursive set is a strict and simpler subset of such bounded recursive template instances. Bounded recursive templates can be converted to non-recursive equivalents by unrolling the recursion, adding in placeholder nodes as needed to represent the individual recursion levels.

**Lemma 4.1.1.** *Given any non-recursive ODML template containing  $n$  nodes, each with has-a relationships of size  $m$ , the maximum-sized instance derivable from that template will have  $S_n = \sum_{i=0}^n \binom{n}{i} m^i$  entities.*

*Proof.* In this proof we will be considering the parent-child structure of the organization formed by the has-a relationship. Specifically, we wish to know what arrangement of nodes formed by has-a relationships will produce the largest possible organization, i.e., the one with the most distinct entities in it. Assume the total number of allowed nodes for any given type is unbounded. Let  $n$  be input size, which is the number of node types in an ODML template,  $\{N_1, \dots, N_n\}$ . We will assume without loss of

generality that each node  $N_i$  will have a single has-a relationship for each node type  $N_j$ ,  $n \geq j > i$  (to avoid recursion) and that each such relationship is of size  $m$ . There will therefore be  $m$  children created by each relationship.

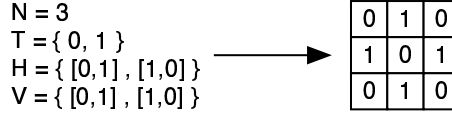
If  $n = 0$ , the organization consists of only the empty root organization. If  $n = 1$ , there is only one arrangement, which by our assumption will contain  $m + 1$  entities, which is maximum. Assume that there is an arrangement of  $n = k$  nodes that will produce an organization with a maximum number of entities  $S_k$ , where  $S_k = \sum_{i=0}^k \binom{k}{i} m^i$ . The maximum sized, non-recursive organization with  $k + 1$  node types will add a has-a relationship of size  $m$  from all existing nodes  $\{N_1 \dots N_k\}$  to the new node  $N_{k+1}$ . Any fewer added relationships will lead to a smaller organization instance, and any more relationships, or any relationships of size greater than  $m$ , violate our base assumptions. This will result in a new instance which has all the entities from the previous maximum sized instance, plus  $m$  new entities of  $N_{k+1}$  for each of those previous entities. The size of this new organization will be  $S_{k+1}$ :

$$\begin{aligned}
S_{k+1} &= \sum_{i=0}^k \binom{k}{i} m^i + m \sum_{i=0}^k \binom{k}{i} m^i \\
&= \left( \binom{k}{0} m^0 + \dots + \binom{k}{k} m^k \right) + \left( \binom{k}{0} m^1 + \dots + \binom{k}{k} m^{k+1} \right) \\
&= \binom{k}{0} m^0 + \left( \binom{k}{1} + \binom{k}{0} \right) m^1 + \dots + \left( \binom{k}{k} + \binom{k}{k-1} \right) m^k + \binom{k}{k} m^{k+1} \\
&= \binom{k+1}{0} m^0 + \binom{k+1}{1} m^1 + \dots + \binom{k+1}{k} m^k + \binom{k+1}{k+1} m^{k+1} \\
&= \sum_{i=0}^{k+1} \binom{k+1}{i} m^i
\end{aligned}$$

Because  $S_{k+1} = \sum_{i=0}^{k+1} \binom{k+1}{i} m^i$  matches the original premise, the result follows by induction.  $\square$

**Lemma 4.1.2.** *ODML-SAT is in NEXP.*

*Proof.* Assume we have an arbitrary ODML structure  $\mathcal{O}$  containing  $n$  node definitions, each of which has some number of has-a relations of size less than or equal to  $m$ . By Lemma 4.1.1 the largest organization derivable from a non-recursive ODML structure  $\mathcal{O}$  will contain  $O(m^n)$  entities. Because the number of decisions that must be made to create an organization is proportional to the number of decisions embedded in each template node and the number of entities in the final organization, the number of decisions is also  $O(m^n)$ . Therefore, if a satisfying organization exists, we can nondeterministically guess a corresponding decision sequence in exponential time. The instance itself may then be generated from this decision sequence in exponential time.



**Figure 4.1.** A sample TILING problem and consistent solution.

The validation step involves visiting each entity in the organizational instance, and verifying that its constraints are satisfied. At worst, a constraint may be based on all data possessed by all other nodes in the structure, which will require  $O(m^n)$  time to gather. Therefore, all entities may be validated in  $O(m^{2n}) = O(m^n)$  time. Because a satisfying solution to an arbitrary ODML structure may be nondeterministically discovered and validated in exponential time, it is in the NEXP complexity class.  $\square$

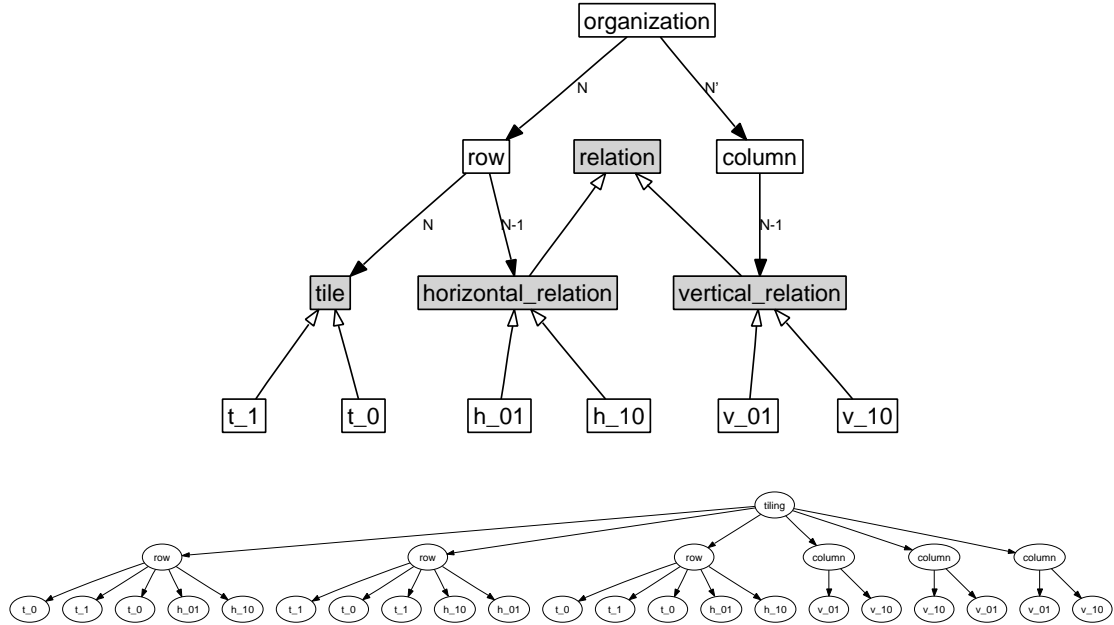
To demonstrate that ODML-SAT is NEXP-Hard, one must show that any problem in NEXP can be reduced to ODML-SAT in polynomial time. This is conventionally done through a reduction from an existing NEXP-Complete problem to the problem in question. Since any problem in NEXP can be reduced to any NEXP-Complete problem in polynomial time, reducing from such a problem is a straightforward way to achieve the larger goal.

The NEXP-Complete problem I will use is TILING, as defined in [140, 10]. A TILING problem consists of a set of tile types  $T = \{t_0, \dots, t_k\}$ , a grid size  $N$  in binary, and a set of horizontal and vertical compatibility relations  $H, V \subseteq T \times T$ . An  $N \times N$  tiling is a mapping  $f : \{0, \dots, N-1\} \times \{0, \dots, N-1\} \rightarrow T$ .  $f$  is consistent only if 1)  $f(0, 0) = t_0$  (the origin  $\langle 0, 0 \rangle$  has tile  $t_0$ ) and 2)  $\forall_{x,y} \langle f(x, y), f(x+1, y) \rangle \in H$  (all horizontal pairs are compatible) and 3)  $\forall_{x,y} \langle f(x, y), f(x, y+1) \rangle \in V$  (all vertical pairs are compatible). The TILING decision problem is to determine, given  $T, N, H, V$ , if a consistent tiling exists. An example TILING problem and consistent solution can be seen in Figure 4.1.

**Lemma 4.1.3.** *TILING*  $\leq$  *ODML-SAT*

*Proof.* Any TILING problem with inputs  $T, N, H, V$  can be reduced to ODML-SAT in the following way. First, construct an ODML model containing the TILING problem inputs. The template for such a model, along with an example solution, can be seen in Figure 4.2. The source for this particular model, which corresponds to the problem shown in Figure 4.1, can be found in Appendix E. For each tile  $t_n \in T$  there will be a corresponding  $t_n$  node that has an is-a relation with the abstract node *tile*. Similarly, each compatibility relation  $h \in H$  and  $v \in V$  will be represented by a node having an is-a relationship with *horizontal\_relation* and *vertical\_relation*, respectively. The *organization* consists of  $N$  rows of  $N$  tiles that make up the mapping. The size of this corresponding template grows linearly with the number of the TILING inputs, and thus can be constructed in polynomial time.

The organizations derived from this template incorporate the elements of candidate mappings in the original TILING problem. The high level *organization* contains



**Figure 4.2.** (top) An example ODML template used to reduce a TILING problem. (bottom) A valid organizational instance created from that template.

the  $N$  row nodes of  $N$  tiles that make up the  $N \times N$  grid. Because *tile* itself is abstract, each has-a relation must be satisfied by one of the  $t_n$  nodes present in the template. Each *row* also has  $N - 1$  *horizontal\_relation* nodes, corresponding to the  $N - 1$  pairs of tiles in the row. The *vertical\_relation* nodes contained by each *column* are used for similar purposes. *column* itself lacks has-a relationships with *tile*, instead referencing those contained by the *row* nodes directly.

The numeric values embedded in these nodes are used to ensure the consistency of the mapping. Each *tile* has a *type* field corresponding to the type of the original tile. Each *relation* has fields  $t1t$  and  $t2t$ , corresponding to the two tile types specified by the corresponding original relation in  $H$  or  $V$ . The compatibility restrictions are modeled using constraints in each of these nodes. Each *relation* contains a pair of constraints, specifying that the types of the two *tile* nodes it corresponds to must match  $t1t$  and  $t2t$ , respectively. The origin condition of the TILING problem is represented with a constraint field in *organization*, which states that the tile at  $\langle 0, 0 \rangle$  must have type  $t_0$ .

If the original TILING problem had a consistent mapping, then there will exist a valid organization. Cell  $\langle 0, 0 \rangle$  in the organization will contain  $t_0$ . Each cell  $\langle x, y \rangle$  in the original mapping can be used as a choice of *tile* node for the corresponding row  $x$  column  $y$  in the organization. Each horizontal or vertical compatibility relation relied upon in the original mapping may be selected to satisfy the corresponding *relation* has-a relationship in each *row* and *column*. All constraints in this organization will be satisfied, and therefore it will be valid.

If a valid organization can be found within the constructed model, then a consistent mapping will exist in the corresponding TILING problem. The origin cell will contain  $t_0$ . Each choice of *tile* for row  $x$  column  $y$  can be used to specify the contents of grid cell  $\langle x, y \rangle$  in the mapping. Each *horizontal\_relation* and *vertical\_relation* represents a valid selection from the appropriate compatibility lists for each horizontal and vertical pairings in the grid.

Because an appropriate ODML-SAT problem can be created in polynomial time from the TILING inputs that contains a valid organization when a consistent mapping exists, and does not contain a valid organization when no mapping exists, TILING  $\leq$  ODML-SAT. □

**Theorem 4.1.4.** *ODML-SAT is NEXP-Complete*

*Proof.* By Lemmas 4.1.2 and 4.1.3, and because TILING is itself NEXP-Complete [140], ODML-SAT is NEXP-Complete. □

## 4.2 Algorithmic Search Techniques

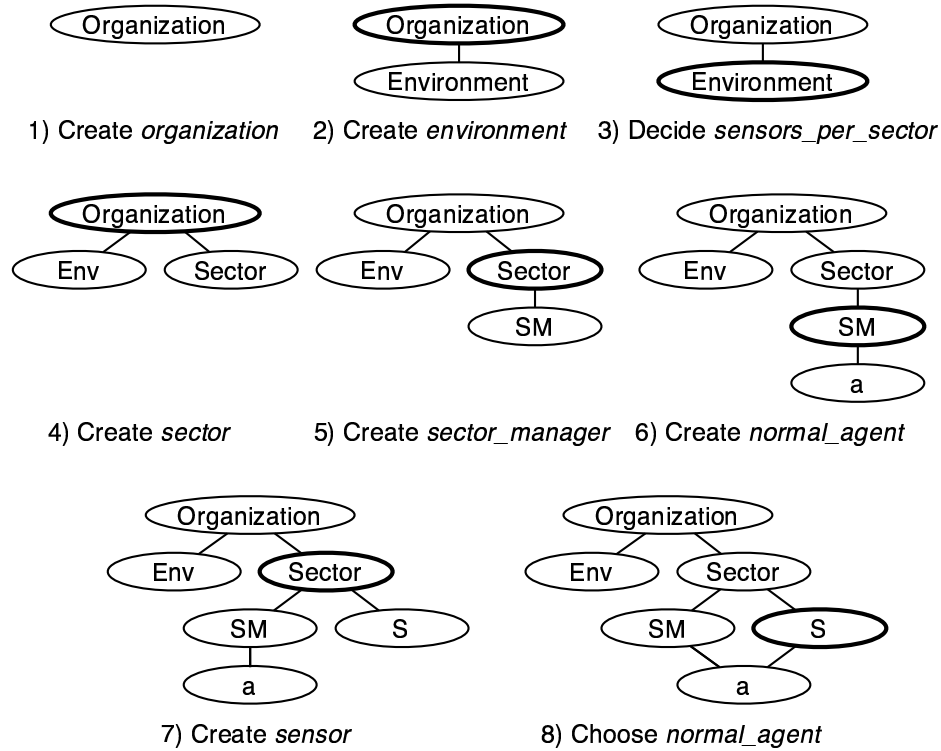
Strategies to cope with the potentially large and complex search space can be placed into two different categories: *algorithmic* and *heuristic*. The key difference is that algorithmic search techniques will always find an optimal solution if one exists, while heuristic techniques may not. The former will be discussed in this section, while the latter are covered in Section 4.3. The reader may also refer to Section 4.4.2.1, which describes a direct, distributed approach to reduce search time.

Because of the computational complexity of ODML-SAT, a combinatorially feasible algorithmic technique that works for all search spaces seems unlikely. However, there may be algorithmic techniques that work for certain classes of problems, or offer benefits to all problems without a formal reduction in complexity. The strategies presented in this section fall into this category. I will assume that they are used as part of an initially exhaustive search of the organizational space. The base assumption is that all organizational possibilities will be explored, and that the technique will eliminate or avoid some of these possibilities. To be correct, a valid candidate with optimal utility (if one exists) must be present in the subset of those possibilities that are remain after applying the algorithmic technique.

Recall that all nodes in an ODML template are descendants of the *organization* node. The search process begins by deciding any variables defined in that node, and continues by instantiating or selecting nodes to satisfy any has-a relationships one-by-one. As each has-a node is created, the search progresses in a depth-first manner by deciding the variables and has-a relationships in each node. If an existing node is selected to satisfy a has-a relationship the initial decisions made in that node are retained.

An example of this process using the DSN model is shown in Figure 4.3, bold outlines indicate the node where the indicated action is being performed. Step 3 shows a variable being decided, while step 8 shows an existing node (agent  $a$ ) being





**Figure 4.3.** The first eight organizational decisions made during a search of the DSN organizational template.

used instead of creating a new one. Note that the incremental nature of this process means that at any given time during the search, the organization may be only partially instantiated, and any organizational values needed for analysis are calculated from only the currently existing nodes and relationships.

The techniques described in this section revolve around this choice selection process, by deciding what choices to pursue and which to ignore. Section 4.2.1 describes and demonstrates a technique that uses the trend estimation of a value to determine if currently unsatisfied constraints over that value are in fact unsatisfiable. Section 4.2.2 shows how choices can be grouped into equivalence classes, which can be used to avoid redundant search. Section 4.2.3 describes how a general mathematical solving engine is incorporated into the search process, which shows both how such tools can be exploited and the relationship ODML has with a simple system of equations. The remainder of this section briefly describes three other search techniques that show promise.

### 4.2.1 Exploiting Hard Constraints

The first technique that I will explore is how to correctly bound the search by exploiting the hard constraints that exist in the model. This is accomplished by determining

the numeric trend of a constrained value, and stopping the search if it can be determined that the constraint is unsatisfiable based on that trend. For example, if a value is already too high and its trend indicates it will only increase as a result of later decisions, no further search needs to be performed along that branch of the decision tree. In this section I show how these determinations can be made from the information contained in an ODML model and under what conditions this technique is applicable.

Recall that all hard constraints must be satisfied in an organization for it to be considered valid. If a constraint has become unsatisfied during the course of an organizational search, it may be therefore be reasonable to halt the search and backtrack from that point. Two issues complicate this process. The first is that constraints may be initially unsatisfied and only become satisfied through the course of the decision making process. The second is that, because values may change non-monotonically, a constraint that is currently satisfied or unsatisfied can change its state repeatedly during the instantiation process. Because valid organizations may potentially be missed in either case, a strategy that blindly backtracks when an unsatisfied constraint is observed is incorrect.

There are two ways that a value may change during the instantiation process: 1) through the application of a modifier on that value; and 2) through a change in a field the value's expression is dependent on. Both these effects are caused by organizational changes that take place after the value in question has been initially instantiated. For example, in the sensor network domain, a sector manager's communication load will increase as the number of sensors in its sector increases. Relating a previously instantiated node to a different part of the structure, which occurs when such a node is selected to satisfy a new has-a relationship, will also usually result in value changes within the node. For example, if the agent serving as that sector manager is later bound to a sensor role as well, that agent's communication load will increase.

Because modifiers and dependent fields are also based on potentially fluctuating values, a change in one location may propagate through the organization, affecting the values of many different fields. For example, the global utility of the organization may be reduced in the above example because the increased average RMS error is indirectly linked to agents' communication load.

In the general case, this characteristic precludes correctly bounding the search because one does not know, for example, if a currently unsatisfied constraint will later become satisfied as more of the organization is realized. More insidiously, it is also possible for a currently satisfied constraint to become unsatisfied for the same reasons. Carried to its logical extreme, this means that the decision process cannot be prematurely bound, and constraints can only be evaluated after the organization has been constructed in its entirety. However, the combinatorics of the organizational search and evaluation process are significant enough that this situation is untenable.

Fortunately, there are some situations where it is possible to dismiss alternatives out of hand by incorporating higher-level knowledge about the organization as a whole. For example, in the sensor domain a single agent cannot control more than one sensor. Each agent has a *sensors\_controlled* value, which is initially zero and later incremented using a modifier when it is assigned to a sensor. The one sensor per agent

restriction is modeled by a hard constraint, which specifies that *sensors\_controlled*  $\leq$  1.

If there are  $n$  sensors and  $a$  agents then there are  $a^n$  possible assignments of agents to sensors, but only  $\binom{a}{n}$  of them are valid according to the *sensors\_controlled* constraint. If it were possible to detect when an invalid assignment had been made before all organizational decisions have been completed, one could bound the search at that point and backtrack to where the constraint was satisfied. Because role assignments made during the search process are permanent, a domain expert performing a similar search by hand would know that there is no decision that could be made in the future to reduce the number of sensors controlled by an agent. Therefore, it is reasonable and correct to bound the search and backtrack if an agent is ever found to control more than one sensor, because that constraint can never be satisfied as a result of later decisions.

Because of its inherently myopic view, the organizational search does not have the same information available to the domain expert. However, the relationship that the expert's knowledge is based upon is represented in the organizational template, in that *sensors\_controlled* is affected by only that one type of modifier from the *sensor* node, which increments the value by a positive constant. If it is possible to computationally deduce such conditions from a gross analysis of an organizational structure, then it will be possible to appropriately bound the search.

Based on this premise, if a constraint is currently unsatisfied, one has the choice of allowing the violation to exist in the hopes that it will be resolved by some future decision, or backtracking under the assumption it will never be satisfied. It is possible to analyze the data dependencies that are expressed by the organizational representation to make this decision. In particular, by evaluating the equations that model the constraint and the modifiers that have the potential to affect its value, it is possible to estimate how future decisions might affect the constraint's satisfaction. In essence, the analysis will deduce the numeric trends of each term in the constraint relation and use that to determine satisfiability. This analysis can be done computationally, based on the information modeled in the ODML template.

The algorithm used to determine satisfiability is outlined in Figure 4.4. The constraint is comprised of two parts, the target's value ( $v$ ) and the expression ( $e$ ) that the target is compared against. For example, if a constraint states that  $x < y + 2$ ,  $x$  is the target and  $y + 2$  is the expression. The target is usually a constant or variable defined elsewhere in the node with its own expression, from which its base value may be calculated. To begin, one must determine the theoretical trends of both components. It will then be possible to determine satisfiability by comparing these trends. For example, if the value's trend is to monotonically increase and it has already exceeded the constraint's constant value, it is unsatisfiable.

The trend estimation process starts by determining what fields the characteristic is dependent on by enumerating the fields referenced by the characteristic's expression. For example, if  $x = a + 2b$ , its set of fields  $D$  will be  $\{a, b\}$ . The bounds and trend of each symbol's value must therefore be estimated to determine the trend of  $x$ .

A symbol that has no dependents or incoming modifiers is considered constant (fixed). The range of the value may be determined immediately by evaluating its

```

is_satisfiable(constraint  $c$ )
  if ( $c$ .satisfied) return true
   $t_v \leftarrow$  find_trend( $c$ .LHS)
   $t_e \leftarrow$  find_trend( $c$ .RHS)
  if ( $t_v = \text{unknown} \vee t_e = \text{unknown}$ ) return true
  if ( $t_v = \text{constant} \wedge t_e = \text{constant}$ ) return false
  if ( $t_v = \text{increasing} \wedge$ 
      ( $t_e = \text{constant} \vee t_e = \text{decreasing}$ )) return false
  if ( $t_v = \text{decreasing} \wedge$ 
      ( $t_e = \text{constant} \vee t_e = \text{increasing}$ )) return false
  return true

find_trend(expression  $e$ )
   $D \leftarrow e$ .dependencies
   $M \leftarrow e$ .modifiers
   $t \leftarrow \text{constant}$ 
  for ( $f \in D \cup M$ )
     $t_f \leftarrow$  find_trend( $f$ )
     $d_f \leftarrow \frac{\partial e}{\partial f}$ 
    if ( $d_f$ .dependencies  $\subseteq \{ f \} \wedge d_f$  is linear)
       $t_f \leftarrow \nabla d_f$  (i.e., wrt  $f$  and  $t_f$ , increasing, decreasing or constant)
    else  $t_f \leftarrow \text{unknown}$ 
    if ( $t_f = \text{increasing} \vee t_f = \text{decreasing}$ )
      if ( $t = \text{constant}$ )  $t \leftarrow t_f$ 
      else if ( $t_f \neq t$ )  $t \leftarrow \text{unknown}$ 
    else if ( $t_f \neq \text{constant}$ )  $t \leftarrow t_f$ 
  return t

```

**Figure 4.4.** Pseudocode for the constraint satisfiability and trend estimation procedures.

expression, and its trend will be to remain *constant*. Symbols which are not fixed reference other symbols that must themselves be analyzed. This analysis process is therefore recursive.

Having determined the bounds and trend of a particular symbol, one must also determine how it affects the value of the expression that references it. This can be done by taking the partial derivative of the expression with respect to the symbol in question. If both the derivative and the dependent symbol's trend are monotonic, then we may infer the behavior of the target's value with respect to that symbol. If either is not monotonic then the target's trend is considered unknown, which indicates the technique is not applicable.

Recall that modifier fields elsewhere in the organization can also affect a constant's value. By searching the organizational template, it is possible to find any and all modifiers  $M$  that have the capacity of affecting a particular constant. For each

modifier one must first determine the trend of the modifier’s expression, and next determine how the modifier can affect the constant’s value. This is performed in a manner similar to the analysis of the expression’s dependent fields. The only difference is that the partial derivative is calculated from a combined expression that includes the potential cumulative effects of the modifier. This will be described in more detail below.

If the effect of each dependent field and modifier is known and predictable and their aggregate effect is coherent, the overall trend of the expression’s value may be estimated. For example, if all individual trends are incremental, the overall trend will be *increasing*. If all the interactions decrease the value of the expression, it is *decreasing*. If some symbols have the capacity to decrease while others increase, the trend is considered *unknown*.

In this way the trends of both the constraint’s target and its expression may be determined. One can use this information to determine if the constraint is unsatisfiable or potentially satisfiable. This is outlined in the series of conditional checks made in the *is\_satisfiable* function from Figure 4.4. If this function returns false, then no organizational decisions exist that have the ability to satisfy the constraint and the search should backtrack. A response of true implies either that such decisions do exist, or conflicting trends make it not possible determine satisfiability using this particular analysis technique.

#### 4.2.1.1 Monotonic Trends in the DSN Model

With the high-level procedure outlined above, I will show how the process works in more detail. This section will describe its use on the distributed sensor model, while the section following it will show when the analysis breaks down.

Once a constrained characteristic has been identified that must be checked, the first step in the process to map dependencies between that characteristics and other fields in the organization (i.e., determining  $D \cup M$  from Figure 4.4). For example, if the expression defining a constant contains references to other fields, then that constant’s value is influenced by and dependent on those fields. Similarly, if a constant can be affected by one or more modifiers, one must identify those modifiers before one can determine precisely how they affect the constant. Because nodes may be passed as parameters to other nodes, and because modifiers may affect non-local fields, dependencies have the potential to originate from anywhere in the organization. By using the information stored in the organization template, it is possible to discover these dependencies and form a dependency graph able to serve as the foundation for the recursive trend estimation process.

A dependency graph generated from a subset of the sensor organization is shown in Figure 4.5. Constants (oval), constraints (rectangle), modifiers (pentagon), variables (trapezoid), parameters (triangle) are all represented. Edges indicate data dependency relationships, where the target of an edge is dependent on the edges’ source. Although this graph is fairly complicated, only the *agent* node at the bottom left corner of the figure is relevant for this discussion. One can see how the earlier sensors controlled example is reflected here. In the *agent* node there is the *sensors\_controlled*

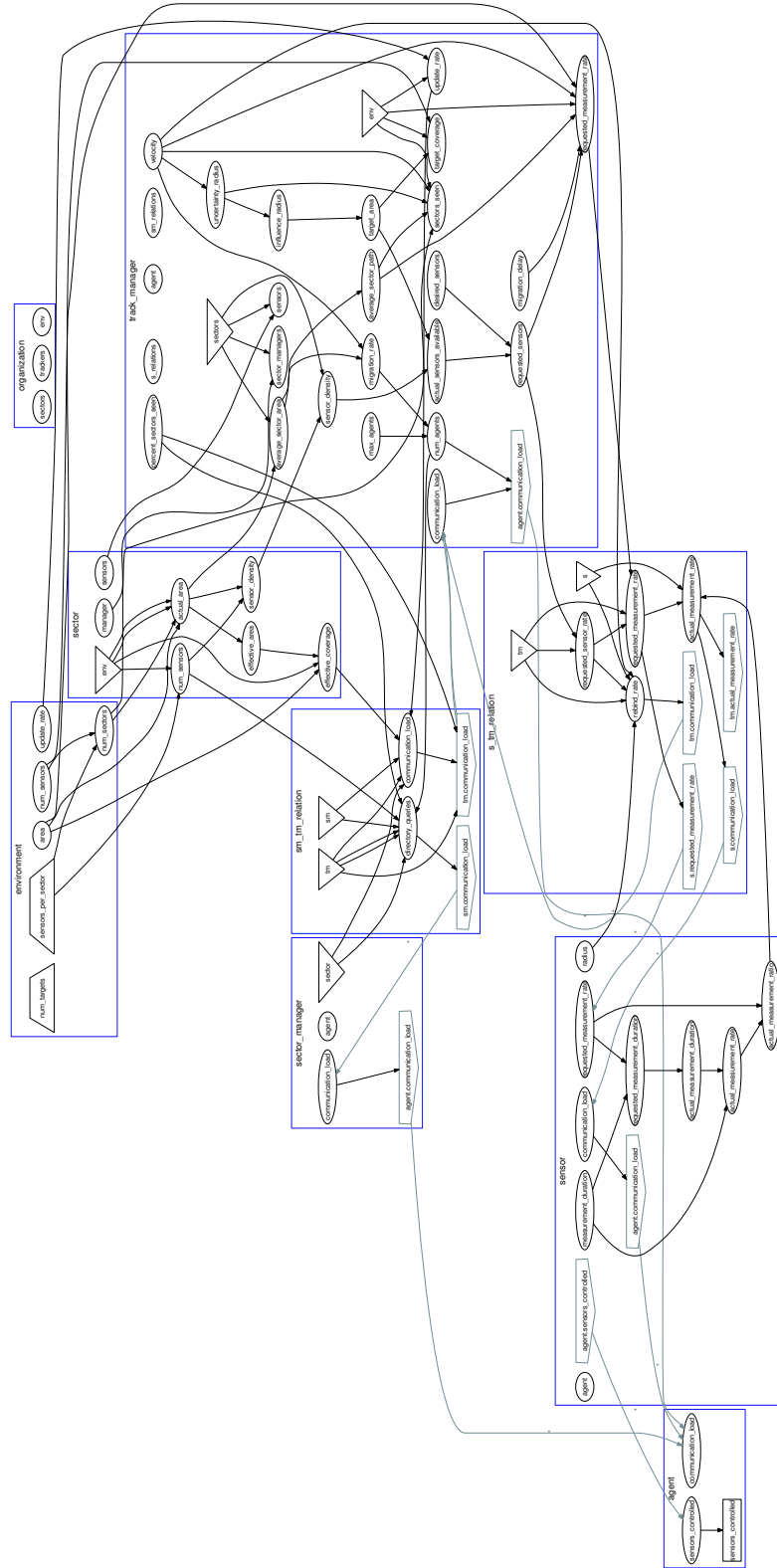


Figure 4.5. Partial dependency graph for the distributed sensor network model.

constraint in question. This is influenced only by the local *sensors\_controlled* constant. That constant is itself only affected by the *agent.sensors\_controlled* modifier in the *sensor* node. This information, along with the underlying expressions they represent, is sufficient to support the trend analysis. The relevant fields are as follows:

```

<node type="sensor">
  ...
  <modifier name="agent.sensors_controlled" op="+">1</modifier>
</node>

<node type="agent" abstract="true">
  ...
  <constant name="sensors_controlled">0</constant>
  <constraint name="sensors_controlled" op="<=">1</constraint>
</node>

```

Because neither expression in *agent* references other fields directly, both trends are zero and the values are locally constant. The same is true of the modifier's expression. The modifier *agent.sensors\_controlled* ( $M_{a.sc}$ ) has an additive effect on its target *sensors\_controlled* ( $C_{sc}$ ). Because we do not know *a priori* how many times the modifier might be applied, a new variable  $n_{a.sc}$  is introduced to stand in for this quantity. The resulting expression for  $C_{sc}$  that includes the cumulative potential effects of these modifiers is:

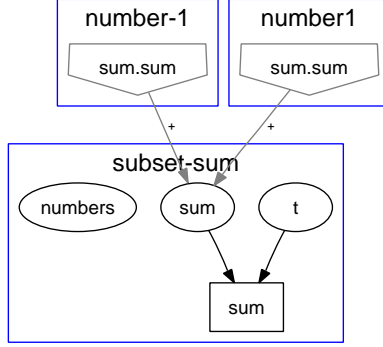
$$C_{sc} = 0 + n_{a.sc}M_{a.sc}$$

The trend of the effect of this modifier can be computed as:

$$\frac{\partial C_{sc}}{\partial n_{a.sc}} = M_{a.sc}$$

Because each individual modifier's expression  $M_{a.sc}$  is a fixed positive constant, and because no other values influence the constant, we can conclude that the value of *sensors\_controlled* will monotonically increase regardless of what organizational decision is made. Since the constraint in question is unsatisfied only when the value is too high, one can deduce that, once violated, it will remain unsatisfied regardless of succeeding decisions.

Part of the power of this technique stems from the ability to exploit constraints over just the subset of characteristics that are predictable, despite the fact that many other unpredictable trends may exist in the model. Figure 4.5 shows that the relationships and nodes involved in the *sensors\_controlled* characteristic are relatively disconnected. Compare that to the *communication\_load* characteristic of *agent*, which has a much more complicated lineage. It is influenced by modifiers in each of the three roles, which are themselves indirectly influenced by nearly every other field in the graph. The complexity here is not a barrier. In fact, it is quite possible that such a tangled web of interactions could simplify to the same type of "always increasing" behavior exhibited by *sensors\_controlled*. In this case, however, the web includes elements that are not differentiable. Because of this, it is not possible to exploit a constraint governing the *communication\_load* characteristic.



**Figure 4.6.** Dependency graph for the SUBSET-SUM example.

#### 4.2.1.2 Non-monotonic Trends in a SUBSET-SUM Model

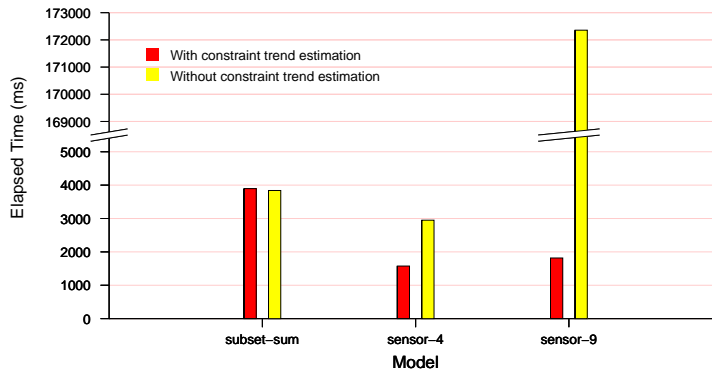
Similar non-monotonic behavior was observed in the IR model defined in Section 3.3. There it was shown that the *query\_rate* and *work\_load* of an individual could change as an indirect result of decisions made elsewhere in the structure. In fact, both the distributed sensor network and information retrieval models have numerous examples of unpredictable trends. However, as the agent’s *communication\_load* characteristic demonstrates, the great complexity and length of those data flows make them cumbersome examples. Instead, I will show a simple model of a SUBSET-SUM problem, and use that as a concrete example of when this type of analysis breaks down.

SUBSET-SUM is a classic example of an NP-Complete problem. A SUBSET-SUM problem is typically formulated as follows: given a target  $t$  and some set of integer numbers  $S$ , is there some subset  $N \subseteq S$  such that  $\sum_{n \in N} n = t$ . As with the TILING problem, SUBSET-SUM can also be reduced to an ODML model. Begin by creating an organization containing a constant *sum* with a value of zero and a constraint such that  $sum == t$ . Create an *number\_empty* node with a size constraint of  $|S| - 1$ . For each  $s \in S$ , create a *number\_s* node which contains a modifier that increments *sum* by  $s$  and has a size constraint of 1. Finally, add to the organization a has-a relation of *number* nodes with a size of  $S$ . This reformulation is proportional in size to  $S$ , and can be constructed in polynomial time. An example ODML model corresponding to the SUBSET-SUM problem  $t = 0$ ,  $S = \{-1, -1, 1, 1, 1\}$  be found in Appendix F. If a valid organization can be found, the sum of the numbers must equal  $t$ , and the *number* nodes used in that organization represent a satisfying subset. If no such subset exists, no candidate organization will satisfy the organizational constraint.

Consider the SUBSET-SUM dependency graph shown in Figure 4.6, which corresponds to the example problem above. The constraint target is *sum*, while its expression is  $t$ .  $t$  is fixed, so we can precisely determine its bounds (in this instance) to be  $[0, 0]$  and its value constant. Let  $K_{sum}$  be the constraint in question. Because  $t$  is constant,  $\frac{\partial K_{sum}}{\partial t} = 0$ , and  $\nabla_{K_{sum}} = \langle 0 \rangle$ , we can assume the constraint’s expression is invariant.

The constraint’s target *sum* is locally defined by the constant  $C_{sum}$  to be zero, so changes do not directly arise from the definition. Two modifiers affect this value,





**Figure 4.7.** A comparison of the search time differences with and without the constraint estimation algorithm across three different models.

however, which can affect the value’s trend. Let  $n_1$  and  $n_{-1}$ , and  $M_1$  and  $M_{-1}$  represent the number of times and values of the *number1* and *number-1* modifiers might be incorporated, respectively. The constant *sum* can be rewritten more completely as:

$$C_{sum} = 0 + n_1 M_1 + n_{-1} M_{-1}$$

The trend of *sum* in then:

$$\nabla_{C_{sum}} = \langle M_1, M_{-1} \rangle$$

$M_1$  and  $M_{-1}$  are each fixed, so this trend can be rewritten as  $\langle 1, -1 \rangle$ . Because the one term decrements while the other increments, the two trends conflict. We can not extrapolate trends from this analysis that would allow one to determine if  $K_{sum}$  is likely to be satisfied by future decisions. Because of this, the gross trend is unknown, and the constraint is verifiable only when the complete structure has been constructed and no decisions remain.

Although the template analysis showed that the SUBSET-SUM search could not be reduced by taking advantage of constraint violations, it is still possible to make use of the trend information to improve the runtime performance of the search. When a conclusive trend for a constrained value cannot be found, it may no longer be worthwhile to check the constraint at all. If we assume that no higher-level heuristic is guiding the search based on current constraint satisfaction, and that the search will not be truncated because of a constraint violation, then we can assume the search will progress regardless of the constraint’s state. Because determining if the constraint is satisfied can be an expensive process, one can save time by simply ignoring unverifiable constraints until the entire instance has been generated.

### 4.2.1.3 Results

The potential benefit of using this technique has been demonstrated through a series of experiments, the results of which are shown in Figure 4.7. A pair of trial types

were evaluated using three different models, one that bound the search using the constraint trend estimate, and one that did not. The search process is deterministic, and the timing differences for identical trials on the same dedicated processor were negligible, so only a single trial was performed for each test.

The subset-sum tests show how search performance does not improve when using the technique in that domain, because of the non-monotonic trends discussed in the previous section. The two other scenarios (sensor-4 and sensor-9) were performed using the DSN model, one with four sensors and one with nine. The technique is seen to be quite beneficial in that domain, as the time elapsed in the “without” trials quickly exploded while the “with” remained low. A third DSN scenario involving 18 sensors was also tested (but is not shown), where the trials using constraint estimation finished after 2000 ms on average, and those without had not completed a single trial after three days of computation. These results are consistent with the discussion and motivation presented above.

The utility and significance of the benefit imparted by this technique depends on the model itself. In models lacking constraints, or only containing constraints over non-monotonic characteristics, no performance improvements will be realized. In models possessing constraints over monotonic characteristics, the amount of improvement will depend on how large the search space is, and how much of that space can be avoided by the technique. In the sensor-4 scenario, for example, the search space was relatively small so only modest gains were observed. The space of the sensor-9 scenario was much larger because of the increased number of role assignments to be made. Additionally, the *sensors\_controlled* constraint could be checked early in the decision tree, as a violation was detectable immediately after the conflicting sensor role assignment was made. Applying the trend estimation technique early allowed very little work to be wasted, resulting in the dramatic savings in search time. Therefore, although the performance gains are difficult to characterize in the general case, this shows that careful structuring of the model can allow the designer to take advantage of the technique.

## 4.2.2 Equivalence Classes

The second algorithmic technique I will discuss exploits the idea of equivalence classes to reduce the search space. To do this, I must formalize the various ways that a has-a relation may be satisfied. Recall from Section 2.2.1 that  $\mathcal{N}$  is the set of node templates present in an ODML model. I will use dot notation to indicate characteristics of objects, using the elements initially presented in Section 2.2.1. For example,  $h.t$  is the type of has-a relation  $h$ , and  $N.I$  is the set of is-a relations possessed by node  $N$ . Then  $\mathcal{N}_t$ , the set of node templates which can satisfy type  $t$  either directly or because of inheritance, can be defined as:

$$\mathcal{N}_t = \bigcup_{N \in \mathcal{N}} (N.t = t) \vee (\exists i \in N.I i \in \mathcal{N}_t) \quad (4.1)$$

Let  $A$  be the set of node that have previously been instantiated during the search and currently exist. Then  $D_h$ , the domain of choices available to has-a relation  $h$  at this particular point in the instantiation process, can be represented as the set:

$$D_h = \bigcup_{N \in \mathcal{N}_{h,t}} a'_{N,t} \cup \{a \in A | a.t = N.t\} \quad (4.2)$$

$a'_{N,t}$  represents a newly created instance of  $N$ . The size of  $D_h$  grows with both the number of related templates and with the number of instances that have been created. Since this latter set necessarily grows as part of the process of instantiation, the domain of has-a relationships will tend to grow correspondingly, causing the decision process to become more challenging as the instantiation process progresses. I will continue by describing a technique that can be used to mitigate this growth in complexity.

If there are  $k$  such decisions which must be made to construct the organization, then the total number of complete paths in the corresponding decision tree is on the order of  $n^k$ . However, many of these paths may be the same, or at least functionally equivalent. Consider the case where one is deciding upon an agent to serve as a sector manager. There may be five previously instantiated agents, along with the option of creating a new agent, resulting in six elements in the decision's domain. Further assume that four of those agents are simply sensor controllers, while the fifth is both a manager of a different sector and a sensor controller. Note that choosing any one of the four sensor controllers will produce the same organization, because they are functionally equivalent with respect to this particular decision. By segregating this agent pool into a set of equivalent classes and choosing a distinguished representative from each pool, the domain can be cut in half to just three options.

More formally, one may define the *equivalence class*  $[a]$  of a particular element  $a \in D_h$  using an appropriate *equivalence relation* ( $\equiv$ ) over the set of elements in  $D_h$ .

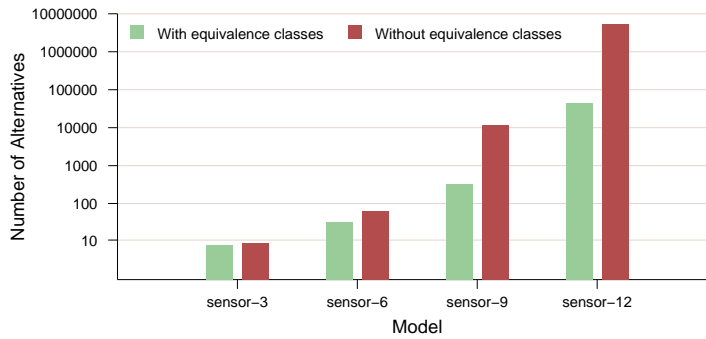
$$[a] = \{\alpha \in D_h | \alpha \equiv a\}$$

This function may be used to derive the *quotient set* ( $D_h / \equiv$ ) of the domain, consisting of all possible equivalence classes as created by the function.

$$D_h / \equiv = \{\forall_{a \in D_h} [a]\}$$

When  $h$  must be satisfied, the quotient set  $D_h / \equiv$  can be used in place of  $D_h$ , choosing a single member of each class to act as the representative of that class when evaluating alternatives. Because the quotient set is at most as large as the original set, this provides the opportunity to reduce the search space, without a corresponding reduction in utility.

Although this segregation does not affect the combinatorics of the decision process in general, it can still have a significant impact on the running time of the search. Consider an extreme but common example from the DSN model. Assume that 99 *agents* have been created so far in the search process, all of which have been assigned



**Figure 4.8.** A comparison of the number of alternatives that are considered with and without equivalence classes across four differently sized sensor networks. The models allowed designs with 3, 6, 9 and 12 total sensors, respectively.

to distinct *sensor* roles. A new *sensor* node has been created, and its *agent* has-a relation  $h$  must be satisfied. Without using equivalence classes, there will be at least 100 alternatives to evaluate in  $D_h$ . Next, let the equivalence relation  $\equiv_{sc}$  be defined to be true when the *sensors\_controlled* value of both nodes are equal and false otherwise. Because all existing agents fall into a single equivalence class, the quotient set  $D_h / \equiv_{sc}$  will contain only two possibilities, thereby avoiding 98 redundant alternatives that would otherwise have been examined.

In practice, the equivalence function for a particular decision is created using a set of *discriminators* that are associated with the has-a relationship. Implicit in the way this information is specified is the idea that different decision processes may have different equivalence functions, since the set of relevant characteristics may change in different contexts, even if they share the same underlying domain. For example, when searching for an appropriate *agent* to fulfill the *sensor* role, one might discriminate based on the agent’s *sensors\_controlled* field. When searching for an *agent* for the *track\_manager* role, the agent’s current *communication\_load* may be paramount. Each discriminator set consists of a list of arbitrary expressions similar to those described earlier. During instantiation, the search process calculates the value of each of these expressions for each member of the candidate set, which are then combined to produce a “fingerprint” for the instance. The equivalence function  $\equiv$  is defined as equality over these fingerprints; instances which have the same fingerprint will fall within the same equivalence class. As above, a single member of each set may be used to represent the entire class for has-a satisfaction purposes, thereby reducing the domain of the decision and the consequent complexity of the search.

#### 4.2.2.1 Results

The potential benefits of using this technique are shown in Figure 4.8, which compares the number of organizational alternatives (both valid and invalid) that are considered with and without equivalence classes across four sensor network design problems. The

“with” model has the *sensor* role create equivalence classes of *agent* nodes using the *sensors\_controlled* characteristic, as described above. The search process is deterministic, so only one trial was performed for each data point. The log-scale graph shows a dramatic decrease in the space of alternatives as the size of the organization grows, which correspondingly reduces the required time to search that space. The same optimal organization was found in both trials for each model. As described above, this improvement occurs because many of the candidate agents can be identified as redundant and ignored.

The significance of these results, and more generally the amount of benefit that one can expect from using this technique, depends on both the model and the designer’s choice of discriminators. The following section discusses this relationship in more detail.

#### 4.2.2.2 Selecting Appropriate Discriminators

Because these fingerprints are derived from expressions rather than simple constant values, one can easily create “fuzzy” fingerprints by mathematically abstracting away unnecessary detail. For example one of the sector manager’s agent discriminators is the agent’s *communication\_load*. If just a simple reference to that field is used, two agents with communication loads of 4.01 and 4.02 will be placed into different equivalence classes. If, in the context of a particular decision, those two values are functionally the same, the discriminator could be changed to

$$\text{round}(\text{communication\_load} \times 10).$$

Both loads are transformed to a new value of 40 in this new formulation, eliminating any detail past the first decimal point and placing them in the same class. This change exists only within the discriminator definition, so such a transformation has no effect on the original values as it is used solely for the purposes of classification.

Selecting the correct attributes to make up discriminator set, like the template design as a whole, is largely a domain-specific process. Typically, a set of local discriminating characteristics can be readily and intuitively created; an agent’s capabilities or current load or a resource’s capacity are common examples. The particular set that is chosen does require a working knowledge of the model itself, and must be selected with care. A poor selection of discriminator may be so precise that the resulting classes are too small to produce a measurable benefit. The opposite situation, when an overly broad discriminator places (what should be) distinct candidates in the same equivalence class, is worse still. In this case viable organizational alternatives may be incorrectly ignored.

A similar but more insidious problem occurs when superficially identical nodes are in fact organizationally different. Consider, for example, two agents *A* and *B* which are in all local respects identical, and have been assigned the same role  $R_1$  in two different parts of the organization. Assume agent *A* as  $R_1$  is part of a team, while *B* operates independently in that role. Assume some new role  $R_2$  must be assigned an agent, and that it will impose additional load on the agent that will impair the agent’s ability to serve as  $R_1$ . If *A* is assigned to  $R_2$ , the team it is part of may

be able to make up for  $A$ 's reduced performance, while  $B$  has no such backup. The characteristics of the final organization will differ depending on the decision, despite the fact that the two choices are myopically the same. This example demonstrates the potential drawback of using the equivalence set technique for organizational design. If the discriminator sets used for  $A$  and  $B$  fail to capture this non-local difference, one of the corresponding organizations may never be considered because it is "hidden" from the search process by the incorrect decision equivalence.

As with the underlying domain  $D_h$ , the quotient set for a domain is not static during the instantiation process, because equivalence class membership will change as instances are created and used in different circumstances. For example, assume that the discriminator in question is the set of roles an agent has been assigned to. If a new agent is created to fulfill the sector manager role, the new agent instance may fall into a new equivalence class if no other existing agent instance has that single role assignment. Similarly, when an existing agent instance is used in a new context, as when it takes on a second role, it will switch to a new or different equivalence class when using this discriminator.

While this same technique could be used to reduce the domain of variables, I currently assume the designer has performed this process as part of the template specification, and all domain members exist in distinct equivalence classes.

### 4.2.3 Using General Mathematical Solvers

Because so much of an ODML model is grounded in ordinary mathematical expressions, a natural direction to explore is the applicability of existing, general-purpose numeric solving engines to the organizational search process. For example, a suitable translation can allow commercial tools such as Mathematica [209] or Maple to handle the symbolic information provided by an ODML. Because these tools generally incorporate a range of analysis techniques and are optimized for processing mathematical expressions, they may be able to address aspects of the search space in a manner that is more direct and efficient than what has been presented thus far. Ideally, this would result in a savings in both computational time during the search and in the complexity of the ODML-specific search algorithm.

In addition to the efficiency and complexity benefits, by demonstrating how to map portions of an ODML model into such a tool I also hope to create a conceptual bridge between the two representations. To those that are familiar with the tool, such a mapping can then serve to illustrate the decision problems represented in the model and make the underlying concepts more accessible. The remainder of this section will explain how such a mapping was created and exploited by the search process.

The engine that was selected for this mapping was Mathematica, because of its ability to perform symbolic reasoning and the relative ease with which the Mathematica kernel can be remotely accessed by a separate computational process [96]. The first part of this process is to identify what aspects of the search are suitable for translation, with respect to the capabilities provided by Mathematica. Particularly relevant in this context are Mathematica's ability to symbolically represent and evaluate expressions, its high-level list and value manipulation functions, and its ability

find a set of variable assignments that optimize (maximize) the value of a nonlinear expression.

As discussed at the beginning of Section 4.1, there are two types of decision points that must be confronted by the design process: has-a relationships and variable assignments. The choices made for the has-a decisions decide the structure of the organization, which in turn decide the arrangement of the underlying equations. Rather than simply assigning values in expressions, this process manipulates the expressions themselves through the incorporation or removal of equations as structural components are added or removed as a result of different has-a assignments. This decision process does not play to Mathematica's strengths, which in its optimization mode exists primarily in its ability to manipulating values. Because of this, I do not see Mathematica assisting with this part of the design process. A more fundamental detail is that it is primarily this point that differentiates a pure mathematical model from that provided by ODML. Where a mathematical model is able to find a solution to a set of equations, and ODML model is able to capture problems where such a set of equations is just one alternative among many that must be considered.

Conversely, Mathematica is much better suited to the selection of values for variable fields. If we consider just a single variable and the set of expressions it influences, then making a choice for the associated decision point can be thought of as a standard numeric optimization problem. In this case, we wish to find the value for the variable that maximizes some characteristic (e.g., *utility*) subject to a set of constraints (e.g., the set of constraint fields present in the model). This process naturally extends to the case where there are multiple such variable fields that must be decided.

To successfully and correctly employ this type of optimization technique, the complete set of expressions relating the variables to the utility characteristic and constraints must be known. If we assume that the variables are linked through expressions to all parts of the organizational structure, the first step of the mapping process is to fix the set of expressions by deciding all known has-a relationships. Although I will not do so here, this condition may be relaxed if the technique is used on a subset of variables which are not so connected (see Section 4.2.5).

Recall the has-a relationship definition from Section 2.2.1. The requirement that all has-a relationships be decided implies that the specific *magnitude* of these relationships must also be known before the optimization mapping can be produced. These magnitudes can be defined either directly or indirectly in terms of variables, which means those variables must be decided before the optimization process and therefore cannot be addressed by this technique. The same is true of any variables that can affect parameters passed into nodes instantiated for the has-a relations. Choices for all other variables may be deferred during the organizational search process, and addressed *en masse* by a final optimization phase to complete the search. Thus, the first step in the mapping is to determine which variables must be decided and which may be deferred. This may be accomplished by using the dependency graph described in Section 4.2.1. After the graph has been created, but prior to initiating the search, the magnitude and parameter definitions for all has-a relationships defined in the ODML template are analyzed. Any symbols those definitions reference are noted, and the dependency graph is used to recursively flag all constants, modifiers and variables

they depend on. The decision for any variable that is not flagged by this process may be deferred.

After the partial organization has been instantiated so that only un-flagged variable decision points remain, the structure can be translated into a form that is appropriate for the optimization process. In this case, the following operations must be performed to create an input representation for Mathematica:

1. The expressions encapsulated by ODML's hierarchy of nodes must be flattened into a single list of equations and relations. Symbol references, which are normally interpreted within the node's namespace, must be rewritten to avoid conflicts in the new global namespace. Other syntactical changes to symbol names, numeric constants and lists must also be performed.
2. Where possible, the built-in functions provided by ODML (see Table 2.1), must be converted to an equivalent function in the new representation. For example:

$$forallprod(\bar{x}) \equiv \prod_{x \in \bar{x}} x \rightarrow \text{Apply}[\text{Times}[\bar{x}]]$$

If an equivalent function cannot be found, the optimization process will not be possible. Equivalent mappings currently exist for all functions except *map* and those dealing with discrete distributions (*E*, *V*, *Pr*, *mc*). A complete list is shown in Figure A.2.

3. The domain of deferred variables must be expressed. Currently this is accomplished with a complete disjunction. For example:

$$V = \langle 1, x^2, x + y \rangle \rightarrow (V == (1) \ || \ V == (x^2) \ || \ V == (x + y))$$

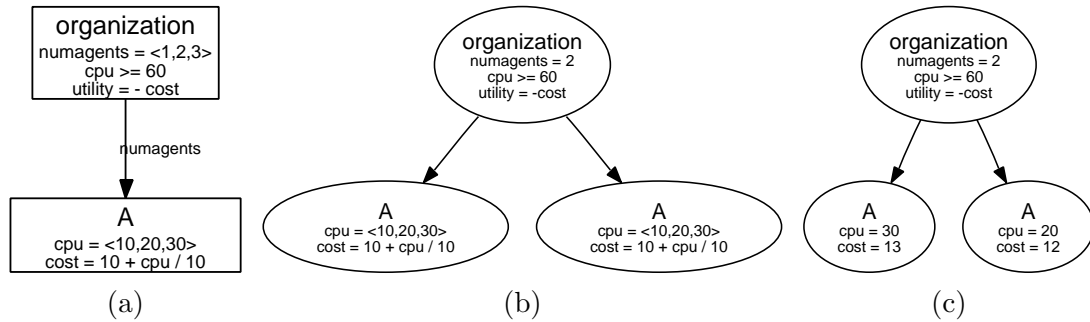
4. The variables and constraints must each been aggregated into lists that may be provided to the optimization function.

Additional details of this process can be found in Appendix A.

After undergoing this translation, the organizational space captured by the deferred variables can be searched by optimizing over the utility function defined by the model. It is worth noting again that because it is operating only on this more restricted space, this process is not solving the entire organizational problem. In this way, the use of a solver such as Mathematica can be viewed as just another analysis technique that can be employed as part of a the larger search of the organizational space. In practice, there will generally be many such invocations of the optimizer, because there will be many different partial organizations created by the earlier search phases, each of which may contain variable choices that have been deferred.

This process is illustrated by a simple example shown in Figures 4.9 and 4.10. Figure 4.9a shows the initial template. This organization consists of a simple group of agents, where each agent is represented by the node type *A*. The variable *numagents* indicates the different number of agents permitted in the group. The variable *cpu* in





**Figure 4.9.** Three stages from the search process using an external optimization engine. The original template (a), the instance with deferred variables (b), and the final organizational instance (c).

$A$  indicates the choice of processing power that the agent will take on. This is also reflected in the agent's *cost*, which is defined in terms of a fixed base value and a supplemental cost proportional to the *cpu*. The organizational *cpu* (defined as the total processing power over all agents) is constrained to have a minimum value of 60. Utility is negative of the organizational *cost* (defined as the total *cost* over all agents). Based on this, the optimal organization will meet some minimum level of processing power while minimizing its cost.

There are two variables present in this template, *numagents* and the agent's *cpu*. The former is used as the magnitude of the organization's has-a relationship with  $A$ , and therefore cannot be deferred. The latter has no such restriction, and can be deferred to the optimization process. The overall organizational search will progress by producing a series of partial instances that can be passed to the optimization. An example of such an instance can be seen in Figure 4.9b. This was then translated into the set of Mathematica statements shown in Figure 4.10. In that code, the `constraints` and `variables` lists are used to contain their respective information. These are eventually passed into the `Maximize` function at the bottom, which performs the optimization process. The result of this process is a pair of bindings for the two *cpu* variables, which are then used to produce the final instance in Figure 4.9c. Agents were assigned *cpu* values of 30 and 20, which meets the high level constraint on organizational *cpu* while producing the minimum total cost of 25.

A depiction of the performance obtained when using Mathematica to maximize the value of deferred variables is shown in Figure 4.11. This graph was produced by comparing the performance using the numeric solver to the centralized search techniques described in previous sections across ten scenarios. Each scenario searched through the space defined by a variant of the template depicted in Figure 4.9a. The complexity of this search was increased in each subsequent scenario by changing the *numagents* variable, which increases the number of agents allowed. For example, in  $n$ th trial  $numagents = \langle 1, \dots, n \rangle$ . The organizational performance constraint was  $cpu \geq n \times 30$ . Each search sought to maximize the organization's *utility*, although

```

Clear["Global`*"]
variables = {}
constraints = {}

organization[agents] = {A1, A2}
organization[numagents] = Rationalize[2.0]
organization[cost] = (Total[Map[#cost] &, organization[agents]])
organization[work] = (Total[Map[#cpu] &, organization[agents]])
organization[utility] = (-organization[cost])
AppendTo[constraints, (organization[work] >= (Rationalize[60.0]))]
AppendTo[variables, A1[cpu]]
AppendTo[constraints, (A1[cpu] == (Rationalize[10.0]) || A1[cpu] ==
(Rationalize[20.0]) || A1[cpu] == (Rationalize[30.0]))]
A1[cost] = (Rationalize[10.0])
AppendTo[variables, A2[cpu]]
AppendTo[constraints, (A2[cpu] == (Rationalize[10.0]) || A2[cpu] ==
(Rationalize[20.0]) || A2[cpu] == (Rationalize[30.0]))]
A2[cost] = (Rationalize[10.0])

Maximize[organization[utility], constraints, variables]

```

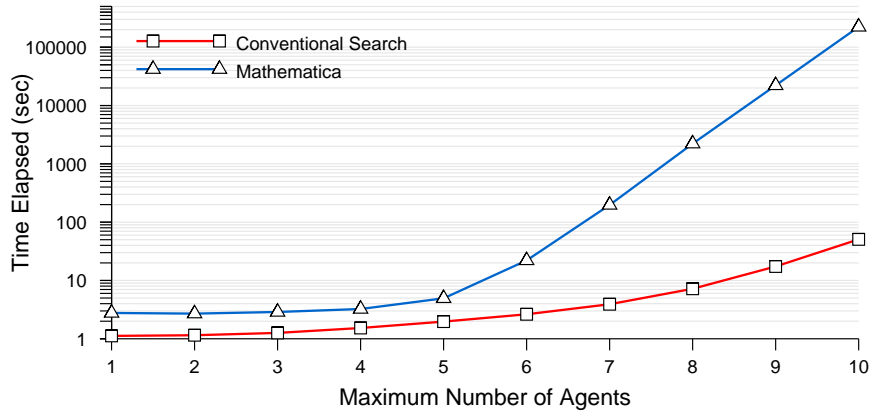
**Figure 4.10.** The result of the ODML translation process. This code is passed into Mathematica to perform the optimization.

the prior constraint caused just a single alternative in the template's space to be valid. All experiments were performed sequentially on the same processor.

As the graph shows, using the solver resulted in dramatically longer search times as the problem complexity grew (note the ordinate axis is logarithmic). At the most extreme point tested, the conventional search required 50 seconds to find a solution that took Mathematica 62 hours. My hypothesis for this behavior is that, because Mathematica's maximization function naturally operates over a continuous space, it is ill-suited to search through the discrete space presented by ODML. This may cause it to explore impossible variable assignments that the conventional search would ignore simply because they are not in the variable's domain, which would result in the longer running time. Other engines capable of attacking ODML's optimization problem in different ways may be more effective than what is presented here.

## 4.2.4 Avoiding Redundant Search

Somewhat related to the notion of equivalence classes is the higher-level pattern of paths explored by the search process. Most (but not all) has-a relationships with non-unary magnitude are unordered. The ordering of members within the set created by that relationship does not matter. During the search and incremental generation process outlined above, a depth-first approach is generally used to instantiate such relationships. The first node is instantiated, followed by the second, and so on. For each such instantiated node, a series of additional decisions might need to be made for additional organizational elements that exist in that child structure. For



**Figure 4.11.** Performance of the numeric solver versus conventional search across different problem complexities.

example, in both the distributed sensor network and information retrieval domains, has-a relationships exist in the structure that have potentially complicated decedent structures. The *sector* node in the DSN model can contain multiple *sensor* roles that must be bound to *agents*. The *aggregator* node in the IR model can contain a mixture of *aggregators* and *databases*, which also require agent bindings along with potentially more complex sub-structures.

Because the membership set may be unordered, a naive depth-first approach to this search will be inefficient through its repeated exploration of equivalent spaces. For example, consider a simple case where an *aggregator* has has-a relationship of size two with *source*. Both *aggregator* (A) and *database* (D) have an is-a relationship with *source*, so a complete search of the space might test the four choice sequences AA, AD, DA and DD. However, because the set is unordered, the results of AD and DA will be the same. There is no need to explore both options.

To avoid this, a list of explored sequences is maintained during the search process for each such decision point. As with the equivalence classes, a fingerprint is produced for each sequence, by storing the sorted sequence of decisions that have been made. During the search process, if the fingerprint from a completed sequence of decisions matches one that has been previously produced, backtracking occurs to avoid additional redundant work. This technique reduces the difficulty of such a search episode by treating the decision space as a combination of choices, rather than a permutation. The computational complexity remains the same, but significant savings in time and memory are still observed in practice by using such an approach.

## 4.2.5 Independent Sub-Problems

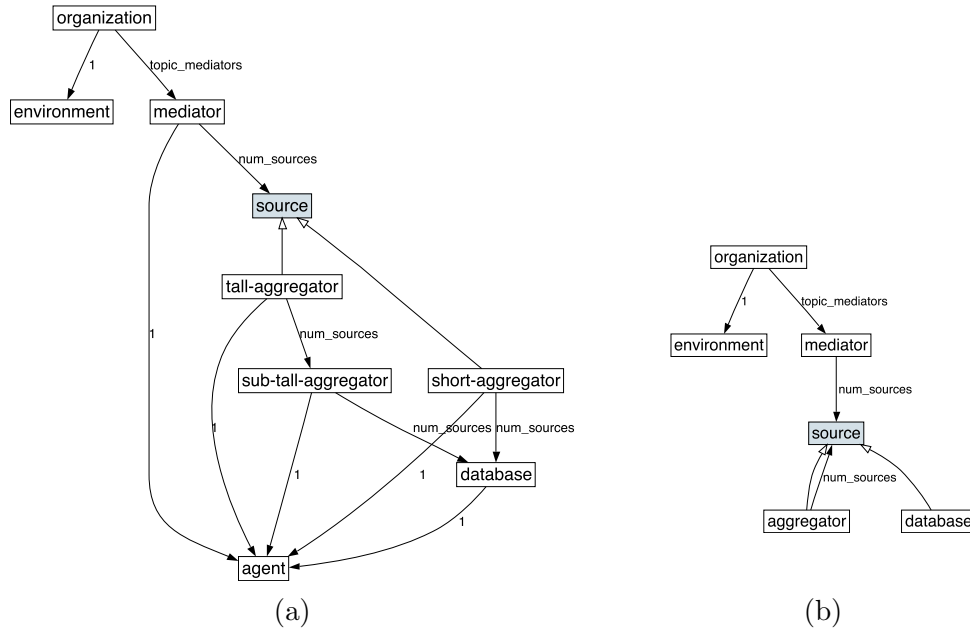
Another approach to optimizing the search process is to exploit the potential existence of independent sub-problems. There may be particular components which operate relatively independently of others, much like divisions in a human enterprise. These weak interrelationships can be detected by searching the template's data dependence graph for disconnected or weakly connected subgraphs. One could then find organizations that maximize or minimize the relevant characteristics of this subcomponent without regard to parts of the remaining organizational structure. Separating the two spaces and optimizing them independently can result in a much smaller search time without adverse effects to utility. This approach has been fruitful in other areas, giving rise to concepts such as nearly-decomposable hierarchies [172] and transition-independent decentralized MDPs [9] that use a divide-and-conquer approach to solving large and complex problems. Such techniques have yet to be developed for ODML, and remain an open area of research.

## 4.2.6 Domain-Specific Techniques

The approaches outlined above are all domain-independent. They have no knowledge of specific types of organizational components, or how they interact, other than what is encoded within the available model. This allows them to operate independently of what (potentially hidden) semantics might exist in or between those components. The obvious benefit is that these strategies work correctly regardless of domain or structure. The drawback is that the semantics they ignore may in fact be useful when guiding the search process. I showed how the semantics of the *sensors\_controlled* behavior could be uncovered by deducing the underlying trend of that field. However, it may be difficult or impossible to analytically deduce the trend of fields with more complex dependencies. A domain-specific search process, however, could know this intuitively, by embedding such knowledge into the search itself. Although it is quite possible to construct such a search process to work with ODML structures, and there may be good reason to do so for particularly complex and important models, in this thesis I will focus exclusively on more general, domain-independent approaches.

The one concession that is made in this direction by the current implementation is a feature of the language that allows the model designer to specify the order in which organizational choices are considered. Each decision point accepts a *decisions* tag, which takes in an ordered list of decision fingerprints. If such a list exists for a particular has-a relationship or variable, it is consulted and followed as that decision point is encountered during the search process. This permits some level of influence on the part of the model designer with a minimum of effort, particularly if it is known that certain areas of the search space are more fruitful than others.

A somewhat related feature that is also available is the ability to construct organizations interactively. As each decision point is encountered, a choice is solicited from the human operator. Of course, because of the vast number of possible organizations, this is not a practical way of constructing very large organizations or exploring the



**Figure 4.12.** Two information retrieval templates, derived from Figure 3.2a. a) Incorporates homogeneity, by limiting aggregator selection to two distinct choices. b) Incorporates abstraction, by eliminating the assignment of roles to distinct agents.

entire space of alternatives. It is, however, a useful tool when creating and evaluating the model itself, and when creating specific organizations for testing purposes.

### 4.3 Heuristic Modeling Techniques

Section 4.1.1 showed that the complexity of an ODML template is derived from the number of decisions that must be made when using the structure to generate organizations, which determines the number of candidate organizations that may be derived from that template. For example, when the number of agents that can be in the final organization increases, the number of agent-role assignments will usually increase accordingly, as may the size and number of the less tangible organizational structures such as hierarchies, teams, resource pools, etc.

In addition to creating ways to efficiently search the space as the previous section did, it is also possible to address the scale problem through changes to the model itself. By altering the template, one can limit the number of decisions that must be made when interpreting the template, thereby making the number of decisions less dependent on the number of agents in the system. This will reduce the number of candidate organizations, which will shrink the organizational space that must be searched. This section will concentrate on two modeling techniques, which have been employed to successfully reduce the complexity of the organizational search.

### 4.3.1 Homogeneity

Enforcing a certain amount of homogeneity, or at least similarity, within an organizational structure can dramatically reduce the number of decisions which must be made by eliminating organizational choices. For example, we might change our template such that all of a mediator's sources must have the same form, e.g., that they are all single level aggregators. We can improve on this strategy by exploiting ODML's inheritance rules to embed multiple distinct alternatives, rather than providing only a single choice. For example, consider the template in Figure 4.12a. In this model, I have defined two distinct source types, a *tall-aggregator* that has two levels, and a *short-aggregator* that has just one. Both contain a total of four databases, but they will have different performance characteristics because of their different structure. In this new template, candidate organizations may contain either or both alternatives, while other permutations of the aggregator hierarchy have been eliminated.

The use of homogeneity as an iterative process best exploited during the design phase. Typically, one would begin by creating a very general template, capable of producing almost all feasible organizations. As variations are generated and compared, it is common that particular organizational characteristics will define certain classes of structures or substructures. Simple examples of this include the "tall" and "short" varieties I have identified in the information retrieval domain. The members of a particular variety may be similar enough that a single representative structure can stand in for the entire class with only minimal loss of utility, in much the same way that members of a formal equivalence class do. For example, there are a vast number of short and wide aggregator hierarchies that have only minor differences in form and function. In Figure 4.12a I replaced this large number of choices with a single *short-aggregator*, which will certainly reduce the organizational search space, and hopefully not limit the quality of the final organization. Such classes can serve as the foundation for a reduction process that captures the notion of homogeneity, by replacing a potentially complicated set of decisions with a set of predefined structures. Ideally, one could incorporate a representative from each distinguished class, producing a template with a smaller candidate search set but negligible loss of potential utility.

### 4.3.2 Abstraction

A different way to reduce the decision complexity of a model is to use abstraction to reduce elements of the structure to their simplest form. Unnecessary or optional details may be removed or captured with a probabilistic representation to eliminate branches of the template which would otherwise add to the decision process. This strategy is used to design human organizations, such in supply chain optimization techniques that reason about entire companies, and the aggregate characteristics of those companies, not individuals within those companies. The ability to represent organizational elements at an arbitrary level of abstraction in ODML is significant feature that is absent most other existing representations, which typically require a complete structure down to the agent level. As with homogeneity, this practice can

potentially lead to an undesirable loss of expressivity in the model, but with care an appropriate compromise can usually be found. Critical details omitted from the model may also be restored to a subset of abstract candidates that have been found to be promising.

An example of this approach, particularly relevant to decomposition-based representations such as ODML, is to truncate the model at some point higher than the level actually used by the running system. This is already used in the models from Section 2.2 in some respects because the internal decision making processes of agents are not represented. A more typical example of this technique is to not model down to the level of assigning roles to individual entities or agents, as shown in Figure 4.12b. Organizations derived from a truncated template will specify what roles exist, and where they are located in the organizational structure, but leave them otherwise unbound. A separate, more detailed role-agent search could then be performed on a subset of the discovered structures, or a role assignment algorithm used to find an appropriate binding [159, 138, 35].

This technique is analogous to those presented by Durfee in [52], which used team-level abstraction to leave specific agent assignments unbound during coordination, also to reduce complexity. If agents were heterogeneous or permitted to take on multiple roles, this can reduce the search space exponentially. Even if agents were homogeneous, in a fully hierarchical structure this can cut the size of instances in half, which simplifies analysis and reduces memory consumption. The precision lost in this instance stems from the details that were previously stored within individual agent nodes. For example, it is more difficult to validate an individual agent's communication or work loads. Generic agent nodes can be retained to compensate for this loss of detail, but one will not be able to predict how the combined effects of multiple roles affect the agent or its performance within the organization.

The further implication of using this technique arises from the fact that the resulting organizational instance will no longer completely specify how it should be applied to a set of resources and agents. Decisions that were previously made during the design process must now be made by an axillary process or at runtime. In the example above, roles must be assigned to specific agents before the system can function. A second process must take the agent population and map them to the nodes proscribed by the selected organizational instance, which is itself a search process [175]. Although this late binding requires additional analysis after the design phase, our belief is that it also fosters increased context-sensitivity by providing a framework to support dynamic allocation. For example, assume that the *mediator* role not been bound to a particular agent at design time. At runtime, when the actual number and types of databases are known (as opposed to the statistical averages used in the models), the organizational design can be inspected to determine what resources that role requires and what burdens it will place on the agent it is assigned to. That entity model, coupled with the new information obtained at runtime can be used to select an appropriate agent to fill that role.

**Table 4.1.** Results from organizational search in small-scale information retrieval templates. Number of agents and utility are given for the optimal found organization.

Template	Decisions	Valid Organizations	Agents	Utility
Baseline	6,210,780,885	12341	9	692.86
Homogeneous	521,379,796	8273	9	691.45
Abstract	59,940	12	9	692.86
Homogeneous + Abstract (a)	5280	7	9	691.45
Homogeneous + Abstract (b)	3483	3	8	690.85

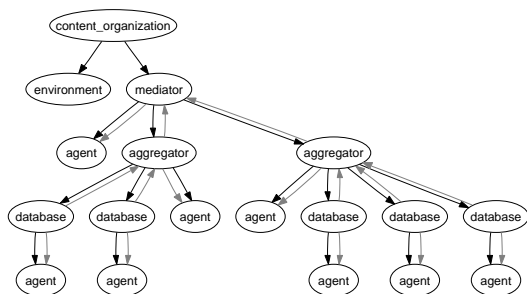
### 4.3.3 Heuristic Modeling Results

The exact amount of search space reduction that is observed using these techniques is dependent on the particular manner in which the template changes are carried out. Some approaches will clearly be better than others in terms of space complexity and achievable utility, and I have shown how hybrid strategies that use a limited set of decisions can help offset the drawbacks associated with model reduction.

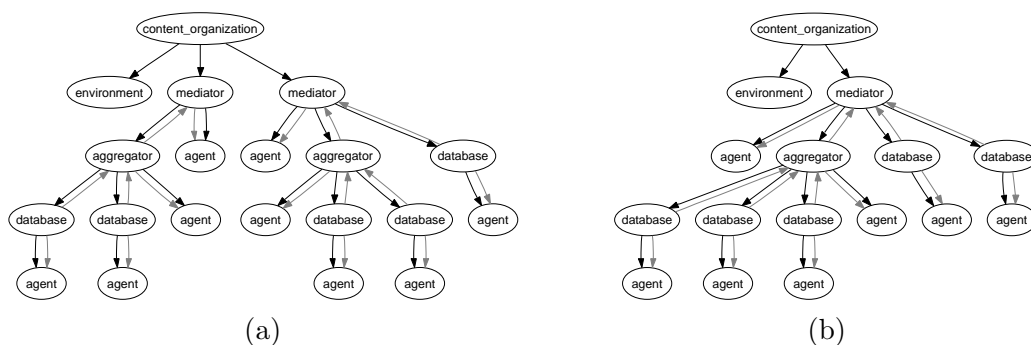
To demonstrate the effectiveness of these techniques, two sets of templates for the information retrieval domain have been created. The small set allowed up to five databases, up to two mediators, and the aggregators could have two, three or four sources. The larger design allowed up to 100 databases, with up to five mediators, each of which could have two to four sources. The number of agents was unbounded. The source node types are the same hierarchies discussed in Section 3.1, with a single level height restriction in the small scenario, and a three-level restriction in the large (i.e., up to two aggregators with a database leaf). Five templates were created for each scenario. The baseline template is shown in Figure 3.2a. The second template, using the homogeneous technique, is similar to that shown in Figure 4.12a. The third employs abstraction, by not assigning a particular agent to each role. Alternatively, one could also view this model as creating a new agent for each role. The fourth and fifth templates incorporate both the homogeneous and abstract techniques. Slightly different modeling changes were made in each, to demonstrate how the search space may be affected by these techniques.

The results from the small scenario are shown in Table 4.1. The table shows the number of decisions made during the entire search, the number of valid organizations that were found, and the number of agents and utility of the optimal structure. The most dramatic reductions in search space occurred using abstraction, which reduced the number of valid organizations that had to be evaluated by two orders of magnitude. The reduction in decisions that were made was even greater. The number of possible assignments of agents to roles can be quite large even for small organizations, so avoiding this process results in a tremendous reduction in search space. The optimal organization found by the baseline template is shown in Figure 4.13. Because there was no limit on the number of agents and no cost associated with each agent the optimal organization assigns only one role per agent.





**Figure 4.13.** The optimal organization found by the baseline template for the small-scale scenario.



**Figure 4.14.** Optimal organization instances produced by the Homogeneous + Abstract (a) and (b) templates for the small-scale scenario.

The organizations produced by the two Homogeneous + Abstract templates in the small scenario can be seen in Figure 4.14. Template (a) allowed single hierarchies with aggregators limited to two databases, while (b) allowed aggregators to manage three. Although both structures produce the same response recall, the optimal organization arising from template (b) had slightly lower utility. As seen in in Figure 4.14, the load in (a) is more distributed than it is in (b). This difference leads to a small increase in average response time, which resulted in the lower expected utility. This demonstrates the intuitive fact that different heuristic modifications will result in different search space modifications, and it is possible to lose high valued solutions in the process. The figure also shows that neither homogeneous template retained the optimally valued organization, because as Figure 4.13 shows the optimally valued organization itself was not homogeneous. This is a byproduct of the fact that an odd number of databases are used, and although the optimal design is lost, the amount of lost utility is quite small.

The optimal organization for three of the five large-scale database scenarios was simply too difficult to determine. Consider that, if agents may take on multiple roles, a single candidate structure containing 100 roles has  $100^{100}$  possible assignments of

**Table 4.2.** Results from organizational search in large-scale information retrieval templates. Number of agents and utility are given for the optimal found organization.

Template	Decisions	Valid Organizations	Agents	Utility
Baseline	Intractable, $\approx 7.4 \times 10^{301}$ candidates			
Homogeneous	Intractable, $\approx 2.2 \times 10^{108}$ candidates			
Abstract	Intractable, $\approx 7.4 \times 10^{201}$ candidates			
Homogeneous + Abstract (a)	52,792,143	473	135	14.561
Homogeneous + Abstract (b)	44,057,638	264,293	116	10.489

agents to those roles. Even if agents may take on only a single role, there may be  $100!$  permutations if the agents are distinguishable. Organizations produced from the structures used in this scenario may contain more than 100 roles, and there are billions of possible structures. Finding the optimal structure in such a large space is intractable. However, by incorporating the concepts of homogeneity and abstraction into the original model, it is possible to design valid organizations in a reasonable amount of time. A quantitative summary of the search process for the remaining structures which employ both the abstraction and homogeneity techniques is shown in Table 4.2. Lacking a baseline comparison, I cannot state that the optimal organizations that were found had the optimal utility originally achievable. However, many different organizations were found, all of which meet or exceed the constraints specified by the original model.

The particular organizations obtained by the two remaining templates are of minor significance here, more important is the fact that it was possible to use these techniques to find appropriate organizations for systems incorporating more than 100 agents. These results demonstrate how generic modeling techniques can be used to reduce the complexity of an organizational search process. If suitable modifications are made, one can vastly reduce the search space with minimal reduction to utility, although the amount of reduction is clearly affected by the skill of the expert making the changes.

## 4.4 Designing Organizations in Practice

This section covers a range of topics that relate to the practical application of the techniques described thus far. Section 4.4.1 describes the organizational instance evaluation process in more detail, including empirical tests of a value caching system used to make the process more efficient. Section 4.4.2 shows how and where the design process actually takes place. This includes both a description of the existing top-down, distributed approach and an outline of how ODML can be used to grow organizations in a more bottom-up, emergent fashion. Section 4.4.3 describes how a selected design is used in practice, using the information retrieval simulator from Section 3.2 as an example system that does so.

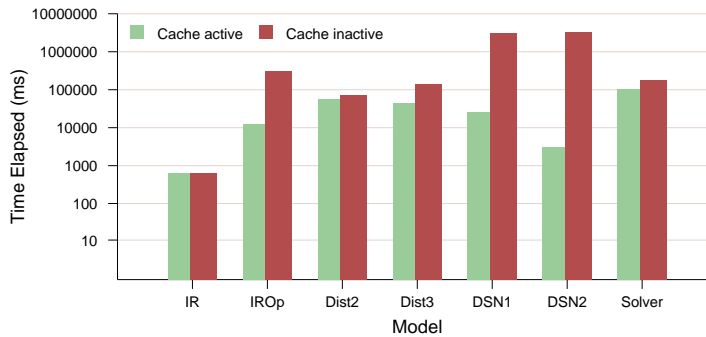
### 4.4.1 Efficiently Evaluating Organizations

Previous sections have shown several ways to search the organizational space, and different techniques may be employed within the search to decide what and how elements are considered. A common theme that runs through nearly all these approaches, however, is the need to analyze and evaluate the structure during its formation and after it has been completed. Much of this analysis is based on the ability to calculate the organizational values embedded within the node definitions, as formalized in Figure 2.9. For example, these are needed to determine the satisfaction of hard constraints, to determine the size of has-a relations and to calculate the quantitative characteristics of individual nodes, among other things. Given this ability, the comparative evaluation of organizational alternatives can be performed by first determining if the design is valid, and then ranking the designs based on their predicted utility.

Recall that an organizational value, such as *utility* in the *organization* node, is defined by a corresponding field that will contain an expression. When a concrete, numeric value for that field is needed, the process begins by determining what other fields (if any) the expression is dependent on. For example, in the DSN model, *utility* is dependent on the field *average\_rms*, also contained within *organization*. Before a value for the field may be determined, values for all its dependents must be calculated. This is done recursively for all the field's dependents, and the dependents of the dependents' fields, and so on. The recursion terminates when a field has no dependents. When values for all the dependents have been obtained, the expression can be evaluated. Note that this calculation may need to leave the original node and traverse the organization to obtain the dependent values. For example, *average\_rms* is dependent on *trackers.rms*, the *rms* field for each of the nodes contained in *organization's* *trackers* has-a relationship. Each *tracker* node must be visited to obtain this information.

A field's value is not determined solely by its defining expression, but also by any modifiers that have been applied to it. After the expression's value is calculated, one must also determine the values for any relevant modifiers and apply them. When a new modifier is created during the search and instantiation process, the node containing the target of that modifier is notified. This information is stored and referenced each time a value is needed, which permits the node to both recognize what modifiers exist and from where those modifiers' values may be obtained. After each modifier is applied in turn, the final value for the field is returned.

Because of the numerous interdependencies that can exist among fields, and the modifiers that can interject at any point, calculating a value can be a relatively time-consuming process. In addition, such calculations will usually be performed frequently during the search process, as the controller needs to evaluate the working state of organization to know when it is appropriate to backtrack. Because it is common for the same values to be requested a number of times in succession, a caching system has been added to the process to make it more efficient. When a value is requested, the computed value is stored and returned without further processing on subsequent queries. This can provide dramatic improvements, especially for highly interconnected fields such as *utility*.



**Figure 4.15.** Search performance with and without cached values.

The drawback to this approach is the relatively complex bookkeeping that must be maintained to ensure cache consistency. Values may change over the course of instantiation, as when a new modifier is applied or if a new element is added to a dependent array. When this occurs, the cached value must be invalidated, along with any other local or nonlocal fields that are dependent on that value, to be sure that up to date information is always used. Unfortunately, although it is possible to directly determine from an expression what it is dependent on, there is no direct way to determine what other fields are dependent on it. In order to accomplish this, a data dependency graph is maintained throughout the construction process. As new dependencies are recognized they are added, indicating what nodes and fields are dependent on a value. Later, if backtracking takes place, the appropriate dependencies must be removed and their corresponding values invalidated. Maintaining this representation does add overhead to the process, however in the majority of circumstances this overhead is far outweighed by the time saved by avoiding calculations. Section 4.5 will also show how this same structure can be exploited during the system’s monitoring phase.

A set of trials were performed on different models to evaluate the benefits of using cached values. The results are shown in the log-scale graph in Figure 4.15. The IR and IROp scenarios used the information retrieval model, returning the first valid and optimal instance, respectively. Dist2 and Dist3 employ a number of Monte Carlo evaluations. DSN1 and DSN2 are two different sensor network models, the first employs 36 sensors, while the second has only 4 and uses Monte Carlo test to vary the number of targets. The Solver is a test model that contains a number variables related through simple equations.

The three models that observed the most benefit were IROp, DSN1 and DSN2, each of which saw at least an order of magnitude reduction in time required to complete. The key difference between these and the remaining models is that they both contain a larger and more interconnected set of equations and have a long search path. The non-optimized IR model was interconnected, but the search was sufficiently brief that little recomputation was needed. The Solver model required a fair amount of

search, but the expression set was simple and small. As each decision was made almost every cached value was invalidated, so no benefit was realized. The Dist2 and Dist3 models were both simple and had essentially no search. All the time that was consumed during the Monte Carlo evaluation process. Like Solver, little benefit was observed because these models were small; nearly every value was invalidated as each new value was tested. The three models which saved time did so because large swaths of expressions that would otherwise need to be processed were effectively hidden behind cached values, and those values remained stable for sufficiently long periods that they could be used. Fortunately, it is precisely this type of large and interconnected model that is frequently required to describe realistic systems, suggesting that similar performance gains can be observed in practice.

## 4.4.2 Constructing Organizations

The design process as outlined thus far has been described without specifying in detail where the actual computation and deliberation takes place. In this section I will provide additional details that concern how and where the design search takes place. Section 4.4.2.1 will briefly describe the initial centralized, top-down approach, as well as the current distributed strategy that retains a top-down perspective. Section 4.4.2.2 will outline how a distributed, bottom-up approach can also be used to grow the organization in a more emergent fashion.

### 4.4.2.1 Top-Down Construction

Top-down construction of organizations from an ODML structure proceeds in a manner consistent with the implicitly centralized design presented thus far. A model is created and given to a single process which searches for an appropriate organization, optimizing the exploration by using some of the search and space reduction techniques mentioned in the previous two sections. The selected organization is then provided to the relevant entities (e.g., agents), which use the information to select roles, identify relationships and guide local activity.

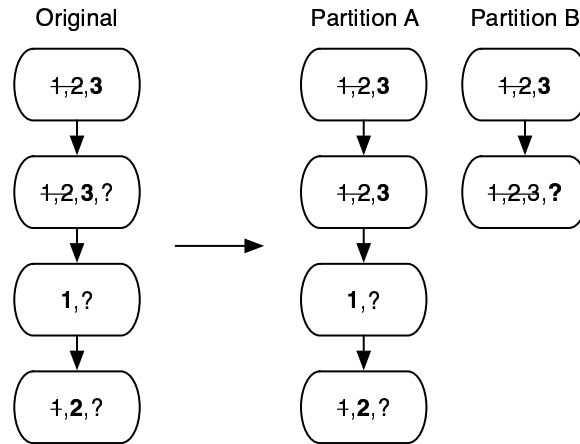
This is known as a top-down approach because a complete, unified view of the organization is decided upon at once through the efforts of one or more closely cooperating entities. The existing search implementation can be distributed across multiple agents or processors to take advantage of parallelism, or operate in a centralized manner on a single processor as a degenerate case of the same design. This distribution is facilitated by the naturally decomposable organizational space. Section 4.1 describes how the design process can be thought of as a series of choices made for the decision points encoded in the organizational template. These decision points form the backbone of a corresponding decision tree, while a series of choices that form a particular organizational instance is a path from root to leaf through that tree. The search space can be cleanly partitioned at each vertex, where the decided path to that vertex is shared and the child choices may be divided as needed. Assuming that the individuals performing the search can be provided relatively equal portions of

that space without excessive communication overhead, the total search time can be dramatically reduced.

The challenge in this design is dividing the search space such that processing nodes are evenly loaded and no redundant work is performed. There are at least two ways to create such a division. The direct approach is to analyze the space and divide it into  $n$  roughly equal-sized parts for the  $n$  available processors. If done correctly, this can maximize parallelism while minimizing inter-processor communication. The difficulty in this approach is that it requires one to characterize the entire search space before a division is made, and it assumes that a series of equal divisions can be devised from such a characterization. Because parts of the organizational space may only become apparent as a result of other choices, it is not trivial to create such a characterization for an ODML template in the general case. A further complication arises in a mixed processor environment, where an “equal division” may have to take into account the capability of the target in addition to the size or complexity of the partitioned space that is provided.

Because of these complications, I have employed an alternative division technique that partitions the space dynamically as the search progresses, similar to that presented in [146]. This trades off optimality in message exchange to create a solution which dynamically adapts the division of labor, regardless of the search space characterization or participant heterogeneity. Assume there exists a set of  $n$  processing agents that will take part in the search. Upon initialization, all agents provided with 1) the organizational template, and 2) the names of their agent “neighbors”, some subset of the  $n$  agents. The search begins when a single agent is told to begin searching. Whenever any agent has no organizational space to search, it sequentially asks each of its neighbors for more work (i.e., another part of the organizational space). If the recipient of such a message has extra work, it partitions its local space and gives the new fragment to the requester. The requester then stops its querying and begins searching the new space. This continues until either a valid organization has been found (satisfaction) or the entire space has been searched (optimization).

The efficiency of this process is determined in large part by how the local space is partitioned in response to a request. If either side of the partition is very small, the time spent searching for and creating that small partition may be relatively large compared to the time saved by the parallel computation. Because the query recipient does not know to expect subsequent requests for work, an ideal optimistic partition would divide the local space in half. Unfortunately, accomplishing this directly requires one to address the same space characterization problem described above. Instead, the algorithm approximates that characterization with a heuristic that divides the space at the highest vertex in the decision tree that has remaining choices. In large, relatively balanced spaces, splitting at the highest point should do a good job of bisecting the space. Even in less balanced spaces, the participants will eventually converge on the larger areas, as new work is sought out after the smaller spaces are consumed. An example of the partitioning process is shown in Figure 4.16. In that figure, the numbers which are struck out represent choices that have already been explored. Those in bold represent the current decision path under evaluation, while a question mark indicates that additional unknown choices remain. The division occurs at the second



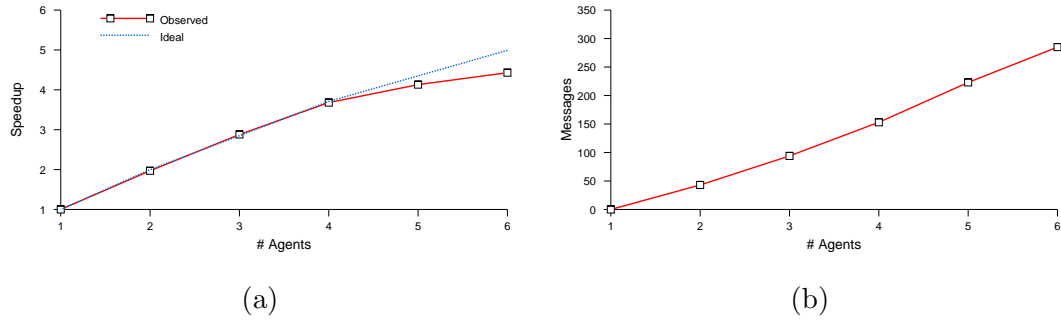
**Figure 4.16.** The partitioning of a local search tree. Strikeouts indicate visited choices, bolded are the current choice, while a question mark indicates additional unknown choices remain.

node from the top. Partition A encompasses the part of the search tree that follows below choice (3, 3), while partition B contains the remaining choices (3, ?).

After a valid instance is found, the decision tree and utility (if any) of that solution is broadcast to all processing agents. If a satisfaction search is being performed, the agents will each cease their search upon receipt of such a message. If an optimization search is being performed, the search will progress, but each node will keep track of the highest utility instance found thus far. The utility of each subsequent valid instance found by an agent is compared against the current best, and broadcast notification takes place only when a higher utility design is discovered. The distributed search terminates when all searching agents become quiescent, at which time each agent will have a local description of the valid or optimal instance that was found.

The performance of the distributed search using different numbers of processing agents is shown in Figure 4.17. These were produced from a series of optimization searches on a template similar to that shown in Figure 4.9a, where the organization could have up to 10 agents, each with 5 different *cpu* choices {10, 20, 30, 40, 50}, and a minimum aggregate *cpu* of 500. This produced an organizational space containing approximately 12 million alternative designs.

Figure 4.17a shows the amount of speedup that was obtained, where an observed value of  $n$  indicates the trial completed in  $\frac{1}{n}$ th the time of the centralized solution. Both the observed and ideal performance profiles are shown. If the underlying processors used by the agents were uniform, the ideal speedup would be linear, achieving a  $n$  times improvement if  $n$  In these experiments the processor pool was not uniform, and therefore the ideal is weighted based on the measured performance of each *cpu* as it is added. The “bogomips” metric reported by the Linux kernel was used to gauge



**Figure 4.17.** Characteristics of the distributed search using different numbers of processing agents. The speedup factor is shown in (a), and the number of messages required in (b).

performance [207]. The six processors were added in order of decreasing performance, with ratings of 5583, 5583, 4771, 4771, 3578, and 3578, respectively.

As can be seen, the distributed algorithm performs well with respect to the ideal for this number of agents, from which one can infer that the distribution process is efficient and agents are spending the majority of their time searching the organizational space. The number of messages required to achieve these results is shown in Figure 4.17b, which indicates a roughly linear increase in messaging as the number of processing agents increases.

I would not expect these trends to be maintained indefinitely, as the speedup must inevitably fall off when there is insufficient search space available to create large partitions. The time and number of messages will increase with the population size for the same reason, as well as because there are simply more agents to search to find work. This problem presents interesting parallels with the simple sensor network problem described in Section 1.1. It seems likely that a more complex organization, such as a centralized work allocation manager or a hierarchical work dissemination tree could help address the challenges that arise as the agent pool scales.

The benefit that this approach offers over those in Sections 4.2 and 4.3 is that it works on all models and requires no modifications to a model to be used. The drawback is that it clearly requires additional physical resources, and the amount of achievable speedup is lower. For example, a roughly linear improvement was observed with the distributed approach, while the search and modeling techniques described earlier produced results that were orders of magnitude better in some cases. These techniques are not mutually exclusive, and in practice the search and modeling techniques are used without modification by the individual participants in the distributed search to further improve performance.



#### 4.4.2.2 Bottom-Up Construction

Unlike top-down construction, an organization developed in a bottom-up fashion is not typically produced in its entirety by a single entity, or by a group of entities that explicitly divide the search space in the manner described above. In this design, several entities will work in parallel on different aspects of the organization. Information may be shared between the workers, but typically no single entity will have a complete, current view of the structure at all times. Instead, the completed structure will emerge piecemeal, and individual components may be changed or adapted as the larger picture takes shape.

The bottom-up strategy I suggest is actually closer to a compromise between a pure, so-called emergent approach and the more directed top-down approach outlined above. In doing so, I believe one can exploit the benefits of both techniques. Bottom-up approaches are typically more dynamic and distributed, but the ultimate product of such a process can exhibit pervasive deficiencies because of the myopic nature of the construction process. The top-down approach is more informed, and thus is less likely to produce structures that fail to achieve high-level objectives while appearing satisfactory from a local viewpoint. By providing agents employing an emergent paradigm with both a modest description of the state of surrounding agents and a model capable of incorporating that data, the local construction process may be guided towards solutions that are more globally coherent.

ODML, with its representation of both local and global behaviors, can serve in this role. Organization might begin by seeding the agent population with a complete or partial ODML structure. Simple clusters or coalitions of agents will begin to emerge as agents become aware of their neighbors. Self-organization would then progress as organizational components that require creation are identified, and the best way to satisfy that need given the agents and resources at hand is determined. For efficiency purposes, a partially centralized approach seems appropriate, where one or more distinguished individuals in those clusters direct the organization process.

In addition to describing the selection of possible organizational components, the ODML structure can also be used to determine what local information might be needed non-locally, or vice-versa. For example, in the IR domain, the structure of a mediator hierarchy can depend on what other entities already exist in the environment. Obtaining and disseminating that information is crucial to creating a globally coherent solution from an inherently local perspective. This can be determined from the ODML structure by analyzing the data dependencies between nodes, and recognizing those features that are needed across boundaries in the emerging organization.

Specific techniques for achieving this vision are a matter of future research.

#### 4.4.3 Applying Designs to Actual Systems

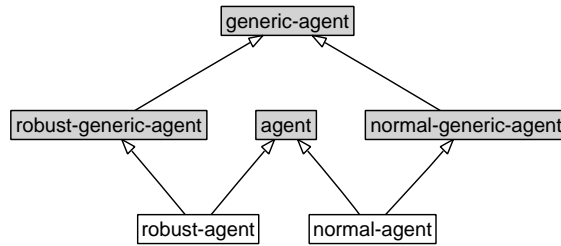
Once the search process has completed successfully, a particular design will be available that must then be applied to a running system, assuming it has not already been gradually integrated in an emergent fashion as described above. ODML's very free-form nature precludes a simple, straightforward mapping from design to system

for an arbitrary model. For example, parts of the model may clearly correspond to tangible artifacts that can be directed, such as roles or agents. Other parts may be more ephemeral, or be there solely to model an environmental response that needs no instruction. Yet another class of information may not correspond to any specific process, despite the fact that it contains details that must be correctly incorporated for the organization to function. The *sector* node from the sensor network model is such a case, because although it is only a logical construct it still contains vital information, such as the size and membership of the set of sensors belonging to it.

Because of this, the translation from design to running system is a model-specific process. At one extreme, a designer can create a model where each node corresponds to a specific, real entity that can make direct use of the information stored in the node. In this case, the node description can simply be used like a normal configuration file. At the other extreme, the model may be just a superficial approximation of the system in question, where individual nodes have no direct connection with any entity that will exist in the real system. In this case the design would be less of a blueprint and more a set of guidelines from which an engineer could derive insights when building the system.

In practice, the construction process usually falls somewhere between these two cases, where some details may be directly used and others require more effort on the part of the designer or running agent to gain access to. For example, the simulation environment created to evaluate the information retrieval domain (Section 3.2) takes an ODML instance model as input. From this, a bootstrap process determines how many agents will exist in the system. Once created, individual agents can first determine the set of roles they are expected to take on, by obtaining the appropriate *agent* node and finding the *role* nodes that have a has-a relationship with it. These roles can then be created and bound to the agent. Each role is responsible for inspecting its counterpart in the instance model to obtain any role-specific parameters. For example, the *mediator* will read the values selected for the *search\_set\_size* and *query\_set\_size* variables that control the mediator search process. Finally, each role also determines the set of other roles that it should be interacting with and how those interactions should take place. In this case, *mediator* and *aggregator* discover the set of nodes below them from their has-a relationships, and their presence in the *sources* field indicates they should be used as information sources. Other information present in the model, such as expected query rates and the performance metrics originally used to estimate the design's utility, are also available and can be used to evaluate runtime behaviors. Section 4.5 will show how these can be particularly useful when the system or environment is subject to change.

The end result of this inspection process is a system that can take an arbitrary information retrieval instance model as input and create a running system from it. The code required to do this, however, is specific to the particular model in question. The model does not indicate exactly what an *agent* or *mediator* should do on a moment-by-moment basis. Nor does it define the communication protocols or techniques needed to search through a local database. Such details are typically much too fine grained to be practically incorporated into a model. If these elements are crucial to organizational performance then they will be appropriately modeled with



**Figure 4.18.** Reusing common agent models in new domains.

an expression and the relevant characteristics used to parameterize the running code, but the model does not provide all the implementation details necessary to enact those behaviors in practice.

#### 4.4.4 Model Inheritance and Reuse

There are frequently design elements that appear again and again in different organizations, particularly among organizations intended to operate in the similar environments or make use of the same resources. For example, both the DSN and IR domains had agents, roles, and an environment. It is possible to reuse and recycle these elements, thereby leveraging existing work and simplifying the modeling process. This reuse is enabled through the use of generic designs, whose components can be incorporated with is-a relationships and specialized as needed.

Recall that the is-a relationship allows object-oriented style inheritance relationships to be defined in an ODML model. In both the distributed sensor network and information retrieval models this capability was used to define characteristics shared by multiple entities inside a common base node. These characteristics could then be imparted on those entities through the is-a relationship, which reduced the size of the model and the time required to create it. The ability to express inheritance relationships in ODML can also allow the time, effort and expertise needed to create organizational components to be exploited in new circumstances. This is accomplished by first creating a domain-independent set of nodes, capturing characteristics that exist regardless of context or application, and then use inheritance to incorporate those nodes in different models across different domains.

For example, in the DSN environment there was an *agent* node. The notion of an agent is quite general, and likely to be used in most models of agent systems. Instead of creating such a node anew in every model, one could create a *generic-agent* just once, that had common attributes such as *communication\_load*, *computational\_load* or *cost*. Generic, related variants such as *robust-generic-agent* and *normal-generic-agent* could also be created that possessed more specialized characteristics and constraints. An example of this is captured by the structure fragment shown in Figure 4.18. The remainder of that structure shows how those generic nodes can be used in a domain-specific manner. In this case, the *agent*, *normal-agent* and *robust-agent* nodes from the DSN domain (see Figure 2.10a) have been added. By simply adding an is-a

relationship from, for example, *normal-agent* to *normal-generic-agent*, the domain-specific agent node inherits all the information present in the generic agent node. Data which is relevant to the new domain can then be reused, and that which is not can be overridden.

## 4.5 Adapting Organizations

While a designed organization may be suitable for the particular context it was envisioned for, it is inevitable that long-lived, real-world systems will encounter new and different environments during operation. One strategy to address this is to model and design for the full range of possible conditions. The strategies presented thus far are sufficient to accomplish this. This is a reasonable approach so long as 1) the predicted range of environments is accurate, and 2) any and all compromises made in the resulting system are tenable. It is frequently the case, however, that one of these two assumptions will not hold.

In such situations, the natural, but potentially complicated approach is to dynamically adapt to the changing conditions. Organizational adaption is one approach to accomplish this. This section will outline one approach by which ODML may be used facilitate this process.

An important feature of the adaptation process, separate from the organizational search itself, is recognizing the cost of making suggested changes. This includes up-front costs, such as communicating organizational change or migrating existing tasks, as well as more indirect costs, such as the reduced quality or increased uncertainty the system may exhibit during transition. Although I will show that ODML can model the range of possible solutions, it is currently unable to calculate the specific costs associated with implementing these solutions. To do so, a process would need to compare the current design with a proposed one, and determine a strategy for performing the migration. This cost could then be incorporated into a structure's utility value, after which candidates could be compared as usual. Because ODML only models organizational characteristics, and does not have implementation-specific knowledge about how a structure is applied to the participants, it cannot derive these costs itself. However, it is certainly possible for a separate component possessing this knowledge to do so.

### 4.5.1 Monitoring for Problems

The first step in the adaptive process is to monitor for problems. These can be both potentially critical exceptions, such as node failures or a hard constraint violations, or inefficiencies, which can generally be thought of as soft constraint violations. It has been argued repeatedly in this work that ODML's predictive capacity allows it to design contextually appropriate organizations. These same predictions can be used as a baseline performance metric, by comparing the value characteristics of the ODML model against runtime observations to recognize problems. This section will outline how a distributed monitoring regimen can be put in place by disseminating

fragments of the ODML model among the agent population, and using them to guide local monitoring and notification processes.

Earlier work by myself and Benyo [78] used a related technique to detect failures. That system employed what is known as a *causal model* to discover and isolate failures. A causal model can be thought of as an operational decision tree that can be fed observations to produce diagnoses. Each node in the tree represents a hypothesis or sub-hypothesis, capable of diagnosing a certain class of problem. A positive detection by a node triggers its children, and by these steps is a more specific diagnosis generated and tested.

The strength in the causal model approach is its ability to capture almost any kind of behavior. Because the linking of nodes within the model and the internal processes used by the nodes to create diagnoses are arbitrary, nearly any failure that has an operational signature can be detected. The drawback to this approach is that such models can be very labor-intensive to create, and they are inherently limited to addressing the faults they were designed to detect.

I propose using a technique more closely related to the family of *model-based diagnosis* methods to serve these monitoring needs. Model-based diagnosis makes use of a computational model of a system that is run in parallel with the actual system. Deviations in the model's output from the observed behaviors can indicate a failure. Therefore, in place of a problem-specific causal model, one can use the original ODML model to serve a similar purpose. This strategy cannot detect the range of faults that a causal model can, but it can operate on any features that were important enough to include in the organizational model. Perhaps more importantly, it makes use of an already existing structure, thus avoiding the effort needed to make a comprehensive causal model and achieving a broad monitoring framework with relatively little additional effort.

Each constant field in the ODML model represents a potential characteristic that might be observed at runtime. By using the model to extrapolate from those characteristics that are monitored to the effects that could take place in characteristics that are not, a more complete view of the system's performance can be ascertained. If agents make the simplifying assumption that expressions in the model are always correct, any observed deviations should be attributable to one or more numeric parameters (e.g., constant-valued expressions) that were initially provided to the model. These can then be subject to more intense scrutiny, or (if possible) their actual values derived from the observed dependent characteristics and the expressions that relate them.

The quantitative effects of higher-level organizational structure, such as nodes and has-a relationships, are also assumed to be appropriately captured by the organizational model. In other words, if the presence of a particular agent is relevant to the utility obtained by the organization, the expression that predicts utility must correctly integrate this fact. Therefore, structural failures such as an agent crashing or a communication link going down will have an observable quantitative effect, and can be detected by the value-centric approach outlined here.

ODML-based monitoring can be accomplished in either a centralized or distributed fashion. In a centralized scheme, the relevant observations are delivered to a single

monitor, which can then use them to directly evaluate the fitness of the existing organization. A more scalable approach distributes this responsibility, where fragments of the ODML model are disseminated among a set of monitors. Nonlocal characteristics, and local characteristics which are not monitored, can rely on the values predicted by the original model when no other information is available. Ideally, the fragments are distributed in a way that maximizes the amount of locally produced data that can be compared to predictions made by the available fragments, thereby reducing the communication overhead need to support the global monitoring process.

For example, the concept of roles has appeared repeatedly in the example organizational designs. A natural mapping would have each agent responsible for monitoring the characteristics of the roles it is assigned, as well as those of the agent node itself. A different approach could address the need explicitly in the design, by creating an explicit *monitor* role with particular monitoring responsibilities that could be assigned to an agent like any other. However, because ODML nodes have no inherent semantics, this is ultimately a domain-specific process that is left to the system designer. The specific characteristics that are measured, and the frequency at which this is accomplished, are also left to the designer.

An unanswered question is, given that a deviation is observed among the monitors, who or what should be notified? The distributed scheme assumes that no single entity has a complete view of the working system. Each will have a particular local context that is observable, but it can not necessarily observe all the characteristics that can directly or indirectly affect its performance. Similarly, each monitor does not necessarily know how the values it observes or deduces locally affect others. Fortunately, the original organizational model itself can provide guidance. Recall from Section 4.4.2.1 the problem of identifying fields in remote nodes that were dependent on local values when invalidating cached data. The problem faced in this situation is the same – we wish to know what characteristics of the organization might be affected by locally observed changes. If those affected characteristics and their owners are known, then that information could be used to correctly route observed deviations. The data dependency graph produced during organizational creation can be used to obtain this knowledge, so by incorporating it into the distributed ODML fragments the routing problem may be addressed. A new observation can be sent to affected nodes, which can use it to recalculate local fields, potentially resulting in a new set of deviations. These deviation notifications can then be routed from one node to the next, following the relationships embedded in the expressions, until all relevant parties have been updated. By comparing these updated values to hard and soft constraints, faults may be identified.

### 4.5.2 Searching for Solutions

After deviations are observed and collected at a location capable of analyzing them, they can be used as parameters to create a new model that incorporates the current working context. This can then be used as a basis for a search for an appropriate alternate configuration. The primary difference between the techniques needed by this search and those presented in Section 4 is the real-time nature of the problem.

Where an initial design problem can be considered an offline or start-up time problem with fairly relaxed time constraints, the adaptation or reconfiguration problem occurs while the system is running. The longer it takes to find an appropriate solution, the greater the chance is that it will lose relevancy because of additional changes that occur during the search.

Because of this, although the search techniques presented earlier can be used as part of the adaptation process, they will likely not be the complete solution. In the case of critical failures, increasingly heuristic and anytime strategies are needed to aggressively search for a new, viable organization. With less critical problems, more incremental solutions might be employed, allowing for a more thorough search of the organizational space. Knowing what type of failure is being addressed, and what effect the continued existence of the organization in question will have on utility, can guide the choice of strategy.

Alternatively, one could attempt to learn or anticipate common failures offline, and develop contingency plans to address them. These plans could take the form of a change policy, or a predefined set of candidate organizations known to function under different conditions. For example, one could place additional restrictions on the original model, such as stricter communication constraints or an artificial limit on the number of available agents. By finding suitable organizations for these model variants one can create contingency organizations that can be deployed when similar or equivalent circumstances occur online.

Whatever the strategy employed, it is clear that devising a suitably quick and reliable adaptation approach is a challenging problem. This will remain an interesting area of research as structured organizations are used in dynamic real-time environments.

## 4.6 Conclusions

This chapter began by describing the organizational design problem ODML is addressing, and showed in Section 4.1.1 that this is a difficult problem to solve in general. Because of this, I have explored a range of techniques, each of which has different strengths and weaknesses. These are currently unified in a single implementation that attempts to use each technique where applicable, and is conservative where they are not. Because these approaches make no domain-specific assumptions, they demonstrate that it is possible to create effective algorithms that address the general design problem.

Although none of the techniques remove the fundamental complexity issues that arise as the general problem scales (notice that, for example, the trends in Figures 4.8 and 4.11 that use the techniques are still exponential, but at a slower rate), they do allow classes of problems that would otherwise be intractable to be solved. Of course, this is by no means an exhaustive study of such general approaches, and I believe that in many cases a suitably crafted domain-specific approach can yield additional benefits. Chapter 6 will describe in more detail how other researchers have addressed

similar problems, and the tradeoffs that are made compared to the ODML-based techniques this dissertation presents.



## CHAPTER 5

# MODELING OTHER CHARACTERISTICS AND PARADIGMS

In this chapter, I provide additional examples of ODML models, to both demonstrate that ODML is capable of capturing a range of important concepts and to provide guidance to those that want to use ODML to model these or similar situations. A number of detailed modeling examples have been shown in Sections 2.3 and 3.1. This chapter takes a broader perspective by outlining how ODML has been used to model a range of additional features and organizational paradigms. The discussion focuses more on the important high-level concepts and less on minutia, so that a sufficiently wide range of topics can be addressed. In doing so I hope to convey what concepts ODML can be used for, how well it serves in those roles, and what compromises (if any) are made.

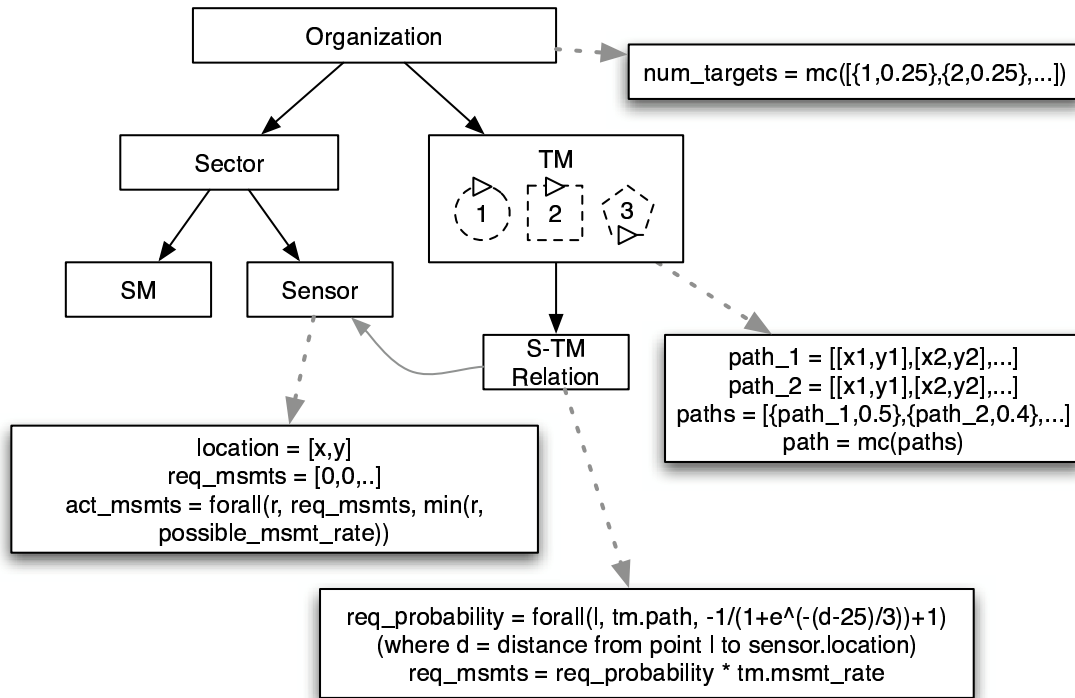
Section 5.1 touches on topics from previous chapters by showing how additional organizational characteristics that were either omitted or highly abstracted in the original DSN model may be integrated. In particular, it shows how geographic heterogeneity can be represented, how temporal interactions can be captured, and how models with different levels of abstraction may be produced.

Section 5.2 explores ODML's flexibility from a higher-level perspective, by describing how models can be defined and used to represent a number of organizational paradigms commonly used in multi-agent systems. These include hierarchies, coalitions, federations, marketplaces and teams, among others. For each paradigm I outline features that represent important design considerations and show how they may be incorporated into an ODML model.

## 5.1 Modeling Common Organizational Characteristics

### 5.1.1 Non-Uniformity

Section 2.3.9 evaluates how accurately the distributed sensor network model predicted the characteristics of the implemented system. For the most part, the predictions made from that model were satisfactory, although there were discernible deviations that could be attributed to simplifying assumptions made within the model. One of those assumptions was that both the sensors and targets in the environment were uniformly distributed, and another was that the set of sensors was used uniformly

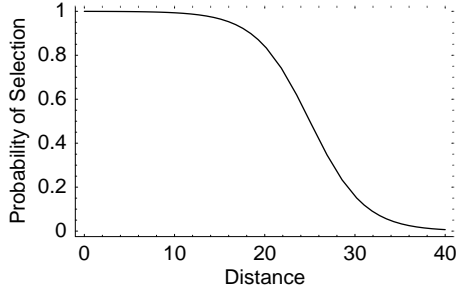


**Figure 5.1.** A depiction of the revised DSN template. Important modifications from the original DSN model are shown in the shadowed boxes.

by the existing track managers. Although both the sensors and targets were in fact uniformly distributed in the empirical tests, this does not necessarily result in the measurement burden being evenly distributed across the sensor population. For example, a sensor in the middle of the field will be used more than its counterpart on the periphery, simply because its central location allows it to sense more targets, causing it to be called upon more frequently by the managers responsible for those targets. This intuitive behavior is not captured by the existing model, which resulted in a consistently underestimated communication deviation (see Figure 2.17b).

Additional information can be added to the model to address this deficiency, as shown abstractly in Figure 5.1. The figure shows the skeleton of the DSN template, along with brief summaries of the most important changes that were made (the expressions in shadowed boxes). The changes revolve largely around the addition of location information to the relevant nodes, along with the information flow modifications needed to support the new information.

I will start at the beginning of the measurement demand chain, at the track manager (TM in Figure 5.1). In the original model, the track manager’s demand was expressed as the single value *desired\_sensors*. This demand was allocated evenly through the sensor population, implicitly specifying the target had an equal chance of being anywhere in the environment at any time. The new model abandons this in favor of a more concrete notion of the target’s location, which takes the form of a



**Figure 5.2.** The sigmoid function used to calculate sensor demand in the DSN model. This reflects the sensor’s range of approximately 30 feet.

*path*. A path is defined using ODML’s syntax as a list of points, where each point is represented by another two-element list. Although the objective behind this revision is to add important detail to the model, in making the decision to model target location as a single path there is the danger of creating a model that is *too* specific. Predictions made from such a model would likely be scenario-specific, limiting their usefulness when evaluating a design in the general case.

To address this, the model defines several different paths, which are defined as *path\_1*, *path\_2*, and so on (an arbitrary number can be defined). These are combined into a discrete distribution constant named *paths* that specifies the probability of each path. The model requires that a single path be selected for evaluation purposes. The *path* constant does so by using the *mc* function to select one at random. In the *organization* node, the *num\_targets* constant is also defined using a sampled distribution. During Monte Carlo analysis (see Section 2.2.3), different numbers of targets and individual targets’ paths will be selected, allowing the design to be evaluated in the context of many different problem configurations, and ranked accordingly. This random path model is consistent with the environment the original system was designed for, and other mobility models exist that could also be used [20].

As before, a set of sensor-track manager relation nodes (S-TM, one for each sensor) is used to impart measurement demands made by the track manager to individual sensors. In the new model, this is done by using the track manager’s *path* to determine the probability a sensor will be used to track the target in question at each point specified by the path. This depends in part on the proximity of the sensor to the target. To make this determination, each sensor is given a *location* constant, a two-element list that is determined by the origin of the sector it is contained by along with its relative position within that sector. For each point in the target’s path, the Pythagorean Theorem is used to determine the distance between that point and the sensor’s location. An offset sigmoid function

$$f(d) = \frac{-1}{1 + e^{-(d-25)/3}} + 1$$

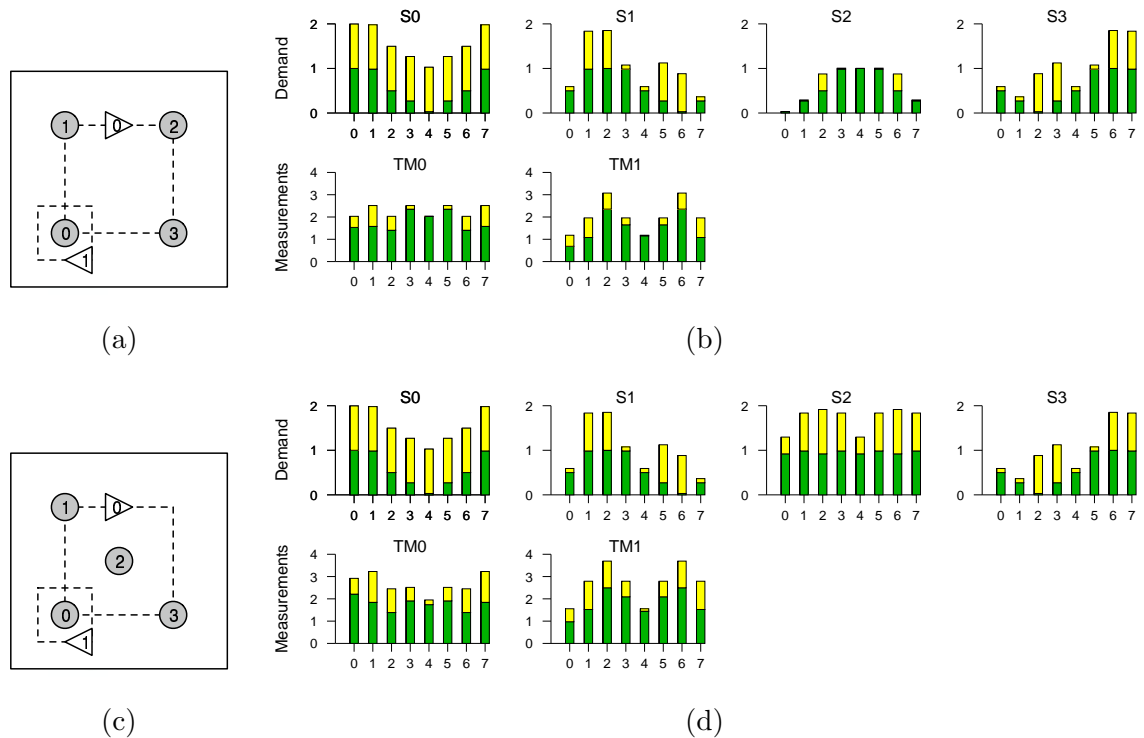
(see Figure 5.2) is then used to determine the probability the track manager will select the sensor based on the computed distance. Unfortunately, this calculation

only accounts for the effect of the sensor’s proximity. It does not take into account sensor density, which would adjust the probability of selection for any one sensor based on the number and quality of alternative sensors the manager could choose from. Thus, this remains only an approximation of the system’s true behavior.

After computing the demand probability at each point, the actual rate of *requested\_measurements* is computed by multiplying that list by the track manager’s measurement request rate. Similar to the original model, this new list is passed to the sensor with a modifier, where it is added to the sensor’s local *requested\_measurements* list (initially zero). After all remaining track manager nodes have been created, this will reflect the total, aggregate demand place on the sensor by all targets in the environment. The sensor’s *actual\_measurements* are computed from the total demand by comparing each aggregate request point with the sensor’s possible measurement rate. This is used by the sensor-track manager relation to determine the rate of measurements the track manager can expect at each point, which is relayed back to the track manager. This is done in the same manner as the earlier model, except a list of rates is maintained rather than a single value. The aggregate number of received measurements will accumulate in the track manager’s *actual\_measurements* list after all sensor-track manager relations have been created, at which point an average RMS value can be calculated as before.

To demonstrate how non-uniformity is captured by this model, the predictions for a pair of simple 4-sensor, 2-target scenarios are shown in Figure 5.3. Sensors are shown as circles, targets as triangles. Each triangle’s orientation matches the transit orientation of the corresponding target, and the dashed lines represent target’s path. For example, Figure 5.3a has four sensors arranged in a grid, while target 0 performs a large square circuit and target 1 a smaller square circuit. Path 0 has eight equidistant points while path 1 just four points at the corners, so target 1 will travel two cycles for each of target 0.

The sensor demand and target measurement predictions for this scenario are shown in Figure 5.3b. The top set of graphs shows the aggregate demand placed on each sensor at each point on the path, where the darker portion is target 0 and the lighter is target 1. The horizontal axis refers loosely to successive steps taken by the targets; the following section shows how this can be generalized to represent time. Intuitively, we would expect sensor 0 to be heavily and continuously used by target 1, which follows the small path orbiting sensor 0, and only occasionally by target 0, which covers a much larger area. This is consistent with the load pattern shown in graph S0. Similarly, S2 shows that has only rare and light usage by target 1, which is also intuitively correct. The lower two graphs show the expected measurement rate for each target, where dark gray represents the rate that was actually received, and light gray the additional amount that was requested by not received because of contention. Here, we would expect that target 1 would have a different actual measurement rate when target 0 is close by versus distant. In graph TM1, this is shown at points 0 and 4, where target 0 is in the lower-left and upper-right, respectively. At point 0 target 1 gets only half its requested measurements, while at point 4 (when target 0 is at its most distant point) it is fully satisfied.



**Figure 5.3.** A depiction of the performance predictions by the enhanced DSN model using non-uniform target paths (upper) and sensor locations (lower). The layouts (a,c) show the sensor and target arrangements for two different scenarios. Sensors are circles, targets are triangles and their paths are dashed lines. The graphs (b,d) show the demand levels by sensor (top, darker is target 0, lighter is target 1) and the measurement levels by track manager (bottom, darker is the amount received, lighter is the additional amount that was requested but not able to be satisfied).

The lower scenario in Figure 5.3c has the same path geography, but sensor 2 has been more centrally located. Because it is closer, it is more useful to target 1, and should therefore see greater demand. This can be seen by contrasting the demand curves for S2 in 5.3b and d. In the former S2 is used hardly at all by target 0, while in the latter it is used nearly all the time. This results in the slight increase in expected measurement rate seen in TM1’s rate graph in 5.3d, although the actual rate that target receives is somewhat diminished because S2’s central location also makes it an attractive resource for target 0.

These two scenarios show how non-uniformity has been captured by the model. In contrast, equivalent graphs from the original model would exhibit identical demand across all sensors, and identical performance across all targets.

Although not detailed here, different approaches can also be employed that do not require a rigid path to be defined. For example, instead of a list of points to represent a path, each track manager can be associated with a list of probabilities,

each of which represents the chance that target will occur in a different geographic area. These can be combined across track managers to derive either a statistical average or distribution-based view of how many targets will be in each area. This can be used in a manner similar to that presented above to determine sensor load and actual measurements at each geographic point.

### 5.1.2 Temporal Interactions

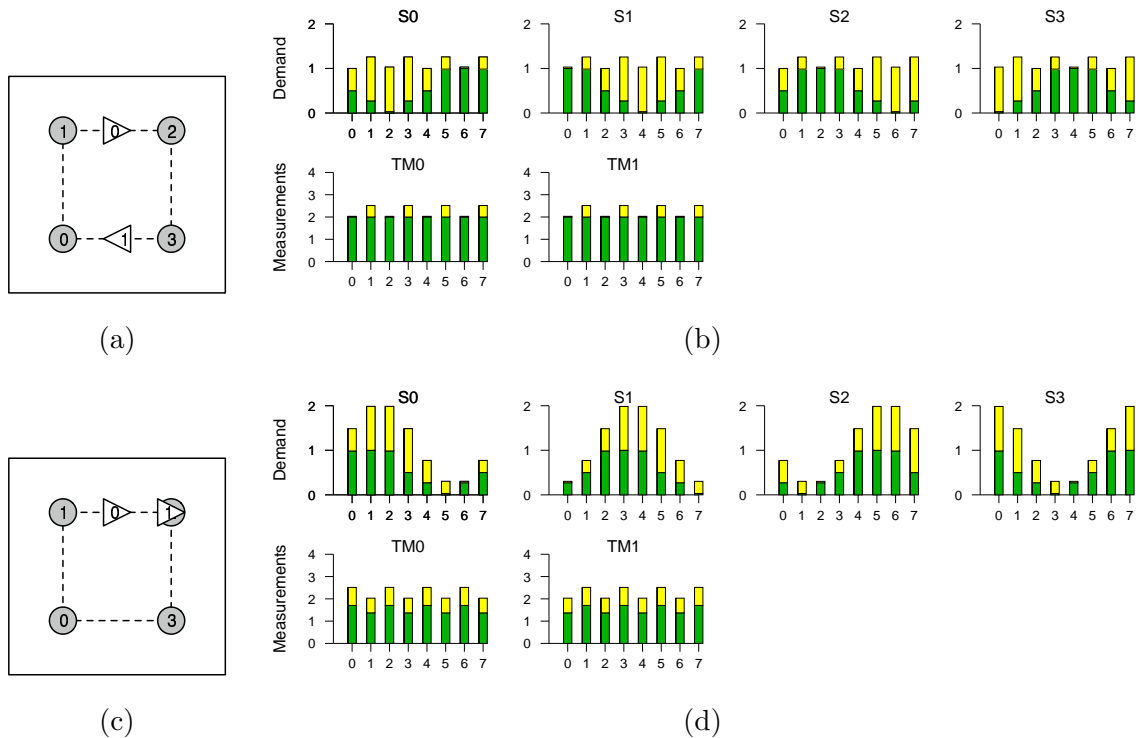
One of the ODML's limitations is its lack of explicit support for modeling time and temporal interactions. Although this was an intentional design choice, there are still many instances when the passage of time or the potential co-occurrence of events can affect the utility of an organizational design. It is because of this, along with my desire to maintain the relatively simplicity of ODML, that I have explored ways of approximating timing information within the existing language. Section 3.3 showed how conventional queuing models are used in the information retrieval model to predict the time-related behavior of that organization. In this section, I describe how a different technique is used by the enhanced distributed sensor network model to predict the demand pattern of targets as they move through the environment.

The previous section introduced the notion of an expected *path* into the track manager's definition. Each path consists of a number of points that represent the potential location of the target. By recognizing that this is not a set of points, but an ordered list, it becomes clear that this representation can be used to approximate the passage of time within the system. At time 0, the target would be at  $(x,y)$  position  $(path[0][0], path[0][1])$ , at time 2 it would be at  $(path[1][0], path[1][1])$ , and so on. Time is relevant in this model because the performance of the organization will depend not only on where the targets will be, but also when they will be there. If two targets are close together in both space and time there will be more contention than if they are separate along either dimension.

Much of the machinery needed to reason about time in this fashion was presented in the previous section. Only two elements need to be added to the track manager node to fully integrate this concept. The first is a *starting\_point*, which is point index of the path at which the target is presumed to start at time 0. The second is a *horizon*, which is the length of the window of time that should be considered. Given these two values, the path is defined as:

```
<constant name="selected_path">mc(paths)</constant>
<constant name="path">fornrange(i, 0, horizon, listitem(selected_path,
(i + starting_point) % size(selected_path)))</constant>
```

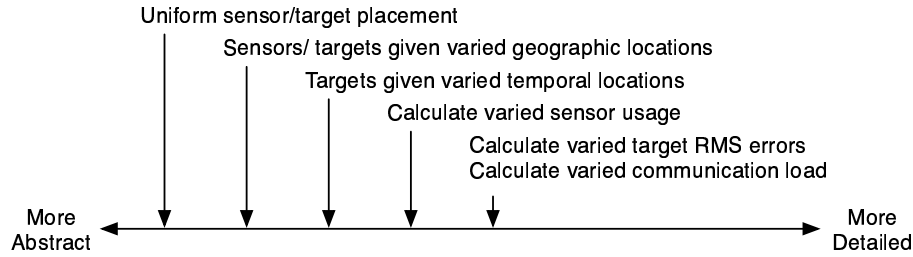
This uses the specific path selected by the *mc* function to fill a list of length *horizon* with the points from that list, starting at index *starting\_point* and cycling back to the beginning when the end of the path is reached. In practice, *horizon* is the length of the longest possible path, while *starting\_point* can be selected from a distribution to add additional environmental diversity. The remainder of the calculations performed in the track manager, sensor-track manager relation and sensor already correlate the



**Figure 5.4.** Performance predictions by the enhanced DSN model, demonstrating the effects that shifting starting points have on performance. Legend is the same as Figure 5.3.

path indexes across targets, so by making this change the model will produce a set of predictions that correspond to different points along a finite horizon timeline.

The results from two scenarios that demonstrate the effects that time can have on performance are shown in Figure 5.4. The first scenario, shown in Figure 5.4a, has two targets that follow the same path, but on opposite sides of that path. In this case the path length is 8, and target 0 has a *starting\_point* of 0, while target 1 has a *starting\_point* of 4. As would be expected, the sensor demand and target measurement graphs in Figure 5.4b show predict the load will be well balanced as time progresses. The scenario in Figure 5.4c depicts an unbalanced distribution, where targets 0 and 1 have *starting\_point* values of 0 and 1, respectively. Because the targets are closer together, the load they impose at any given point is unevenly distributed. This is expressed in the wave-like sensor demand curves in Figure 5.4d. The increased contention for common sensors results in fewer measurements for each target, as can be seen by comparing the target measurement predictions in Figure 5.4d with those in Figure 5.4b. For example, in Figure 5.4d each target obtains 2 measurements at each time point (the dark portion of the bar). In Figure 5.4d the targets alternate between receiving 1.3 and 1.7 measurements per time point.



**Figure 5.5.** A range of modeling possibilities, each with different levels of abstraction.

### 5.1.3 Levels of Abstraction

The notion of abstraction in ODML models has been covered earlier in Section 4.3.2. That discussion presented abstraction through the elimination or assimilation of nodes at the bottom of the organization. The enhanced DSN model described in the previous sections demonstrates another way that abstraction can be employed, through the increase or decrease in detail expressed by the equations defined by the nodes in a model.

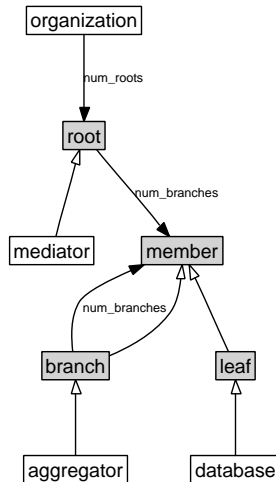
Consider Figure 5.5, which shows a range of modeling strategies, ranging from more abstract to more detailed. The original DSN model existed at the far left of this figure. The modifications presented in the previous sections represent a point further to the right. This represents a change in the level of abstraction that was captured, not through the addition of entire nodes (although that is certainly a valid approach), but through the modification and enhancement of the expressions within the existing nodes.

This is relevant because it highlights a key feature of ODML that is absent from other organizational representations. As discussed in Section 2.2.1, the somewhat primitive, expression-based format allows the designer to incorporate the details the designer deems most relevant. In other representations, assumptions are made about what components will exist, what characteristics may be considered, and in what way those characteristics affect the behavior and utility of the design. By avoiding these assumptions, ODML gives the designer the flexibility to select and change modeling strategies as needed. A simple model might be used during the initial design process, and the model could be refined during the prototyping or implementation phases. The two DSN model variants that have been presented demonstrate how one can trade off the complexity of the model with the accuracy and detail of its predictions.

## 5.2 Modeling Common Organizational Paradigms

Previous chapters, as well as the topics described in Section 5.1, have shown that ODML can be applied to represent a range of individual characteristics in two different designs. This section will show that ODML can also be applied to a range of





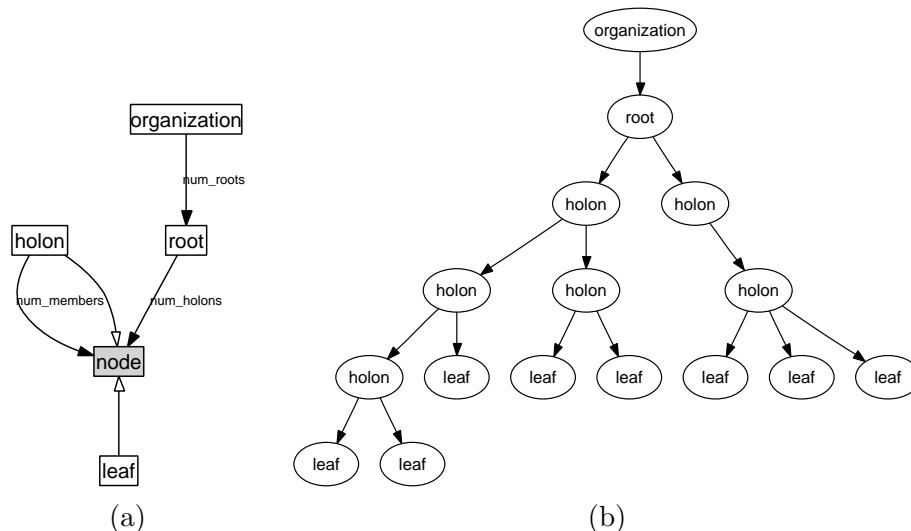
**Figure 5.6.** Using a hierarchy template in the IR domain. The design shown in Figure 3.2b shows a possible corresponding instance.

fundamentally different designs, by discussing a number of organizational paradigms and outlining how they may be modeled. These will be drawn from a survey of the field that I have compiled [80], which can be found in Appendix B. Background information describing the paradigms and their characteristics can be found there. With the creation of appropriate inheritance relationships (see Section 4.4.4), the concepts in models I present can be used as the basis for new, domain-specific designs. Taken a step further, one can envision a library of such organizational templates that could be exploited to simplify the creation of new organizational models. This would be a more quantitative and computational realization of the goals embodied in MIT’s Process Handbook [122].

### 5.2.1 Hierarchies and Holarchies

Hierarchies are ubiquitous in distributed and multi-agent systems because of their relatively simple structure and the immediate scalability benefits that they provide. The information retrieval domain used several such hierarchies as the basis for its mediator-driven approach. That particular model constitutes a complete example of how several different characteristics of a hierarchical system can be modeled using ODML (Section 3.3), including the effect that tree height and width have on timing characteristics, and the trade-offs that consolidation and summarization have on quality as information propagate through the tree.

That model can be used to provide a slightly more complex example of how inheritance and a template library can be used to simplify the design process. Consider the design shown in Figure 5.6. In this case, a generic model of a hierarchy has been constructed, containing *root*, *branch*, *leaf* and *member* nodes. These capture the notion that a hierarchy has a recursive structure, can be of arbitrary height and width, and nodes have particular relationships with their peers. This same information is



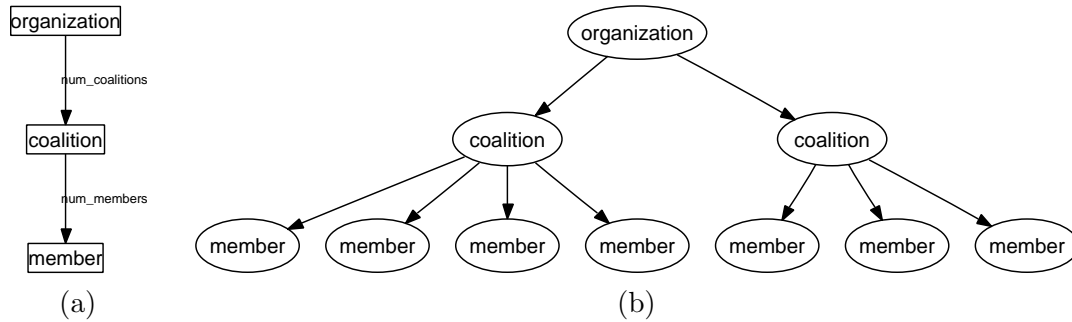
**Figure 5.7.** An ODML holarchy a) template and b) example instance. The structure is quite similar to the hierarchy in Figure B.1, differences lie primarily between their internal behaviors.

also present in the IR domain model, but it had to be defined and embedded specifically for that domain. If a model were created that used this generic hierarchy, this effort could be avoided. Instead, the *mediator*, *aggregator* and *database* roles present in the IR domain would have is-a relationships with their appropriate counterparts in the generic hierarchy, as shown in Figure 5.6. Domain specific knowledge could be passed by overriding constants, as parameters, or by using modifiers. For example, a modifier in *mediator* could add *num\_mediators* to the inherited *num\_branches* value in *root* to specify the number of hierarchy leaders that would exist. It is also possible to migrate the queuing models into these generic nodes, so that (for example) *database* need only provide a task arrival rate to its parent class to reap the benefits of that more complex model.

Conversely, the task-replication behavior exhibited by the IR mediator and aggregator, where a received query is sent in parallel to all children, is sufficiently different from the conventional load-balanced behavior of a hierarchy that it is unlikely to exist in a generic template. In this case, the existing design employing queuing models and order statistics would remain, overriding or ignoring any existing capabilities provided by the generic parent.

Holarchies (Figure 5.7) are quite similar to hierarchies, where individual nodes in the hierarchy are referred to as *holons*. A holon is a group consisting of one or more entities that function as a single unit from the perspective of the nodes above it, but remain distinct to those that exist below it. Unlike participants in strict hierarchies, holons typically also retain a certain amount of autonomy from their superiors.

Viewed in this light, aspects of the distributed sensor network organization can be seen as a simple form of holarchy. Recall that the participants in the organization



**Figure 5.8.** An ODML coalition a) template and b) example instance.

are partitioned into a number of sectors. Each sector contains one or more sensors, along with a sector manager that provides local services. From a vantage point higher in the tree, however, the sector is a single unit that covers a particular geographic region. The behavior exhibited by the sector is determined in part by the sector manager, which acts autonomously in response to perceived global and local needs. By the simple definition given above, the sector itself can therefore be considered a holon. Furthermore, if the DSN existed as part of a yet larger structure, the DSN organization as a whole, consisting of all the available sectors, could also be considered a holon.

### 5.2.2 Coalitions and Congregations

The general form of a coalition is quite simple – it is simply a grouping of entities that have banded together to serve some common purpose. A suitable template is seen in Figure 5.8a, which shows that both the number of coalition, and the number and type of participants in each coalition can vary. A sample coalition instance is shown in Figure 5.8b.

In their purest form, coalitions are disjoint, so that entities may be a member of only one coalition at a time. This constraint can be represented in ODML in the same way that the one-sensor-per-agent condition was modeled in the DSN organization. Each agent would have a *num\_coalitions* characteristic, which would be initialized to zero and incremented through the use of a modifier as they are added to a coalition. Each agent would also specify a *max\_coalitions*, along with a constraint  $num\_coalitions \leq max\_coalitions$ . By setting *max\_coalitions* to one, the disjoint constraint will be upheld. Some researchers have demonstrated the utility of relaxing or removing the disjoint constraint [164]. This can be modeled by setting *max\_coalitions* to some value greater than one. Additional research has explored the possibility of hierarchical coalitions, where coalitions may be nested within one another [102]. Such an organization would closely resemble the holarchy shown in the previous section.

There are three key characteristics associated with coalitions that must also be represented: the strength of the resulting group, the costs associated with formation and maintenance, and the manner in which rewards (if any) are apportioned to the

participants. These can take many forms, so it is worth exploring how a range of possibilities might be modeled. The strength of the coalition, for example, is in some cases simply the number of participants, or some valuation of the total “mass” of the participants. A bargaining collective or union are examples of this, and can be modeled with a simple summation in the *coalition* node:

```
<constant name="strength">forallsum(m, members, m.mass)</constant>
```

Consider a somewhat more complex situation, where participants can have different types of skills and different levels of proficiency in those skills. Furthermore, assume the goal requires some minimal combined skill set, and there are upper bounds beyond which further capability adds no additional strength. For example, assume each member agent has constants *skill\_a* and *skill\_b*, which are assigned their respective numeric levels of proficiency. The coalition gets no strength if the total proficiency level of *a* is less than 0 or if the total level of *b* is less than 6, and gets no additional benefit if *a* is greater than 1 or *b* greater than 10. The coalition’s strength can then be modeled using sigmoid functions as:

```
<constant name="skill_a">forallsum(m, members, m.skill_a)</constant>
<constant name="skill_b">forallsum(m, members, m.skill_b)</constant>
<constant name="strength_a">1 / (1 + e^-((skill_a-0.5)/0.1))</constant>
<constant name="strength_b">1 / (1 + e^-((skill_b-8)/0.5))</constant>
<constant name="strength">strength_a * strength_b</constant>
```

If the set of member nodes was limited, through a node instance limit or a constraint-based mechanism, then a purely strength-driven assignment of the members to coalitions would use these definitions to make the contextually appropriate trade-offs. As before, if these expressions were embedded in a reusable template, they could be overridden in the derived class to whatever form is most appropriate.

Modeling coalition cost can be quite similar to modeling strength. For example, instead of aggregating the strength of the participants, one could create a profile of the total communication behavior, and bound it against the available bandwidth in the environment. There may also be fixed costs associated with coalition formation, such as the time needed needed to elect a leader or disseminate goal or participant information. Once the costs have been determined, they can be combined with the strength to determine the overall utility of the coalition.

In a self-interested situation, it is not only the utility of the coalition as a whole that is of interest, but also the perceived benefit that individual members will observe. This is frequently described as the “reward” the agent will receive. If there is a fixed amount of reward available, the manner in which the reward will be distributed is a key factor that can determine if an entity will choose to join the coalition. A simple approach is to divide the reward into equal portions among the participants. In the *member* node, this would be defined as:

```
<constant name="reward">c.reward / c.num_members</constant>
```

...where  $c$  is a reference to the parent *coalition*. A slightly more complex approach would divide the reward according to the proportional benefit the member brings to the coalition (i.e., its *strength*):

```
<constant name="reward">c.reward * strength / c.strength</constant>
```

which would encourage the inclusion of valuable members, particularly if costs grow proportionally with coalition size.

Congregation-based organizations differ from coalitions in that they are longer-lived, and not necessarily formed with a particular goal in mind. The intent is to band together entities that have similar or complementary skills, so that individual entities can more efficiently make use of those skills. So, where the *coalition* was previously used as the locus of capability evaluation, the individual members will individually do so in a congregation model, using the containing *congregation* node as a common data store.

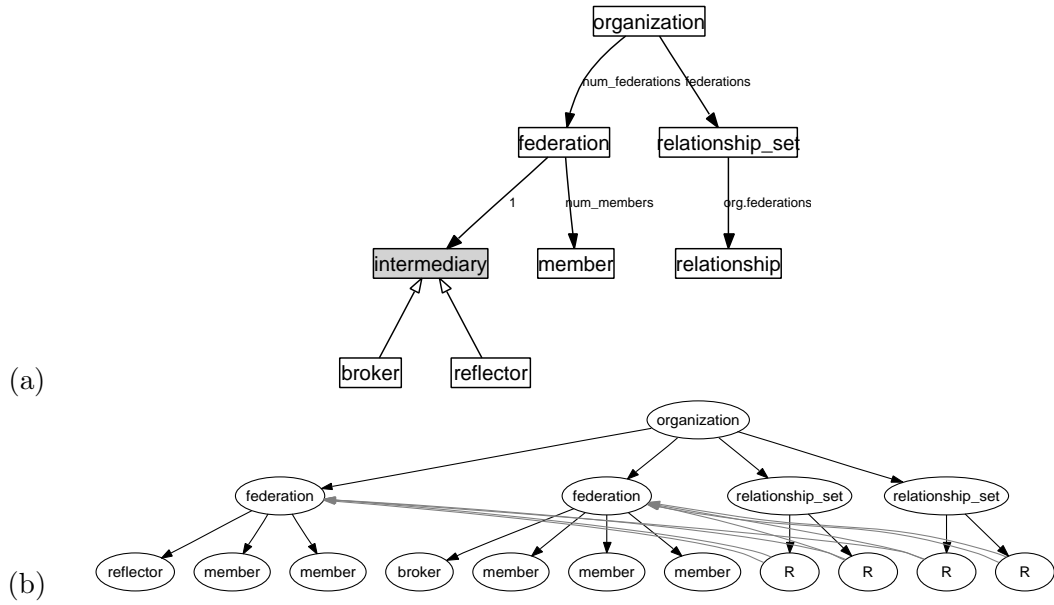
For example, assume that each agent supplies its local proficiency levels to the *congregation*, where they are aggregated as before into *skill\_a*, *skill\_b*, etc. To avoid the need to explicitly enumerate all skills these could also be represented as lists of numbers. Each agent would then have a set of utility definitions that expressed how much of each type of skill it could make use of, in the same way that *strength* was previously defined through a composition of skill strengths. Unlike coalitions, which are relatively short-lived, the creation costs associated with congregations are secondary. Of more importance is the continuing overhead of processing and communication needed to interact with other congregation members.

As before, these strengths and costs can be combined to derive a utility value. In this case, however, utility is determined by the individual agent, replacing the reward structure used in coalitions. This distributed utility makes evaluating the overall organization somewhat subjective. Constraints could easily be added to place lower bounds on individual utility. Overall utility statistics could also be gathered, allowing global utility to be defined in terms of minimum or average observed utility, which would allow Pareto optimal organizations to be favored.

### 5.2.3 Federations

Like coalitions and congregations, federated systems are organized around agents that are grouped into clusters. Unlike those systems, federations are formed with the express purpose of consolidating the group's functionality behind an intermediary agent that acts as an interface for the group. In particular, in forming a federation, the participants cede authority to the intermediary, allowing it to guide their local choices of action and goal. Figure 5.9a shows how a federation template can be represented in ODML. Here the number of federations in the system and the number of members in each federation can vary, as can the relationship between federations and the type of intermediary that is used. A sample two-federation instance is shown in Figure 5.9b.

As with coalitions, the members of a federation can have different skill sets, that can be represented and accumulated in the federation node that contains them. This



**Figure 5.9.** An ODML federation a) template and b) example instance. A federation consist of a set of members that have relinquished control to a distinguished *intermediary*. The *broker* and *reflector* nodes represent two possible intermediaries. Only intermediaries interact between federations, as represented by the links through *relationship* nodes in b).

can be used to calculate the strength of the federation. Costs can be similarly calculated, although they will typically be more concentrated than in coalitions or congregations because the members of a federation only interact with their local intermediary. Where the communication load of an individual member of a congregation may increase with both the size of the congregation and the number of tasks, a federation member's communication load will depend on only the number of tasks. The intermediary's burden, however, will depend in part on the size of the federation. In particular, it will depend on the size of its local federation, the number of tasks in the environment, and the number of other federations in the system. This last dependency arises because intermediaries in different federations interact with one another to advertise work to be done or capabilities that can be offered.

A distinguishing aspect of the federated design is the presence of an intermediary. This agent connects the internal members to the outside world, and can be endowed with different capabilities that leverage its place in the organization. The template shown in Figure 5.9a allows two different types of intermediary: the *broker* and *reflector*. These particular terms have been given different definitions in different research contexts. Here, I assume that a reflector is acting as a simple translation service, routing internal requests to outside entities, and reflecting external queries to all federation members. In this case, a member's work request is routed through the re-

reflector to all other federates, from which many responses may be received. Individual members behind a reflector are responsible for deliberating among alternatives.

Brokers take a more active role. In response to an internal task request, a broker may query only a subset of external federations, and select the most appropriate response. When an external task request is received, a broker has the authority to assign that task to any member of the federation it represents. A broker must therefore maintain state describing its federation members and any extant work commitments, and be able to reason about how to best satisfy incoming and outgoing requests.

Several of the characteristic features of the federation paradigm have been discussed and modeled previously. For example, the DSN model demonstrated how to describe and aggregate communication load and the effects it can have on performance. Both the information retrieval and coalition models showed how to represent the capabilities of entities, and how those capabilities can influence the behavior and quality of the larger organization. The information retrieval model also showed how to capture the effects that perceived substructure quality have on a search process among members of the organization, and the work distribution patterns that can be produced as a result (Section 3.3). A quality-driven search is also typical of the federation, either by the members via a reflector proxy or directly by a broker. This could be modeled using a formulation similar to Equation 3.6, using a federation's capacity to perform work to estimate quality. The model I describe below address the same issue in a different way, by creating a fixed set of relationships between federations that guide how work is apportioned among them.

I will focus the following discussion on how a broker-style intermediary can be represented. In this model I will assume that all member agents have a *task\_arrival\_rate* (*tar*) that reflects the rate at which it receives unit-time tasks from the environment. It will satisfy them locally if possible, and if not it will ask its local intermediary to find a suitable worker. On average, if  $tar > 1$  the agent will request external help at rate  $tar - 1$ , if  $tar < 1$  the agent will have excess capacity of size  $1 - tar$ .

```
<constant name="max_work">1</constant>
<constant name="local_work">min(max_work, tar)</constant>
<constant name="requested_work">tar - local_work</constant>
<constant name="spare_capacity">max(0, max_work- tar)</constant>
<constant name="external_work">i.external_work * (spare_capacity
  / i.local_capacity)</constant>
<constant name="work_rate">local_work + external_work</constant>
```

This formulation breaks the member population into two sets: those which request work to be done and those with the capacity to service those requests. For those that have extra capacity, the *external\_work* characteristic deduces how much extra work will be asked of the member, based on its proportion of the total spare capacity of the federation and the amount of extra work that has been accepted by its broker intermediary *i*. This relies on the additional information aggregated by the intermediary:

```
<constant name="local_work">forallsum(m, federation.members,
```

```

    m.requested_work></constant>
<constant name="local_capacity">forallsum(m, federation.members,
    m.spare_capacity)</constant>
<constant name="requested_work">max(0, local_work
    - local_capacity)</constant>
<constant name="spare_capacity">max(0, local_capacity
    - local_work)</constant>
<constant name="external_work">0</constant>
<constraint name="external_work" op="<=">spare_capacity</constraint>

```

Assuming the broker will prefer to use the local extra capacity before making external requests, the amount of *requested\_work* that will be sought can be computed from the *local\_work* and the *local\_capacity*. The remaining *spare\_capacity* (if any) can be similarly computed. If we wish to develop a probabilistic model like that used in the information retrieval domain, the *spare\_capacity* characteristic can take the same role as *perceived\_response\_size* (that is, an approximation of the federation's quality). In this model, these values are instead used by a set of *relationship* nodes contained within a per-federation *relationship\_set*, to derive and evaluate a specific pattern of interactions among the federations:

```

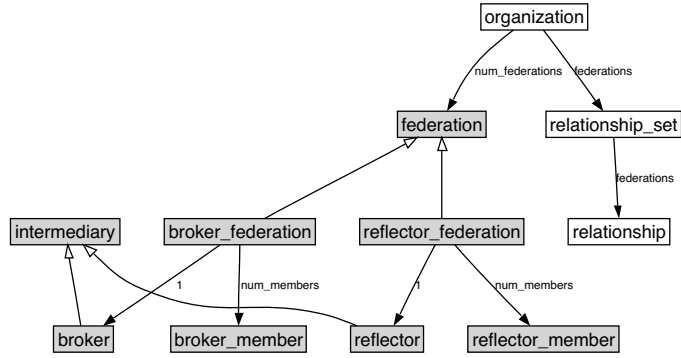
<node type="relationship_set">
  <param>organization:org,federation:source</param>
  <has-a name="relationships">forall(target,federations):
    relationship(this,source,target)</has-a>
  <constant name="requested_work">forallsum(r, relationships,
    r.requested_work)</constant>
  <constraint name="requested_work" op="=">
    source.requested_work</constraint>
</node>

<node type="relationship">
  <param>relationship_set:set,federation:source,
    federation:target</param>
  <variable name="work_proportion">0, 0.25, 0.5, 0.75, 1</variable>
  <variable name="requested_work">source.requested_work
    * work_proportion</variable>
  <modifier name="target.requested_work op="+">
    requested_work</modifier>
</node>

```

Each *relationship\_set* connects its *source* federation to all possible *target* federations in the organization. In each *relationship*, the proportion of the source's *requested\_work* to assign to the target is represented as a variable. This allows the model to represent a federation assigning some, none or all of its extra work to other federations. Constraints ensure that the total amount requested and assigned do not exceed the amounts available at the source and target, respectively.



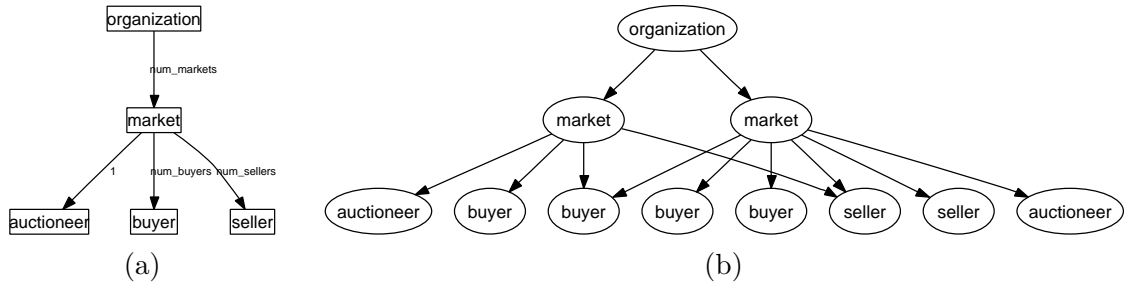


**Figure 5.10.** An alternate ODML federation template that allows greater differentiation among intermediary styles but is less able to inherit and reuse common elements.

There are various costs associated with the work distribution organization that have been omitted for clarity. For example, a member performing its own work will have the least amount of communication, followed by request redirected within the federation, followed by one redirected to an external federation. Entities in this organization may also suffer from the same queuing problems addressed in the information retrieval model. Both communication and processing costs can affect the intermediary, making large federations less efficient. In each of these cases suitable cost expressions can be devised and used along with the aggregate work rate to calculate the utility of the federated organization.

The expressions and node definitions that have been given so far assume that the intermediary is acting as a broker. As mentioned earlier, there are several styles of intermediary, each with potentially different characteristics. A natural question that arises is how the model must be changed to correctly represent a co-existing alternative like the reflector described above. To answer this question the modeler must identify the components of the model that would be affected by such a change, and either design those expressions so that they can obtain and use the characteristics that are subject to change, or create a new, complementary set of nodes to be used when the other decision is made. These alternatives correspond to the templates in Figure 5.9a and 5.10, respectively.

For example, in Figure 5.9a, the federation model is reused for both types of intermediary. The model implicitly assumes that the choice of either *broker* or *reflector* can be accommodated by the common *federation*, *member* and *relationship* nodes, and any nonlocal behaviors can be appropriately exhibited through or within the *intermediary* interface. Some of the changes, such as the communication and processing costs incurred by the intermediary itself, seem tractable. Appropriate expressions can be devised for within the *broker* and *reflector* definitions that compute those values. What is less clear is how the behavior of the members will be affected. In practice, the member node behind a reflector may require significant additional functionality



**Figure 5.11.** An ODML market a) template and b) example instance.

to operate correctly. Simplifying assumptions may need to be made to combine the quality and cost calculations necessary to correctly reason under this circumstance.

The model in Figure 5.10 depicts an alternate template that makes fewer assumptions, but in doing so is less able to re-use common components and is correspondingly more complex. The benefit it provides is that by creating a distinct member types for each intermediary type, it is possible to more directly and accurately model the organizational alternatives. For example, the original template does not allow for the possibility that entities in the environment may be unable to handle the additional complexity required when interacting with a simple reflector. Legacy or light-weight agents may not have the computational or software resources to correctly decide among alternative tasks asked of it, or providers advertised to it. By creating separate *broker\_member* and *reflector\_member* nodes, an entity or agent could create is-a relationships with either or both to reflect its actual capabilities.

## 5.2.4 Markets

Superficially, market organizations share several characteristics with federations. Both involve groups of participants that are interacting almost exclusively with intermediaries (an auctioneer, in the case of markets). In both there are different types of intermediaries with different characteristics. Both frequently revolve around the allocation of services or goods among the participants.

Unlike a federation, market participants do not cede authority to their auctioneer. Auctioneers do not generally interact with one another. Participants are typically competitive, and can take part in several auctions concurrently. There are also a great deal of subtleties involved in the bidding and awarding process that can be difficult to model directly, but can have significant impact on the behavior of the system. For example, the truthfulness of bidders, the potential for collusion and counter-speculation and the type of auction that is used can all play a large role in the performance of the system. Producing accurate, mathematical models of these characteristics has been studied in the fields of game theory and economics, and is beyond the scope of this document. In this section, I outline how results from these fields could be incorporated into an organizational model.

A template for a market-based organization can be seen in Figure 5.11a. This can vary the number of markets, the set of buyers and sellers in each market, and the type of auctioneer that is used. The participants in each market may not be disjoint, as shown by the instance in Figure 5.11b. Each seller will have a set of products types it needs sold, along with a frequency for each. This can be represented as a list of floating point values:

```
<constant name="products">[0.1, 0.25, 0.3]</constant>
```

where each element represents a different product. Each buyer will have a complementary *products* description of the products they desire and when they will need them. By aggregating this information, the auctioneer can determine the supply and demand profiles for each product type. A finite-horizon timeline approach similar to that seen in Section 5.1.2 can be used to determine who wants what when, and how long it will be before they have a chance to obtain it. The auctioneer can use these values to produce an estimate of how long it will take the auction to converge and a winner decided. Using a suitable abstraction of the buyers' behaviors, sellers can use this same information to estimate the sale price, while buyers can similarly estimate the cost they will be expected to pay. This, along with the inherent utility each entity places on the product and the time needed to acquire it, can be used to estimate local utility.

This modeling approach implicitly assumes that agents have full observability of one another's state, because that information is needed to produce the information flows that allow concrete predictions to be made. Because markets are competitive environments, this is not necessarily a reasonable assumption. This would be the case if the template were being used to create the entire marketplace, but the open nature of the marketplace design typically means that only part of the market's design is under consideration at any one time. In particular, I will show below how a variant of this model can be used to guide the local decision making process.

More so than other paradigms that have been discussed, markets are frequently preexisting, open systems put in place that agents can choose to join or not. Because of this, the marketplace design problem is often less one of creating an entire organization than it is deciding what type of market to create (from the auctioneer's perspective) or what markets to join (from the buyers' and sellers' perspectives). Thus, a useful ODML model would be one that enabled the entity to reason about those classes of decisions.

The key insight needed to support this use is to recognize that the majority of the model will be used to evaluate a relatively small number of design decisions (typically confined to a single individual) that must be made within an expected operating context. This is different from most of the models that have been previously shown, where the intent was to represent complete organizational alternatives, and decision points existed throughout the model. This is most closely related to the existence of targets in the DSN model. In that case, a suitable sensor network organization had to be found, given an environment that contained a set of independently moving targets. In this situation, a number of decisions need to be made for a particular individual

in the market, given an environment that contains a set of independently acting participants. As with targets in the DSN model, the participants can be modeled with distributions. This avoids the observability problem by allowing the individual to evaluate its decisions with respect to a range of different conditions rather than a single exact instance, under the presumption that it is reasonable the agent would be able to define an estimate of the conditions it will operate in.

For example, assume that the market owner needs to determine what sort of auction to create. This might include specifying upper or lower bounds on the number of participants, whether to impose time or bid granularity limits on the auction process, and the categories of products that will be sold. Each can be specified with a variable, and perhaps a range of auctioneer types, and an appropriate set of expressions that predicts the ramifications of that choice in the context of some expected set of market participants. The set of participants can then be represented with a series of distributions within the *market*, *buyer* and *seller* nodes that control the number of buyers and sellers, their internal behaviors, and so on. A Monte Carlo evaluation process can be used to sample from these distributions, producing a range of possible environments in which the market owner's design decisions can be evaluated.

A similar scheme can be used by a candidate participant. It must decide how it will behave and what market(s) it should join. The market and auctioneer (which presumably already exist and are known) would be static entities in this case, while a similar set of distributions can be created to model the expected set of peers that might be observed over time. By again using Monte Carlo evaluations, the participant could correctly decide behavioral choices and place itself within the organization to maximize its expected local utility.

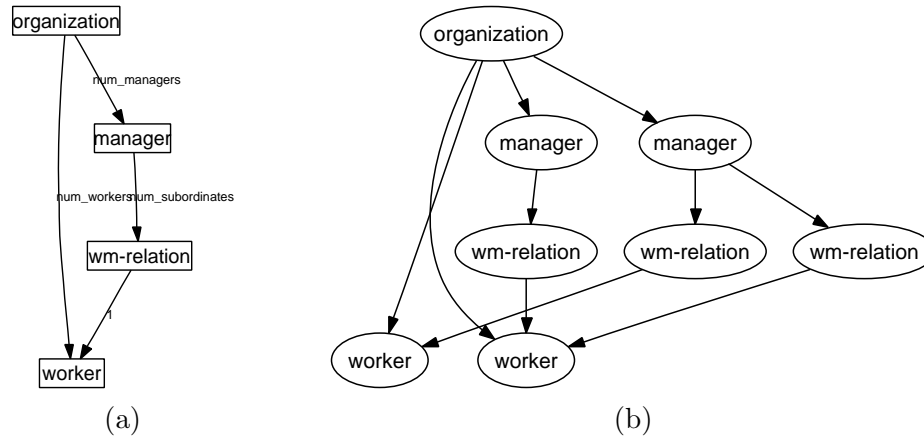
### 5.2.5 Matrix Organizations

The primary benefit of a matrix organization, as compared to a simpler hierarchical structure, is that goals can be addressed more flexibly by performing more work with the same amount of resources. This is accomplished through the reuse of worker time that would otherwise be spent idle, or through the intelligent selection of actions that can take place when the actor has a more complete view of its working context.

The structure itself is fairly simple but potentially tangled because of the mass of manager-worker relationships. Figure 5.12a shows a matrix organization template, where the number of workers and managers can vary, as can the number and specific assignment of workers to managers. A small two-worker, two-manager instance is seen in 5.12b.

Like other paradigms that have been discussed, tasks of particular types will arrive in the organization with a particular frequency. We can assume that they will arrive at the managers, each of which can have a list of task arrival rates:

```
<constant name="arrival_rates">[0.5, 0.1, 0.0, 0.9]</constant>
<constant name="available_service_rates">forallsum(r, subordinates,
  r.worker.service_rates)</constant>
```



**Figure 5.12.** An ODM matrix a) template and b) example instance. The *wm-relation* node is used to propagate the effects of each manager to a worker.

where each list slot corresponds to a particular type of task, and the *available\_service\_rates* list is used to keep track of the service capabilities of the workers that report to it. Each worker can have a corresponding set of service rates, defining how quickly they can perform each type of task:

```
<constant name="service_rates">[0.9, 0.0, 0.5, 0.1]</constant>
```

Within the *wm-relation* node, the particular amount of work that will be assigned to the worker can be determined based on the per-type proportion of service that worker can provide to the manager. This information can then be applied to the worker with a modifier:

```
<constant name="work_rates">forrange(i, 0,
  size(manager.service_rates), listitem(manager.arrival_rates, i)
  * listitem(worker.service_rates, i)
  / listitem(manager.available_service_rates)</constant>
<modifier name="worker.work_rates" op="+">work_rates</modifier>
```

This straightforward approach does not permit the manager to request a disproportionate amount of service from any one worker. For example, the manager who's *arrival\_rates* are given above could not use the worker specified above for only its tasks of type index 0, it would also use it to service tasks of type 3 (the only other type where the manager has arrivals of tasks the worker can service). If disproportionate assignments are desired, a separate set of variables would be needed to make the organizational decisions defining how much each worker should be used for each task type. The manner in which this load would be passed to the worker would remain the same.

After these connections are formed and the work assignments decided, each worker will have a list of the total demand that is being placed on it, broken down by

task type. As with the information retrieval organization, the behavior produced by these arrival and service rates can be predicted using queuing theory. This problem differs from that one in that there is just a single level of queues, which reduces the complexity of information that must be passed. On the other hand, there are several different classes of task, each with different arrival and local service rates. Mathematical expressions have been developed that allow the service time for each task to be estimated [87], and can be incorporated into the ODML model. This work completion delay can then be collected by the managers to help estimate the utility of the solution.

This model will capture the detrimental effects of potential over-usage of common entities, which will be exhibited through slower response times and lengthening queues. What is not yet modeled are the potential benefits that can be obtained when similar or complementary tasks are performed by a single provider. For example, if two managers required a task to be performed, and one agent could serve both in the same amount of time needed to serve one, then a net reduction in duration would be seen if the managers worked through that same agent. Detailed, situation-specific interactions, such as the effect that relative arrival time has on performance, would be hard to capture exactly in an ODML model. An approximation is more practical in this case. For example, to model the effect of avoided redundant tasks, the worker could artificially reduce the arrival rate of a task type based on the number of managers:

```
<constant name="efficiency_gains">forrange(i, 0, size(work_rates),
  listitem(work_rates, i) * -1 / e^(0.01*(listitem(num_managers, i)))
  + 1</constant>
<constant name="actual_work_rates">work_rates
  - efficiency_gains</constant>
```

where *num\_managers* is a list of the number of managers with non-zero arrival rates for each task type, and the agent now uses the *actual\_work\_rates* list to compute expected durations. This adjustment will gradually increase the efficiency of the worker as more managers ask it to perform tasks of the same type. An inverse adjustment could also be formulated that negatively affects work rate when many different types of tasks are given to the agent. This could be used to model the efficiency lost through the context switching that is performed as the agents shifts focus.

## 5.2.6 Societies

Unlike the paradigms that have been explored thus far, agent societies do not typically subscribe to a common form that is amenable to direct modeling. By their most general definition, an agent society is simply a group of agents that have agreed in some way to exist in a shared social structure. The common element across society instantiations is the existence of a set of social laws or norms that are used to control agent behavior. These serve the same purpose as human laws in providing a common foundation from which simplifying assumptions and guarantees can be made.

These characteristics are perhaps most closely related to the design issues raised by the market paradigm. Like markets, societies are open systems, so design choices are typically focused on the initial creation or maintenance of the society framework (i.e., selecting the appropriate set of laws) and whether or not an agent should (be allowed to) join an existing society. The same modeling approach used in the market discussion may work here, which is restricting the organizational decisions to a small subset of the total possible participants and using distributions and Monte Carlo analysis to approximate the remainder of the system.

The question then, is how well ODML can capture the concepts that lie behind norms, both as an explicit design choice and as an extant characteristic that affects agent behavior. López y López and Luck present a general model of norms from which we can evaluate ODML's effectiveness [211]. In that work, three classes of norms are described: obligations and prohibitions, social commitments and social codes. Obligations and prohibitions are norms which when violated have some sort of explicitly penalty or punishment. Social commitments are formed between individuals to codify a negotiation or coordination event, and are typically associated with some sort of reward. Social codes have neither explicit rewards or punishments, but are norms that must be respected to remain in the society.

The intent behind social codes is consistent with ODML's notion of organizational validity. They are edicts that must be followed, there is no choice, and therefore if they can be quantified it should be possible to represent them in ODML. For example, the "one sensor per agent" constraint in the DSN model could be construed as a social code. Another could be that all agents must take on at least one role. Yet another could put a lower bound on the acceptable RMS tracking error. In each case, if the code is broken the organization is considered invalid. A process creating the society could evaluate a set of social laws by determining the frequency at which invalid organizations are produced during the Monte Carlo process. An entity trying to determine if it should join a society can insert itself into a similar model and verify if the society remains valid.

Obligations, prohibitions and social commitments are more like soft constraints. There is some leeway in the norm, in that if it is not satisfied something bad will happen, or something good will not happen. In fact, if we view a punishment as a negative reward, they can be treated as a single effect. For example, there is an implicit norm embedded in the track manager-sensor relationship that obliges the sensor to do work for the track manager. If that work is not performed, the RMS error of the track manager will increase and the utility of the organization will decrease. In a more self-interested context the model would need to assign the change in utility directly to the entity at fault. There is an implicit social commitment behind most of the relationship nodes that have been defined in the models that have been presented. In each case there is some amount of work, information or raw utility that is provided to one or both of the parties in question. If there is some probability the commitment will not be satisfied, because of scarce resources, operational faults or simple bad luck, this can be expressed with a probabilistic distribution. By incorporating that distribution into the reward or penalty calculation, the norm's effects can be correctly propagated.

My experience thus far has shown that norms are more easily represented and incorporated implicitly. The problem with this approach is that it is less useful if one is trying to reason about the inclusion or exclusion of a particular norm. To overcome this, a way of defining the norm as explicit choice must be found, so that it can be reasoned about during the design process. This is typically done by defining a variable or creating a node that embodies the characteristics of a norm. For example, one could create a variable in the DSN's *environment* that determined the number of sensors each agent could control (e.g., a choice of 1 or *MAX\_INT*). A choice of two different nodes derived from a common *relationship*, one which enforces penalties and one which does not, could be used to reason about the utility of social commitments. In both cases the decision of norm inclusion will not necessarily be as explicit as one would like, especially in contrast to other representations that maintain norms as a discrete set that agents can inspect and reason about. However, it should be possible for the designer to create a mapping to produce such a set with the same knowledge used to design the model itself.

### 5.2.7 Teams

There are at least two key characteristics of a team-based organizations. The first is the increased productivity, redundancy or utility that can be obtained through the cooperative and coordinated efforts of a number of individuals. The second is the increase in coherence that arises when individuals' decisions are grounded in a representation of team-level objectives, shared beliefs and joint intentions. I will explore how the former characteristic can be modeled first.

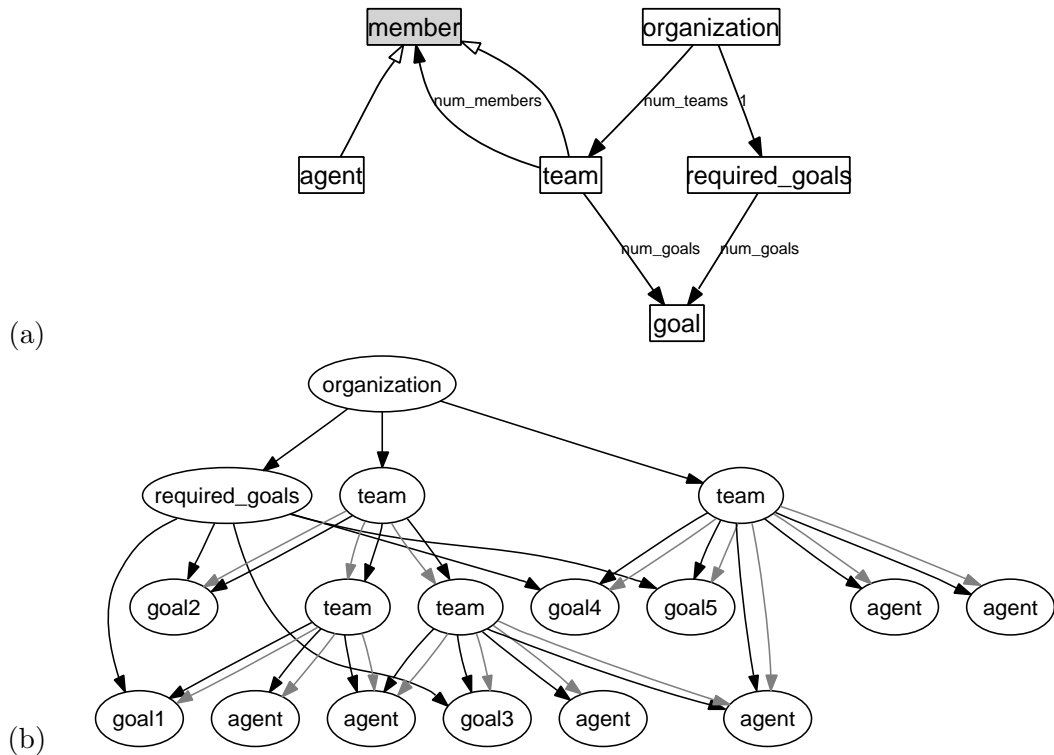
An ODML team template is shown in Figure 5.13a, which allows for different numbers of teams of varying composition, hierarchical sub-teams, team-specific goals, and non-disjoint team membership. An example instance produced from that template is shown in Figure 5.13a, exhibiting most of these features. The structure of the organization as a whole is driven by the set of goals that need to be achieved. This is specified with *required\_goals*, which contains a set of *goal* nodes. Each goal needs to be assigned to a team, the composition of which is determined from the *goal* nodes it is associated with. Each *goal* has a set of requirements, that correspond to skill sets:

```
<constant name="requirements">[1, 0, 2]</constant>
```

As with matrix organizations, each slot in the list corresponds to a skill type, with the number indicating the total amount of skill that is required. Each agent possesses a corresponding *skills* list, which is aggregated in the team to determine the total amount of proficiency that is available. Agents belonging to multiple teams have their skills distributed evenly among them. The requirements of the goals are aggregated within the team, and a constraint used to ensure that the total amount of available skill meets or exceeds the goals' requirements.

This approach treats the team design process as a resource allocation problem, where agents with skills are allocated to particular goals. The contents of *required\_goals* are generally not an organizational decision, they are an environmental factor being

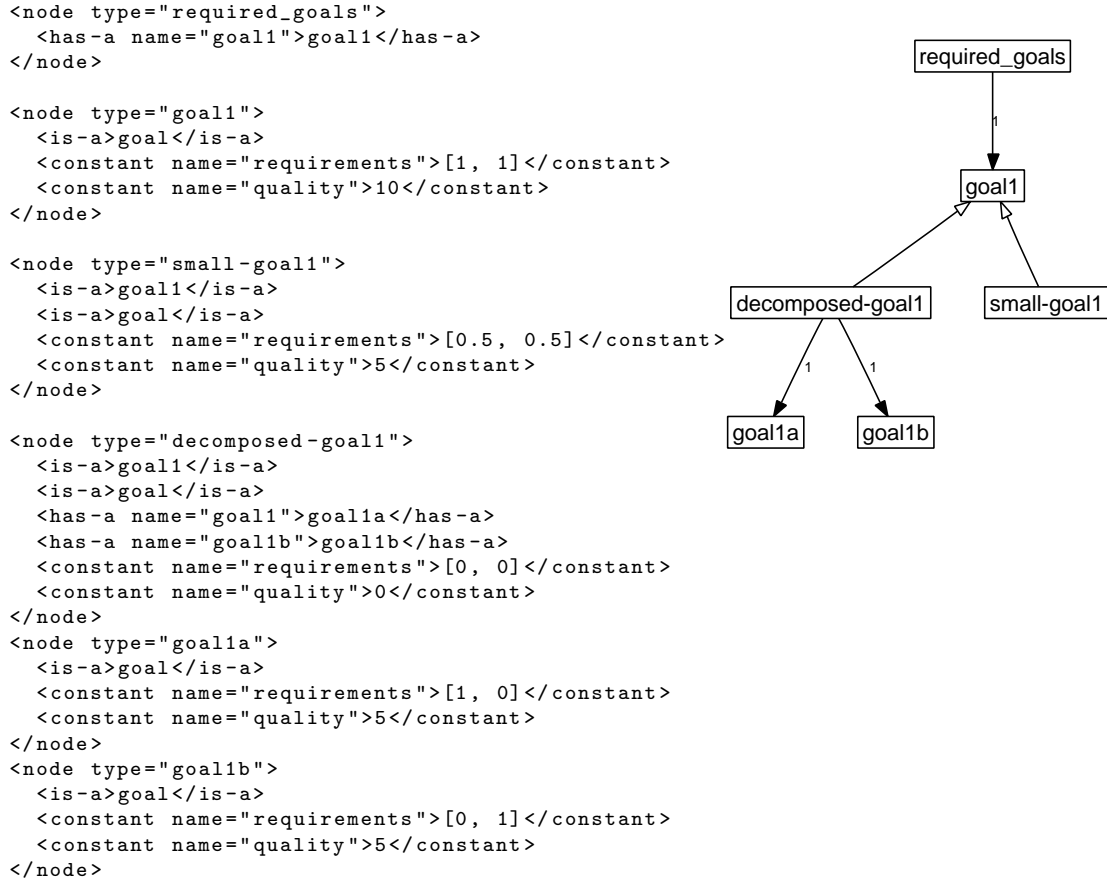




**Figure 5.13.** An ODML team a) template and b) example instance. Has-a relationships from *teams* to *goals* are used to allocate work among the entities.

addressed by the team organization being created. Modifiers from each team and constraints within goals are used to ensure that each goal is taken on by a single team, in much the same way that the agent-sensor proportion was enforced in the DSN model. Adjustments to this mechanism could be used to reward redundancy, by allowing multiple teams to pursue the same goal. The utility of the organization can be calculated from the set of goals that are satisfied, each of which can be assigned a *quality* that is aggregated up the team hierarchy. If goals have an arrival rate and are not singular events, a queuing model can be put into place to estimate delays, which can also be incorporated into the utility metric.

The second characteristic of team-based organization is the increased coherence that can be expected, because the participants have a more explicit, global sense of their purpose and can recognize how their local actions may interact with their peers. The drawback to this technique is the necessary communication required to maintain the teamwork semantics, as participants exchange local information. Communication is more frequent among members of the same (sub-) team, and generally increases with the number of members in that team. It is this reason why a single, all-encompassing team approach does not scale well. This is represented with a set of modifiers in the *team* node that increase the agents' *communication\_load* by an amount proportional



**Figure 5.14.** Portions of an ODML team model supporting goal alternatives.

to the size of the team. Constraints within individual agents ensure that this load is kept to a sustainable level in any valid organization.

One aspect of team formation that this model does not address is the explicit possibility of alternative ways to achieve a goal. For example, it might be possible to achieve *goal1* using a different set of skills by reducing the expected level of quality. Similarly, it might be possible to satisfy *goal1* with a pair of smaller goals that represent a decomposition of the original objective. It is possible to model these conditions by taking advantage of ODML's is-a relationship.

For example, consider the task structure excerpt shown in Figure 5.14. This model contains the original *goal1* as well as two additional alternatives that may be used in its place because of their is-a relationship with it. *small-goal1* is still atomic, but has weaker requirements and produced correspondingly less quality. *decomposed-goal1* itself has no requirements or quality, but does have two sub-goals. These sub-goals act as ordinary goals, and therefore can be assigned to separate teams. It should be possible to translate simple AND-OR goal trees into this type of model. It is not clear if the complete semantics of more complex representations (e.g., TÆMS [40]) can be represented. It seems likely that an approximation would be required that sacrifices some of the original fidelity.

An aspect not captured by this initial team model is a notion of time. Specifically, goals may need to be achieved at different times, which affect when their corresponding requirements must be fulfilled. The allocation mechanism described above lacks any temporal directives, implicitly assuming all goals will be pursued concurrently and producing an excessively conservative organization. An approximate timeline approach like that described in Section 5.1.2 can be useful here. Each goal can be associated with a time and duration, which the aggregation process can use to create a timeline of skill requirements. Constraint validation can ensure that the necessary skills are available at each point. This technique would not be able to handle uncertainty or choice in the timing specification. For example, it could not shift goals around in an effort to optimize the allocation process, a problem that is much better suited to a conventional scheduling engine.

More generally, one can ask how much of the agent's internal scheduling process can and should be represented at this level. Even if it were possible, it seems unlikely that one would want or need to re-implement a scheduler within the ODML model. The necessary level of detail then depends on how crucial the low-level characteristics of that process are to organizational performance. For example, the temporal interactions from Section 5.1.2 were in some crude way a prediction of the sensors' attempts to schedule multiple competing tasks. That section demonstrated that these effects can be important, which motivates their inclusion into the model. As with other characteristics, the designer must trade off the complexity of the model and the difficulty of the search (because a more complex model will tend to require more time to evaluate) with the requisite detail of the predictions.

### 5.2.8 Compound Organizations

A compound organization is one that incorporates elements from two or more different organizational paradigms. Examples of such structures have already been discussed in detail – both of the complete organizational models given in earlier chapters fall into this category. For example, the sectors of the distributed sensor network organization act like federations, because they are divided into groups with a distinguished agent (the sector manager) serving as an intermediary in some interactions. In that same structure there are elements of a matrix organization, because individual sensors can report to multiple track managers at the same time. In the information retrieval model it is easy to see the hierarchies that exist, with databases at the leaf nodes and aggregators as the nodes above them. The behavior and status of information retrieval mediators, on the other hand, makes each hierarchy seem like a federation because they act as intermediaries between the users and the information sources below them.

The ability to create such complex structures is one of ODML's key strengths and distinguishing characteristics. As shown in Chapter 6, most existing organizational design representations and techniques are limited to structures that have a particular characteristic shape. ODML makes very few assumptions about organizational structure, and therefore avoids this problem. The preceding sections have demonstrated how a wide range of organizational paradigms can be modeled using ODML. Cre-

ating a compound organization from disparate elements is accomplished in precisely the same way, by selecting or creating an set of components and linking them with expressions or node relationships that allow the behaviors of the organization to be appropriately predicted.

## **5.3 Conclusions**

Where previous chapters have taken a depth-first approach to explaining the benefits of ODML and the details of a small number of models, this chapter has attempted to provide some breadth to that argument, by discussing at a relatively high-level how a range of topics could be addressed. Incorporating concepts such as time, heterogeneity and abstraction were shown to be possible. Key characteristics from a number of commonly used organizational paradigms were also shown to be representable. When included with those more detailed results from Chapters 2 and 3, this provides convincing evidence that ODML's approach to allow flexibility while retaining quantitative detail can be a successful strategy.

## CHAPTER 6

### RELATED WORK

The techniques described in this paper intersect with several different research areas. This section describes prior related work done on organization representations and design techniques.

#### 6.1 Organizational Representations

As mentioned when ODML was first introduced in Section 2.2.1, there have been many different organizational representations produced by different researchers. One of two different approaches have been typically used to encode organizational knowledge. The first is to use formal logic primitives that define the capabilities and interactions between agents as a set of rules [59, 45, 50, 185]. One can then use theorem-proving techniques to deduce higher-level information from the primitive statements. These approaches are almost universally qualitative in design. The second is to use a mesh or decomposition tree, where nodes represent roles, goals, agents, or other organizational entities, and edges represent interactions between them [40, 143, 88, 174]. This additional structure helps focus analysis, assuming the structure is arranged in a logical fashion. This approach, which is used by ODML, is more easily integrated with quantitative information. In this section I will describe a range of representations that use both styles, and compare their strengths and weaknesses with those of ODML.

Before progressing, I will discuss ODML's applicability to non-computational domains. Although the goals I have outlined in this paper certainly have relevance to human organizations, the use of concrete, formal models to describe human and businesses activities has historically had mixed results. Formal models of human interactions frequently break down because human behavior can be hard to quantify and predict, at least at the individual level [124, 7]. Statistical descriptions can help address this, but this inherently raises the level of abstraction and limits the direct applicability of such models. Agents, being inherently computational and relatively predictable, don't suffer as much from this problem, and are therefore more amenable to the quantitative, expression-based technique used by ODML. So, while ODML could in theory be used to model human organizations, I believe the unpredictability and imprecision involved with quantitatively describing human activities would make it difficult to fully exploit ODML's strengths.

Although I will focus on general purpose, domain-independent organizational representations in the remainder of this section, it is worth mentioning the successful use

of domain-specific frameworks to capture particular relevant organizational characteristics. For example, Shen et al. [166] use a formal model to uncover a relationship between the environment, the level of cooperation exhibited by the agents, and the performance of the system as a whole. Decker and Lesser [39] motivate the need for meta-level communication to guide dynamic organizational adaptation when conditions are uncertain. Sen and Durfee [163] have created a model predicting the effects of various meeting scheduling heuristics, so that agents may dynamically adapt their behavior to correctly trade off local needs versus those of the larger organization. Malone, et al. [123], Gnanasambandam, et al. [68] and Schmitt and Roedig [160] have all used techniques from queuing or network theory to model organizational aspects of distributed or agent systems. In each of these works, a quantitative model was used to describe and predict particular organizationally-affected characteristics in much the same way that ODML is used. However, none of them do so in a manner that permits the linking of all relevant characteristics into a single, domain-independent model that is amenable to search. Therefore, while these approaches are individually quite useful, and in several cases have been incorporated successfully into ODML models, they do not address the same class of problems as that representable in ODML.

Tambe's STEAM framework [187, 186] is a general-purpose framework that uses a decomposition tree to encode organizational information. In fact, STEAM uses two such hierarchies. The first is a team organization hierarchy, which defines the possible teams, sub-teams and roles that may exist. Roles may be either persistent, existing for the lifetime of the organization, or task-specific, based on the needs of the current task. The second is the team activity hierarchy, or *operator hierarchy*, which combines aspects of goal decomposition with team concepts to describe how goals may be addressed from an organizational point of view. Operators can be team or individual-based. Individual operators represent activities to be performed by a single agent, while team operators are performed by multiple agents acting cooperatively. More specifically, the team operators express the team's joint intentions, which incorporates semantics governing the agents individual behaviors as members of that team and their desire to maintain a consistent mutual belief. Both types of operators include precondition, application and termination rules which determine their applicability, although detailed quantitative information is not present. The roles from the organization hierarchy assigned to agents or sub-teams dictate what operators in the activity hierarchy may be used. For example, if the role a particular agent is currently performing is in conflict with a particular operator's requirements, then that operator may not be used.

Section 5.2.7 showed how ODML could be used to capture some aspects of team creation and performance. Like STEAM, it could reason about goal decomposition, team composition and assignment of roles to agents. Unlike STEAM, the information it uses to do so was much more quantitative, which allows it to capture and predict more aspects of the agent and team level performance. It should be noted that although STEAM incorporates an organizational representation, its contributions are derived more from the well-defined team level semantics that agents operating within the framework exhibit. Although decisions made in an ODML model could parame-

terize such behavior, the model itself does not and was not designed provide this type of capability.

The TÆMS task description language, created by Decker, Lesser, et al. [40, 83], can also be used to describe organizational structure. At its core, it is a goal decomposition tree, where high level goals are subdivided into subgoals (tasks), which eventually terminate in executable, primitive actions (methods). What distinguishes TÆMS from other, similar languages is its explicit description of quantifiable aspects of the task. For example, each method is labeled with its expected quality, cost and duration. Because these values are stored as distributions, they also explicitly describe the uncertainty related to those metrics. The manner in which tasks may be achieved by their subtasks are also described, using so-called quality accumulation functions (QAFs). A QAF models how the quality of subtasks should be combined to produce the quality of the parent task, which can then be used to describe the alternative ways that task may be achieved. For example, a *min* QAF, denoting the quality would be that of the minimum quality subtask, indicates all subtasks must be completed successfully. A *sum* QAF says that the quality would be the sum of the subtasks' qualities, so better quality can be achieved with more work. TÆMS also incorporates the concept of interrelationships, which allows one task or method to affect the characteristics of another. An *enables* interrelationship, for example, indicates that one method must be successful for the target method to be achievable. A *hinders* interrelationship is weaker, indicating that if the source method is completed it will limit the target in some way, but not prevent it outright. A second set of interrelationships describes the effects method have on resources, and vice versa.

TÆMS may be used in an organizational manner because different parts of the structure may be performed by different agents [136]. In fact, the global structure typically does not exist in a complete form anywhere in the environment. Instead, individual agents possess a subcomponent (or subtree) that logically connects with other agents' structures. The manner in which those components are connected represents the manner in which those agents can be organizationally related. For example, if an agent has a TÆMS description of a high-level goal  $T$  and how it decomposes into two subtasks  $T_1, T_2$ ,  $T_1$  and  $T_2$  may be high-level goals in task structures owned by other agents. If it contracts with those two agents to perform  $T_1$  and  $T_2$ , they then have an obligation to report their results upon completion. Interrelationships can also represent organizational knowledge. If an *enables* interrelationship spans the structures owned by two agents, then it may be in their best interest to coordinate their activities. If the source agent notifies the target when the action has (or is intended to be) completed, the target can then be aware of when its action is achievable. These type of interactions were the basis of the GPGP framework [109], which first discovered structural elements that might require interaction between agents, and then employed one of several protocols to perform the coordination.

It was shown in Section 5.2.7 how ODML is capable of modeling similar task decompositions, and relating them to the abilities of individual agents. Aspects such as *hinders* and *enablement* can be modeled with modifiers that use the expected quality of the source node to affect the characteristics of the target node. Unlike TÆMS, ODML is not restricted to modeling a limited set of such characteristics, and

is therefore capable of capturing more aspects of the system at a greater level of detail. This capability does not come without a cost, however, as the more regular TÆMS structure is much more amenable to conventional search and analysis techniques.

Later work by Abdallah et al. [1] extended TÆMS by incorporating more explicit aspects of teamwork, specifically templates and achievability conditions. Templates are generalizations of TÆMS nodes, such as methods or tasks, which can be applied to an entire team rather than a single agent. This allows team-based structures to be represented with a single (or constant number) node, instead of requiring a separate node for each agent. This layer of abstraction does limit the expressiveness of the structure, but the complexity benefits are acceptable and appropriate in large team-based environment where agent activities are likely to be more homogeneous. In a conventional TÆMS structure, success is defined simply as when a method or task obtains non-zero quality. Achievability (and unachievability) conditions in this work provide a more sophisticated way to represent success, using formal boolean expressions that combine monitoring terms over the standard quantifiable characteristics (e.g., quality, cost, deadlines, etc.). These are also applicable to single-agent situations. These enhancements bring the capabilities of TÆMS closer to those of ODML, although the representation is still limited in its ability to model characteristics other than those that are explicitly embedded in the language.

The EFIGE [143] language is also intended to encode organizational structures. More so than STEAM and TÆMS, its focus is on describing classes of organizations, rather than particular instances. Organizations described in EFIGE consist of a set of parameterized templates which are populated with runtime knowledge to produce an organization. The primary EFIGE structures are hierarchical and recursive. At any level, the node may be denoted individual (terminal), or composite (recursive). A composite node is then decomposed into some number of sub-nodes. Composite nodes do not have physical counterparts within the organization, instead they are logical constructions which allow a group to be treated as a single, atomic entity. This facilitates describing characteristics that may exist in the organization, but are not manifested by any single individual. ODML shares this ability (Section 4.3.2), because it also does not enforce a mapping from every organizational structure to a tangible entity in the running system, and because organizational characteristics can be easily consolidated in one location. The *sector* node in the DSN model is such an example. Each node in EFIGE can take a set of parameters, which are used along with a separate, predefined set of preconditions to determine how the organization will be instantiated. This process is also affected by a set of constraints, which either limit the scope of potential instantiations or provide a partial-ordering (preference) among them. ODML differs from EFIGE in its ability to incorporate more general information about the organizational constraints, expected behaviors and working environment. For example, EFIGE lacks the ability to deduce a single-valued *utility* ranking for organizational alternatives - it only seeks to satisfy the set of constraints that are specified. Relatedly, there is no means by which environmental or goal characteristics may be encoded, which are a prerequisite to calculating a utility metric. Because of this, EFIGE is a more descriptive than proscriptive, making it less useful from a design point of view.



In MaSE [45, 46], DeLoach and Matson present a more free-form organizational modeling technique, based on the role interrelationships that occur in the organization. A role model is first created, derived from the high level goals and functional requirements of the system. Roles are connected in a graph structure, where directed edges represent communication acts. This model is given further detail by incorporating the (high-level) tasks the individual roles must perform, and depicting the agent interactions that will be performed as part of those tasks. The tasks are further refined in a separate concurrent task diagram, which models both intra and inter-agent interactions. The characteristics of the organization may then be more concretely specified in an organization model, which consists of a set of first-order logic rules that place constraints on agent behaviors and interactions. The rules make use of the previously defined role model, along with ontological information, to construct a set of axioms or social laws which govern how the organization may be instantiated. These aspects describe the range of organizational decisions that can be made, and a separate organizational state captures a particular instance of those decisions. Like ODML, the validity of a MaSE instance is determined first by its ability to satisfy the indicated logical constraints. It also has the ability to determine the quality of a particular instance, which can be used to differentiate alternative valid structures, but this metric is calculated using a relatively simple *capability* metric associated with each role-agent pair. The strict nature of the language also precludes its use in domains or models that lack explicit notions of roles and goals, although this same restriction makes the representation more tractable than ODML from a design perspective.

Unlike most of the other representations outlined here, the SADDE framework [170, 171] employs an explicit set of equations designed to capture aspects of the agent system. SADDE's Equation Based Model (EBM) describes the desired global behaviors of a system through a series of interrelated expressions. Using the EBM as a set of guidelines, one then constructs an Electronic Institution Model (EIM). The EIM is essentially an organizational design, intended to enforce the characteristics initially described in the EBM through a series of social interaction norms. A third Agent-Based Model (ABM) describes the behavior of individual agents. The veracity of these models may be determined by building and testing an agent system based on the specifications they embody. The values for any variable elements, such as the number of agents or the settings for specific variables, are set and evaluated in these tests. In particular, they propose a genetic algorithm for exploring the space of these particular values using the agent system.

ODML differs from SADDE in the pervasiveness of quantitative expressions used in the ODML model. For example, the equations in SADDE do not contain references to individual agents or their characteristics. Instead, they refer to environmental conditions, or averages of observable group behavior. ODML also differs in that it combines the expressions and organizational structure into a single, unified view. This tighter coupling allows the quantitative global and local ramifications of structural differences to be directly determined, rather than through experimental observation. For example, it is possible in ODML to determine the exact organizational ramification of a particular relationship. In SADDE, the equation based model is more

of a target for the organization model to emulate than a predictive description of how the organization will behave. Because there is no direction correlation between the structure and equations, the impact of the particular relationship could only be derived through additional experimentation.

The OMNI model created by Dignum et al. [50] is composed of three distinct levels: abstract, concrete and implementation. The abstract level defines the high-level objectives and values of the organization, along with the context in which it is expected to exist. The concrete level codifies the needs represented in the abstract level into a set of norms, rules and roles intended to satisfy those needs. These structures are then instantiated at the implementation level, where these roles and norms are applied to actual agents. OMNI further differentiates each of these levels into normative, organizational and ontological dimensions. As one descends through levels, these aspects are defined and grow progressively more refined. For example, the organizational dimension of the concrete level contains the social structure that is to be used, including roles, groups and role dependencies. It also contains the interaction structure, which defines the manner and pattern of interactions among roles. The normative dimension of the concrete level defines the specific norms and rules which govern agent behaviors using formal logic rules. These are somewhat analogous to hard constraints in ODML. The ontological dimension in OMNI differentiates it from most of the other representations presented here, by facilitating the identification of shared concepts among organizations. The structure of an OMNI model can be quite complex, and it is capable of explicitly describing a wider range of organizational features than ODML. It does so only in a qualitative manner, however. It is not possible to use an OMNI model to make numeric predictions about the system's behavior, and because of this it seems less well suited to design applications.

Fox et al. [59] describe the enterprise ontology model used in the TOVE project, which uses first-order logic to describe organizational structure. The basic structure used by the framework is called an object, which are organized into taxonomies. The taxonomy can contain a diverse collection of entities, including organizational goals, divisions, agents, roles, teams, activities and resources, among other things. Each individual object, as well as the relations it has with other objects, is defined with first-order logic. Together these rules present a working model of many aspects of the organization, linking abstract concepts such as goals and skills with more tangible objects such as agents and activities. A situational calculus is also used to allow these objects to exist and change over time, an important characteristic needed to project and hypothesize about the consequences of actions. The rules also allow one to make discrete queries to an organization, such as "who has role  $X$ ?" or "what behaviors may agent  $A$  exhibit?". These formalisms do presume the existence of a theorem proving engine capable of reasoning about the collection of predicates, this work uses Prolog. Like OMNI, although TOVE can represent many different concepts, the lack of quantitative information limits the representation's utility.

The  $MOISE^+$  model described by Hübner et al. [88] defines organizations along three separate dimensions: structural, functional and deontic. The structural specification is itself broken down into sets of roles, relations and groups that define the individual, social and collective characteristics of the structure, respectively. Roles

in this framework are associated with a set of constraints that relate them to other roles, and to the responsibilities that role has within the set of global plans. The set of relations describe interactions between entities in the organization, including communication acts, authority and inheritance. The structural specification is completed with a set of groups, which control how roles may be combined to form coherent clusters of agents. A group specification may include relations between group members and between other groups, role compatibility lists, and cardinality constraints. The functional component of a  $\text{MOISE}^+$  model consists of a goal decomposition tree. Goals within the tree may have different satisfaction conditions, specifying for example that its sub-goals must be completed sequentially or in parallel, or that a choice of only one sub-goal should be performed. This is similar to the task structures used in TÆMS, although quantitative details are not available. The deontic specification, which concerns the duties and responsibilities associated with the organization, ties the otherwise independent structural and functional components together. Through this structure, structural roles may be permitted or obligated to satisfy functional goals. In this way, agents assigned a particular role may first determine their responsibilities from the relevant deontic definitions, and then use the goal tree from the functional specification to construct a viable plan. This representation is also largely qualitative.

Malone, et al. [122] describe their experiences developing a process handbook, which attempts to represent the array of techniques used to form and organize business processes. This knowledge base can then help to design and redesign organizations, as well as facilitate the sharing of organizational knowledge. The representation they employ draws from two existing fields: object-oriented programming and coordination theory. From the former, they employ concepts of specialization and inheritance, which allows one to reuse knowledge from an existing structure while specifying how the new structure is different. A group of specializations of the same concept are further organized into *bundles*, which allows for direct comparisons among alternatives and facilitates the process of selecting a technique. This is loosely related to ODML's inheritance relationships, which can be used to the same effect (Section 5.2). These comparisons are explicit, stored in a table, and may be qualitatively and quantitatively ranked across multiple dimensions. Arbitrary quantitative data can be attached to organizational objects, but no mechanisms like ODML's expressions exist to link or relate that data between objects. The process handbook draws upon coordination theory to model the notion of interconnectedness between processes. For example, activities may share a resource between them, or information may flow from one site to the next. Each such instance represents a point where coordination may be necessary. To address this need, the handbook includes a library of generic coordination strategies.

Sims et al. [174] have created an organizational representation that is used within an automated organizational design process. This model separates domain and coordination knowledge, so that existing coordination templates may be reused by applying them to new domain information. The model itself consists of four distinct components. The first consists of a set of performance requirements and an environmental model. Both contain a set of arbitrary feature-value pairs which describe

a particular desired or existing characteristic. The task environment is represented with a goal decomposition tree that describes how goals are related to sub-goals, what responsibilities are needed to satisfy those goals, and where sub-goal interactions must take place. The third component is a set of role descriptions. Each role is defined by the responsibilities it can perform, how well it performs, what it requires and how it may be distributed across a set of agents. The last component consists of a set of agent descriptions. Each description contains a set of arbitrary features and capabilities, along with a set of role satisfaction tuples. Each tuple specifies a role the agent may take on, how much that role drains local resources, and how many messages it will produce. The search process works by first determining which roles can satisfy which goals, and then by finding agents which can satisfy those roles. The agents can themselves be related through higher level organizational structures that affect their performance characteristics.

The principal difference between this representation and ODML is the amount of flexibility. Sims's structure has a well-defined format that facilitates construction and manipulation. ODML's structure is generally less rigidly-defined, which adds both flexibility and complexity. Despite this, the class of organizations that are representable by the two formats seems to be similar. For example, both can model agents, roles, capabilities and constraints. They can also both represent potential role-goal and role-agent bindings. In Sims's structure this information is captured in lists associated with roles, goals and agents. In ODML this information is captured within nodes by using modifiers, is-a and has-a relationships. The former strategy permits a more straightforward search process, while the latter can capture a wider and potentially more abstract range of relationships and effects. The utility of an instance in Sims's representation is deduced by an external evaluation component, which uses domain and model-specific heuristics to predict organizational characteristics. This is in contrast to ODML, where the information necessary to deduce utility is defined within the organizational structure itself.

None of the approaches outlined above meet all the representational needs outlined in Section 2.2.1. The TOVE enterprise, STEAM and MOISE<sup>+</sup> models are not sufficiently quantitative, while the process handbook is not a computational model. The EFIGE and TÆMS frameworks are both quantitative and computational, but the former lacks the level of detail needed to serve as a deductive model, while the latter does not have the structural primitives needed to represent and evaluate a wide range of organizations. The SADDE framework is probably the closest structural relative of ODML, but it does not have a way of explicitly modeling fine grained detail or comparing organizational variants without employing a simulation. Sims's framework shares many of the same capabilities as ODML, but does so using a less flexible model that lacks the innate ability to evaluate organizations. A summary of key differences between these representational styles is shown in Table 6.1. As one can see, there are many structures capable of representing the alternative ways to organize a system. Fewer model detailed quantitative information, and none other than ODML can represent both aspects in a way that captures the integrated, interdependent nature of organizations. It is this key capability which differentiates ODML from existing organizational representations.

**Table 6.1.** A comparison of the characteristics and capabilities of several different organizational representations.

Framework	Organizational Alternatives	Organizational Knowledge	Heterogenous Objectives	Explicit Task Structure	Current Structure	Quantitative Relations	Deductive Capability	Hard Constraints	Soft Constraints
ODML	✓	✓	✓		✓	✓	✓	✓	✓
STEAM	✓	✓	✓	✓				✓	
TAEMS	✓			✓	✓		✓	✓	✓
EFIGE	✓	✓						✓	
MaSE	✓	✓	✓	✓	✓			✓	
TOVE	✓	✓	✓	✓			✓	✓	
Process Handbook	✓	✓	✓	✓	✓	—			
$\mathcal{M}$ OISE <sup>+</sup>	✓	✓	✓	✓	✓			✓	
SADDE		✓	✓		✓	✓		✓	✓
Sims	✓	✓	✓	✓	✓		✓	✓	✓
OMNI		✓	✓		✓				

## 6.2 Organizational Evaluation and Design

In Section 1.2 I claimed that nearly all multi-agent systems have an organization, because any agent system will have a pattern of interactions and an assignment of responsibilities, and these characteristics by themselves constitute an organizational design. Viewed in this light, there has been an enormous amount of research into the process of dynamic organizational search and design, because many of these systems decide such organizational characteristics dynamically. For example, the simple process of contracting out or assigning a task creates a manager-worker interaction [178]. The related work presented below is differentiated from the majority of these projects in that the search process in each is grounded in an explicit representation of the organizational space. Where the others are superficially related because they implicitly result in organizational decisions, those below are more directly related because they address the problem of automated organizational design in a direct and (mostly) general fashion.

Before focusing on particular design methodologies, it is worth mentioning the relationship that organizational evaluation has with design. A key characteristic of any domain-independent design process is the ability to predictively evaluate how well competing designs will behave under expected conditions. This can be accomplished in various ways, including the use of heuristic analysis, predictive models and simulation. Like the technique-specific models described at the beginning of Section 6.1, ODML uses predictive mathematical models for its evaluation. The benefit this

approach has over a simulation-based evaluation is speed. A mathematical model's evaluation type is typically measured in seconds – it requires only as much time as needed to evaluate the equations. Conversely, a true simulation can require minutes, hours or even days to determine characteristics for a single design consisting of a large number of entities. Heuristic approaches are less easily categorized, as any sort of approach (case based, analytic, simulation, or other) might possibly be embedded in the process. In practice the drawback to using heuristics is their typical lack of generality – a new battery of tests may needed to be devised as the organizational alternatives change. i

As with organization representations, each of these techniques has its place. The utility of each depends on the user's relative need for detail, optimality, speed or generality, among other things. The analytic approach was selected for ODML because a design process that may potentially consider all of the (potentially vast number of) organizational design alternatives can greatly benefit from its speed and accuracy. Although not demonstrated in this work, it is also quite possible to use a hybrid scheme that exploited the strengths and avoided the weaknesses of each. In this case, one might use the analytic approach to form a rough view of the organizational space, and use detailed simulation to empirically compare a small subset of representative instances.

The SADDE design methodology [171] discussed previously can be thought of as an approximation of this technique. Each agent in SADDE expects a set of parameters, each of which controls some aspect of the agent's behavior. The search process thus attempts to find an appropriate set of values to assign to organizational parameters for all participants. In this particular work, agents act as consumers and producers in an electricity market. A typical parameter dictates how a producer should generate its electricity, while another specifies the strategy consumers should use when placing bids. This strategy allows local agent characteristics to be modified, and consequently affect how they interact with other agents, but changes to the higher-level organizational structure are not considered without human intervention. Iterative evolutionary computing is used to explore the space of parameters. Each agents' parameter set comprises a single gene, and the complete set of genes makes up a chromosome. During the search, a particular chromosome is used to instantiate a multi-agent system, which is then evaluated with a fitness function to determine the utility of the organization. Biologically-inspired notions of mutation and genetic crossover among chromosomes, weighted by the results of the fitness evaluation, are used to create new organization candidates. These candidates are then ranked and evolved in an iterative manner by comparing them against the equation based model (EBM), and the most fit organization is chosen.

SADDE is only an approximation of the hybrid scheme because of its loose connection between the EBM and the agent organization. As mentioned earlier, the EBM is a descriptive target, not a predictive model, and the genetic manipulations are used to coerce the agents into a design that conforms to those equations. In ODML, it is the modeling process itself that ensures the consistency between the form and function of the design, which is what allows the utility expression to directly guide the search process.

Like ODML, So and Durfee [180, 179] search for structures that are a best match between organization and environment using closed-form expressions to model the various relevant performance metrics. The search process makes use of an organizational model, a task environment model, and a performance model. The organizational model defines elements such as the available agents, and the way tasks and work flows are assigned among them. The particular organizations considered are limited to various forms of hierarchies, corresponding to a functionally decomposable task. The task environment model describes the task itself, along with any environmental characteristics that might affect the organization’s performance. This detailed view coincides with my own, that all relevant features of the domain must be considered when forming the organization. The performance model consists of a number of metrics by which the organization may be judged, such as response time, throughput, reliability, etc. The particular metrics which are listed are all global in character – per-agent analysis is not discussed. The search strategy employed in this work is broken down into four phases: monitor, design, evaluate and implement. Like ODML, evaluation is accomplished by using closed-form formulas to predict various performance metrics. Unlike ODML, these concepts were not formalized into a general organizational representation language suitable for defining a search space, and only hierarchical systems are considered.

Although the primary function of the STEAM framework outlined above is to enforce teamwork semantics at runtime, it also incorporates some adaptation facilities through the specification of monitoring points and re-planning [185]. A set of logical rules, called role-monitoring constraints, may be defined to monitor team performance. At runtime, the validity of these statements may be ascertained through plan recognition or explicit communication among participants. If a constraint is found to be violated, and that failed constraint implies that a team goal is no longer achievable, a repair mechanism may be invoked to handle the failure. Like the adaptation process described in Section 4.5, this repair process can be viewed as a form of organizational design. The repair mechanism itself is just another team operator, which allows the existing organizational machinery to perform the repair task correctly. Repair techniques are generally limited to role reallocation. A suitable, non-conflicting participant is found to satisfy the failed role, and the substitution is then announced to the other team members. Although this approach lacks the quantitative grounding available in ODML and is limited to role-based repairs, the runtime capabilities inherent in STEAM provide a level of automation absent in an ODML-based implementation. Thus, while the level of design and adaptation is more limited, the process itself is more straightforward to enact. This is a key difference between STEAM, which has robust runtime semantics and behaviors and a less detailed representation, and ODML, which reverses this tradeoff.

Nair et al. offer a organizational adaptation technique based on a formal model called a RMTDP (Role-based Multiagent Team Decision Problem) [138], based on a distributed POMDP framework. The goal of an RMTDP is to provide a way to create and adapt role-agent matchings. Like ODML, the space of possible assignments is combinatoric, so heuristic search techniques must be used if assignments are needed in a timely manner. Also like ODML, the RMTDP model is used for both creation

(team formation) and adaptation (team reformation). The model itself consists of a set of states, actions, transition probabilities, observations, observation probabilities, a reward function and a set of roles. Actions and rewards are divided into role-taking and role-executing types, where the former reflects an agent taking on a role and the latter the actions agents can perform in that role. The action timeline is also divided, so that each role-taking epoch is followed by a role-executing one. It is pointed out that although this limits the representation, it also facilitates the cost-assignment problem.

Initial role allocation is accomplished by associating an RMTDP policy for each possible assignment of agents to roles. This is in contrast to ODML, where a single template represents the entire organizational space. In order to avoid searching the entire space of assignments, a branch-and-bound pruning approach is used. The maximum value of each child in the search tree is estimated using the system's plan hierarchy, and if it is not possible gain utility by pursuing a child, it is pruned from the tree. RMTDPs can also be used for adaptation, although like STEAM it is limited to role-based reallocation strategies. However, it was demonstrated that using an RMTDP-based policy along with STEAM's normal interpreter-based repair results in behaviors that more reliably accounts for cost and the probability of failure. RMTDP and ODML also differ in their ability to relate organizational decisions to tangible performance characteristics. Like ODML, the policy derived from a RMTDP may be based on a range of characteristics. Unlike ODML, the quantitative rationale for or explanation of a particular decision is lost in the policy's transition function. This can make an RMTDP policy difficult to use when runtime conditions do not match those the policy was derived under, while an ODML structure would directly propagate such changes through the existing model.

The MaSE framework [127, 46] uses capability functions to evaluate and rank different organizational decisions. For example, a role capability function is used to determine how well a particular role can satisfy a particular need. Similarly, an organizational assignment function computes the capability score for the organization as a whole. The design process works by finding a set of agents that have the capabilities needed by the roles that have been selected to meet the high-level goals. Although details are not given describing how this search is performed, the process seems conceptually similar to the role assignment problem discussed earlier, with a well specified way to evaluate role fitness. Although the design process used by MaSE is based upon quantitative data, it is driven by just the singular quality value that may not be sufficient to capture the complex tradeoffs different designs can have along different dimensions. ODML can explicitly represent and compare among an arbitrary set of quantitative features, and therefore can reason about tradeoffs that MaSE cannot.

Hübner's *MOISE*<sup>+</sup> model has also been used as the basis for controlled, top-down organizational adaptation [89]. Similar to the theoretical ODML-based process outlined in Section 4.5, this process is separated into four phases: monitoring, design, selection and implementation. Monitoring determines if the current organization no longer satisfies the needs of the global purpose. In the design phase a set of candidate, alternative organizations are created which address the deficiency. The selection



phase chooses one of these alternatives, which is then implemented. This must be done according to a utility calculation where the benefits of the new organization are balanced against the costs of implementing it. The authors envision this process performed as a normal task within the system, and can therefore exploit organizational concepts in doing so. Their proposed strategy would employ roles such as a reorganization manager, monitors, a designer and selector. General descriptions of how those tasks would be performed are not given, and thus are difficult to compare against ODML's techniques. In a case study that is provided, several designers operating concurrently use a range of domain-specific techniques to explore the organizational space. A single selector then ranks and chooses the most appropriate structure. It seems quite possible that ODML or an ODML-like process could coexist within this set of designing agents.

The organizational work by Sims et al. also incorporates an automatic organizational design feature [174, 173]. This more general technique was inspired by their earlier design-specific work on bottom-up coalition formation [175]. As indicated in the previous section, the representation used by the design process is composed of a number of separate characteristics, including a goal tree, a set of role specifications, and a description of the available agents. The principal design process revolves around finding a set of role-goal and role-goal-agent bindings. It must first find a suitable set of roles that meet the requirements laid out in the goal tree and elsewhere, and then find a set of agents able to provide the capabilities specified by the roles. The search process uses a set of heuristic techniques to find and verify these bindings. As part of that process, it may be decided that a role should be distributed among a set of agents, to satisfy computational needs, address spatial requirements or improve quality in some other way. This then creates the need for coordination among agents in the set, which is itself a separate organizational design process applied to a newly recognized "coordination goal". Thus, the search is separated into two phases: the search for application-specific bindings (domain goals), and the search for coordination-domain bindings (coordination goals). Organizations are selected based on their ability to meet specified domain and role requirements. The current modeling framework lacks a direct means to quantitatively evaluate and rank otherwise valid organizations in a general way. It currently uses an external heuristic evaluation component to do so, which relies on model-specific rules and information to predict behaviors and guide the search.

A summary of these approaches is shown in Table 6.2. All of these provide suitable and reasonable mechanisms to enable organizational search. However, none of them address the range of organizational characteristics representable in ODML. Chapters 2, 4 and 5 demonstrated how one can use ODML to model concepts including agent interactions, role assignments, bounded resources, detailed performance predictions, and high-level organizational structures such as hierarchies, coalitions and teams at various levels of abstraction. This range encompasses and subsumes nearly all the characteristics considered by the approaches described in this section. Because of this, the scope of the design space using ODML can be richer in possibilities, and correspondingly harder to effectively search.

**Table 6.2.** A comparison of the characteristics and capabilities of several different organizational design and adaptation schemes.

Framework	Characteristics Considered	Search Technique	Centralized or Decentralized
ODML	Organizational structure	Generate and test	Both
SADDE	Agent parameters	Evolutionary computing	Centralized
So	Assignment and hierarchy	Generate and test	Centralized
STEAM	Role assignment	Rule matching	Decentralized
RMTDP	Role assignment	Branch and bound	Both
MaSE	Role, capability assignment	Role matching	Centralized
MOISE <sup>+</sup>	Various	Various	Decentralized
Sims	Organizational structure	Heuristic search	Centralized

## CHAPTER 7

### CONCLUSIONS

In this final chapter, I summarize the work presented in this dissertation, along with the major intellectual contributions that have arisen from the process. This is followed by a discussion of the conclusions that were formed and some of the high-level benefits and drawbacks of this approach. I conclude with a brief outline of future directions for this line of research.

#### 7.1 Summary

The primary objective of this work is to evaluate the potential benefits and feasibility of using a domain and technique-independent representation based on predictive quantitative information to design and evaluate multi-agent organizations. To motivate this problem, Section 2.1 demonstrates that organizations have a measurable impact on a variety of performance characteristics, and that different organizations will result in different performance. This is shown through experiments with an existing, organizationally-driven distributed sensor network application. Those experiments demonstrate that the organizational design had a measurable effect on a range of characteristics, including overall communication volume, how evenly communication and action loads were distributed, the relative frequency of different message types and the tracking quality exhibited by the system as a whole.

These experiments also show that certain members of that organization may locally benefit under some organizational conditions while the performance of other members degrades under those same conditions. It is only when the needs of both are taken into account that the global performance may be optimized. Together, these results also provide the necessary insight that the utility of an organizational design can depend on the complex interactions of the participants, the environment, the designer's expectations and the relationships and direction imparted by the organizational design itself. Being able to capture and predict the effects of such interactions in an essential first step towards determining which alternative design is most appropriate.

The remainder of Chapter 2 introduces ODML, an organizational design language that exhibits the quantitative and predictive character this dissertation seeks to explore. ODML possess a core set of features that allow it to address a range of problems, including:

- support for abstract entities called *nodes* that can be used to represent tangible and intangible components of the organization;

- the ability to precisely define characteristics of each node with mathematical expressions or a set of intrinsic functions;
- the ability to define node interaction and inheritance relationships;
- the ability to define organizational decision points;
- support for hard constraints and soft constraining effects;
- support for floating point, list and probabilistic distribution data types;
- support for analytic and Monte Carlo-based organizational evaluation.

These features enable it to capture a range of designs, particularly those that consist of a set of related entities that possess a set of interconnected, potentially constrained characteristics.

One of the key differentiating characteristics of this representation is that it strives to be both flexible and concrete. As shown in the comparison to related work in Chapter 6, existing representations achieve one aspect or the other, to varying degrees, but not both. For example, very flexible representations typically do so at the expense of the detail needed to make concrete predictions. Detailed representations make assumptions about the structure's form, function or characteristics that preclude their use across a wide range of applications. ODML's representation is built around arbitrary mathematical expressions and makes few assumptions about the model's contents, which allows it to capture a wide range of structures and interactions at a fine level of detail. An example of how these expressions exist and interrelate within a larger organizational structure is shown in Figure 2.13. Figures 2.9 and 2.14 show the algorithms used to deduce quantitative information from such a structure.

Section 2.2.2, Chapter 3 and Chapter 5 demonstrate ODML's capabilities by presenting a number of models of different organizations in different domains. They also show predictions made from these models, which are validated through empirical testing. Complete models are shown for the distributed sensor network (DSN) application mentioned above and for an information retrieval (IR) network. A number of additional examples in Section 5.2 show the breadth of ODML's applicability by capturing the common organizational paradigms surveyed in Appendix B. Each of these models shows how different characteristics may be defined and approximated, how existing mathematical modeling techniques may be integrated, and how the full set of interacting characteristics may be used to evaluate organizational utility. These are offered both as proof of ODML's representational ability and as examples of how its features can be used in practice.

For example, the DSN model from Section 2.2.2 shows how agents, roles, coalitions and organizational relationships may be represented. Interactions between individual agents, the environment and performance expectations are shown. The potentially negative consequences of multiple-role assignment are taken into account, as are the consequences of choices made dynamically in response to changing conditions. Section 2.3.8 demonstrates how ODML can be used to craft an appropriate model of utility. The validity of this model is evaluated by comparing the predictions it makes

with the experimental results mentioned above. In particular, Figure 2.17 and 2.18 in Section 2.3.9 show that the model accurately predicts both global and local characteristics, such as the expected RMS tracking error the system will exhibit and the communication and computational loads of individual agents. Figure 2.19 shows how these predictions hold as the environmental demands change.

Additional discussion in Section 5.1 shows how this same model can be enhanced to predict additional detail, including geographic heterogeneity and temporal interactions. In doing so, this demonstrates ODML's capability to capture information at multiple levels of abstraction, which assists in the design process by ensuring only relevant characteristics are present in the model. Organizations for larger systems can be made more tractable by omitting low-level information, while focused predictions can be made for smaller designs when necessary.

The IR model from Chapter 3 is a second example of ODML's application to a complex domain. That model shows how the effects of a quality-driven search process may be represented, as well as the queuing and potential bottleneck behavior of individuals taking part in the resulting work flow. The former characteristic, known as response recall, is predicted with an appropriate probabilistic function that models the search process performed by agents in the system. The latter, known as response time, is predicted using existing techniques from networked queuing systems and order statistics. This model also shows how very detailed information can be propagated through the organization, as in lieu of simple averages it predicts a complete probability density function of the response time for the organization. A simulation environment was used for validation experiments shown in Figures 3.4 and 3.8. As with the DSN model, a utility function for the IR system was devised. Figure 3.9 shows how this utility changes for different designs of the same organization as the environmental work load changes. This is a concrete example of the type of organizational space ODML can define and help evaluate, and also demonstrates how the appropriate design of an organization can change depending on the conditions in which it is used.

Chapter 4 continues this line of thought by showing how the organizational space defined by an ODML model can be searched in a variety of ways. The NEXP-complexity of the search is proved, demonstrating that it resides within a class of computationally very difficult problems. A set of algorithmic techniques are shown in Section 4.2 that help address this challenge, by avoiding parts of the space that can be determined to be invalid or redundant. In particular, Figures 4.7 and 4.8 show that the search space can be reduced by orders of magnitude without affecting optimality by intelligently pruning the space. Using the first technique, the monotonicity of certain constrained characteristics is exploited to avoid provably invalid alternatives. Experiments in the distributed sensor network domain, which contains such constrained monotonic trends, show the search time of a nine-sensor organization is reduced from two hours to two seconds with this technique. The second technique defines classes of equivalence that can be used to avoid effectively redundant design choices. For example, over 90% of the possible design alternatives in the nine-sensor DSN search are eliminated when this technique is used, with no loss of optimality. Ex-

periments with other models provided additional examples of when these techniques are and are not beneficial.

Although both of the above approaches are domain-independent, they do exploit characteristics that do not necessarily exist in all models. I have devised other approaches that can be used in situations where they are not applicable. For example, Section 4.3 shows how the model itself can be manipulated, by increasing homogeneity or abstractions to shrink the search space. Tables 4.1 and 4.2 show how models that incorporate such modifications can significantly reduce the number of alternatives that must be considered. By limiting choice within the model, which in turn makes the internal structure more homogeneous, entire branches of the search tree may be eliminated. Increasing the level of abstraction can have a similar effect, by removing large numbers of choices at the periphery of the search tree. Results from using these techniques in the information retrieval domain show that even small organizations can reduce the number of valid design alternatives from thousands to tens with modest reductions in utility.

If the algorithmic techniques are not applicable, and the designer does not wish to change the model itself, there are still other techniques that may be employed. Section 4.4.1 describes a value caching technique that helps avoid unnecessary computations that occur during the successive evaluations in the search process. The distributed search process from Section 4.4.2 attacks the search directly through parallel computation. Experimental results in Figure 4.17 show that ODML's search space can be effectively partitioned, allowing the distributed approach to achieve significant savings in time. Although the achievable performance gains from these two techniques are not as great as those shown above, they are attractive because they require little or no assistance from the designer and work to varying degrees under all conditions.

The output of this search process, regardless of what techniques are employed, is an organizational design that can then be used to guide the construction and behavior of a working system. Section 4.4.3 shows one way this can be accomplished.

Chapter 6 discusses how this approach differs from the representation and design approaches that have come before it. In particular, it shows how no other representation exhibits the quantitative yet flexible approach that ODML does, and contrasts the approaches others have used to address the automated organizational design problem.

## 7.2 Contributions

As part of this research, a number of tangible contributions have been made to the state of the art.

1. **I have developed a quantitatively-grounded representation, and shown that it is an effective way of capturing organizational characteristics.**

As shown in Section 6.1, many different organizational representation languages have been created by researchers in the past. None meet all the requirements first outlined in Section 2.2.1, particularly the need to incorporate the arbitrary

quantitative information I believe is required to correctly evaluate the utility of an organization situated in a complex, real-world environment. To demonstrate that it is possible and useful to do so, I presented the Organizational Design Modeling language (ODML) in Chapter 2.

ODML differs from these approaches in its capacity to directly model both the quantitative characteristics of an organization and the range of possible designs that can be created. Lacking this information, those earlier efforts may be able to describe the range of possible organizations, but they are generally not able to directly and computationally deduce *how* or *why* one design is better than another in a given context. In the few representations where it is possible to deduce some aspects of runtime performance, those predictions are either qualitative [59] or not tightly bound to organizational decisions [171, 40]. I believe ODML's ability to incorporate detailed, quantitative information, along with a concrete description of how organizational characteristics interact to produce local and global behaviors make it a fundamentally more capable representation.

**2. I have demonstrated that it is possible to use a quantitative language to effectively and accurately model and predict the complex, interrelated characteristics of real-world computational organizations.**

A representational language is only as useful as its ability to represent. Existing representations were shown to be either limited in their capacity to model quantitative details, or limited by their assumptions to model a wide range of organizational situations. ODML's intentionally primitive yet quantitative approach avoids these limitations.

This was demonstrated through the creation and validation of many different models. These include a complete model of a distributed sensor network architecture (Section 2.2.2), whose predictions were validated against the existing system it describes. Modeled characteristics include aggregate level features such as average RMS tracking error and communication disparity, as well as individual entity elements such as communication load, role frequency and sensor usage. A second complete model was generated for a peer-to-peer, distributed information retrieval network (Chapter 3), which was verified through empirical comparison to a simulation environment that implements the concepts. This model incorporates techniques from probability theory and queuing theory to predict the results of search and the probability density function of the organization's response time.

Section 5.1 also showed how organizational characteristics that lack explicit support in ODML, such as geographic heterogeneity, temporal interactions, and task decomposition, could also be approximated. As part of this set of modeling experiments, a survey of common multi-agent organizational paradigms was compiled (Appendix B), which is intended to be of interest to the broader MAS community.

- 3. I have demonstrated it is possible to use the same representation to describe the range of organizational alternatives, which defines a space of possibilities that may be searched.**

Chapter 6.1 and Section 4.2.3 showed how simple mathematical optimization procedures fail to capture the range of possibilities that are typically considered as part of an organizational design process. ODML's variable fields capture the capabilities of these mathematical representations, and its has-a relationships allow one to define many alternatives that they do not capture. Section 3.3.6 showed explicit examples of some of the different organizational designs that can be derived from the information retrieval model, for example.

In addition to the two complete models that were provided that employ these features, Section 5.2 demonstrates this ability to represent a range of situations by applying ODML to a number of the organizational paradigms commonly used in multi-agent systems. Section 4.1 uses the information retrieval domain to show how these choices map to a space of organizational alternatives. Sections 4.3.2 and 5.1.3 further showed that this representation can capture organizational concepts at different levels of abstraction.

- 4. I have demonstrated it is possible to use the embedded quantitative information to evaluate and rank organizational instances as part of an automated organizational design process.**

The quantitative expressions embedded within an ODML structure have the capability to model a range of complex and interdependent organizational behaviors. By using those expressions, along with a suitable description of the expected operational context, the model is able to predict the characteristics the organization will exhibit when instantiated and used in that context. These quantitative predictions can then support a validity evaluation, by checking the satisfaction of hard constraints that exist in the model. When combined through a suitable expression, these predictions also support a utility-based calculation, with which candidate organizations can be directly compared.

Existing representations either lack the ability to automatically compare organizations [122, 50, 89], or rely on a separate analysis or simulation component to do so [171, 174, 46]. More uncertain environments, or environments that are known to be more dynamic can be also evaluated with ODML's built-in hierarchical Monte Carlo evaluation process, described in Section 2.2.3.3. The results from these evaluation techniques were verified for the two most complete models through empirical comparison to tests in simulation environments.

The principal reason to employ a quantitative, predictive model such as ODML is its ability to support an automated organizational design process. This work has shown that it is possible to embed a range of organizational alternatives within the model itself, and that these alternatives can be viewed as a space of designs that can be searched. Several techniques to search through the space embodied by an ODML template, including algorithmic techniques to prune the



search space of invalid, equivalent and redundant instances (Sections 4.2.1 and 4.2.2), modeling techniques that employ abstraction and heterogeneity (Sections 4.3.2 and 4.3.1), the ability to use general mathematical solvers as part of the search (Section 4.2.3), and the ability to distribute the search across several machines (Section 4.4.2). Although the utility of each of these approaches changes under different circumstances, it was shown that a suitably informed model designer can create models that are able to exploit these techniques. The observable benefit of employing these search techniques, in the form of a reduced search space or a shorter search time, was shown through empirical tests.

## 7.3 Discussion

In addition to the contributions listed above, I have also reached several additional conclusions that merit discussion.

1. *Creating models can be difficult, but if successful the model is capable of evaluating designs much more rapidly than a prototype or simulation.*

Both the DSN and IR models required a fair amount of domain and modeling expertise to create. Techniques were explored, revised and sometimes abandoned during this process, but ultimately a useful, working artifact was produced. Although searching through the organizational space created by these models is difficult, the ability to quickly evaluate candidate designs makes it possible to compare many more alternatives that would be otherwise possible. Evaluating an ODML instance is orders of magnitude faster than running the trials needed to analyze a working system. This ability means that more designs can be searched in less time, which can correspondingly increase the quality of the final result.

2. *The value of a particular organizational design depends on a range of characteristics that can differ between designs, domains and system objectives.*

The empirical tests performed in Sections 2.1.2, 2.3 and 3.3 showed how the organization affects a range of different characteristics of the running system. By their nature, different designs will exhibit different behaviors, and the interaction of those behaviors with the environment and the system's stated objectives will ultimately determine the value of those designs. Because of this variety, there is no single, succinct set of detailed characteristics that can be used to determine the utility of a design in the general case. The availability of different resources, such as communication, computational or other physical devices, will change what behavioral effects are important. Domain-specific metrics such as RMS error or information recall may be critical in one domain but absent in another. The ability to explicitly represent the relevant characteristics and combine them into a single comparable utility value is a key component of any design evaluation process. This need to precisely encode a variable set of inter-related characteristics motivates the flexible "web of equations" approach used by ODML.

3. *Quantitative modeling is possible and effective, but not practical in all cases.*

The benefit that employing a simulation over a more mathematical representation such as ODML is that the simulated system may become arbitrarily complex. What may be a trivial change to a running program could in the worst case require a complete reformulation of the model that describes it, if a basic assumption is violated. A mathematical model is almost always much more compact than the system it describes. To arrive at the same result using less machinery, appropriate techniques must be found and correctly employed that capture the system's behavior. This is not always possible to do, because the predictive techniques have not been discovered or those that are known are insufficiently accurate. It is for this reason that I believe that, given enough time, a well-designed simulation environment can always be used to elicit more "truth" than an abstract model. The tradeoff, as the previous point alluded to, is that the running time or computational resources needed to do so may be so great as to make a methodical search of a large space impractical with this technique. Both approaches have their place, and can work well in harmony if simulation trials are used to analyze a set of solutions previously culled by a model-based approach.

4. *Allowing arbitrary mathematical expressions can give a great deal of flexibility to the organizational designer, but has the potential to create a correspondingly difficult search space.*

One of ODML's key advantages over existing representations is its ability to quantitatively capture a wide range of characteristics. It does so by embracing the complexity and diversity that I believe exists in the real world, through its lack of structural assumptions and support of arbitrary mathematical expressions. Most related representations have a set of supported organizational concepts, which generally interact through a predefined set of parameterized relationships. Although the latter systems have less flexibility, their ability to make strong assumptions about the contents of the organization facilitates the search process.

For example, if one assumes that agents take on roles and that in doing so some of the agent's local resources will be consumed, it is possible design a search technique around that fact. Because ODML does not make such assumptions, it can be difficult to create techniques that efficiently search the range of organizational spaces that can be encountered – one is limited to features that can be deduced from the underlying mathematics. For example, the constraint-based technique from Section 4.2.1 exploits the existence of constrained, monotonic characteristics. If such characteristics are not available, the technique cannot be used.

Although the search process is made more difficult, this dissertation demonstrates the potential benefits of making such a tradeoff. The ability to make concrete, quantitative predictions about the set of contextually relevant characteristics makes such a representation better equipped to judge utility than

those that lack such information or hide it within relatively coarse-grained abstractions.

5. *Existing mathematical techniques can play an important role in a model without being the entire model.*

Another key feature of ODML is its ability to merge disparate aspects of organizational behavior into a single model. For example, Chapter 3 showed how techniques from probability and queuing theory have been successfully integrated into the IR model. By themselves, these techniques only capture certain aspects of the system, despite the fact that they are intimately related. Only when the predictions from both are available can the performance and utility of the design be known.

While existing theoretical models of such systems frequently choose to focus on particular aspects of their performance, the overall behavior of a system of any realistic amount of complexity will not be so narrowly defined. As with the domains described in Sections 2.2.2 and 3.1, the behavior of most realistic applications will be dictated by a range of external and internal factors, each of which may be appropriately modeled with a technique different than the others. It is for this reason that I believe representations that are limited to a single, specific approach may be unable to capture all the relevant details of a realistic system. This will necessarily impact their ability to make accurate predictions, and degrade their utility in evaluating alternatives.

This co-mingling of techniques can give other benefits, by making the search process more efficient than it might be otherwise. For example, in the DSN model the relatively inexpensive constrained trend technique from Section 4.2.1 was used to prune the search space before the more expensive organizational utility calculation was performed. Therefore, although the trend of *utility* was not monotonic, the search as a whole could be guided by other aspects that were. This is an example where the potentially large and difficult space of one design aspect is assisted by the presence of others that are more tractable.

## 7.4 Future Directions

Many of the possible future directions from this work have been discussed in earlier sections. In this section I will summarize and expand on these and other issues that merit further attention.

1. *Develop techniques to adapt the organization at runtime.*

In the face of a changing environment, any static organization is vulnerable to contextual changes that can render it inefficient or ineffective. I believe the search space provided by the ODML model can be used to facilitate the adaptation or reorganization of the running system. The existing model can first be reused as the basis for a model-based diagnosis process that monitors for

and evaluates operational faults, by comparing predicted characteristics to those that are observed. Appropriate search strategies must then be developed to find organizational appropriate solutions in a timely manner. I feel ODML's ability to represent not just the space of adaptive solutions, but also the rationale behind observed symptoms and the consequences of organizational change allow it to support a more deliberate and directed search than is currently possible. Section 4.5 outlines a high-level approach that uses ODML to accomplish this.

2. *Develop an effective, incomplete search.*

The existing search techniques described in Chapter 4 are all integrated within an encompassing, exhaustive search. The benefit of this approach is that it is complete and optimal, but it can be a very expensive process if the space has sufficient size or complexity. When the size of the pruned search space remains intractable, an incomplete but informed search of the space can be used in an attempt to derive good but not necessarily optimal results. A heuristic approach could attempt to guide decision making based on locally observable characteristics, although the ramifications of a particular choice can be difficult to ascertain and it is not always obvious what characteristics are most relevant. Genetic, sampling or other stochastic approaches could be employed to avoid this by attempting to divine what the space of possibilities looks like (through their utility or other characteristics), and guide the search into promising areas.

3. *Develop a general approach to domain-specific search.*

As mentioned above, the space of possible organizations in an ODML template can be quite large, while the underlying mathematics do not necessarily facilitate navigating through this space. I have intentionally focused on domain-independent approaches in this dissertation, but in a practical, more restricted application it seems likely that domain-specific knowledge could significantly aid the search process. Section 4.2.6 outlines some ways this could be done.

4. *Experiment with bottom-up organization construction.*

As introduced in Section 4.4.2.2, a more emergent and bottom-up approach to organizational design could be an interesting alternative to the top-down manner that is currently implemented. This seems particularly appropriate where privacy is an issue, where computational resource or time are constrained, or where agents are not necessarily cooperative. In each of these cases, dividing the model and passing restricted types of information between entities would allow agents to locally organize without losing sight of more global concerns.

5. *Explore applicability of ODML to more competitive domains.*

The majority of the exploration that has taken place in this dissertation concerns cooperative domains - those where the participants operate in concert to achieve a common global objective. Many realistic domains exist that do not have this property, and are partially or fully competitive. Sections 5.2.6 and 5.2.4 touched on ODML's use in such circumstances while discussing the society and

marketplace paradigms. It is precisely because these domains are competitive that there is typically no single organizing authority capable of imposing a high-level structure on all participants. The centralized, top-down approach that this work focuses on is therefore is not likely to be appropriate.

As described in those sections, it is still possible to use ODML to guide the decision making processes of individual entities that exist or can exist in the environment. In a marketplace, for example, an agent seeking to determine if it should join that organization can use a model of the existing market forces and participants to estimate the potential benefits and drawbacks of doing so. In this case it is a utility function in the agent's node, not the global organizational utility, that would guide the search and evaluation process.

Even more than has been demonstrated, it may be difficult to craft effective and accurate models of such environments. In particular, because the other participants in the organization are competitive, it may be impossible to obtain good models of their behavior. In addition, it is quite possible that those behaviors could change in unexpected ways as a result of organizational decisions made by the modeling agent. Additional research is needed to evaluate ODML's usefulness in such situations.

6. *Explore applicability of ODML to existing organizations.*

Another area where ODML has potential benefits is in the evaluation of existing organizational structures. For example, one might wish to know how good an existing structure is relative to how good it could be. One might also want to determine how robust or brittle the structure is to changes in the environment, behavioral assumptions or participants.

Section 4.5 outlined how ODML could be used as part of a model-based diagnosis system to support online organizational adaptation. The same traits that would seem to support that can offer assistance in this problem. Assume, for example, that an appropriate template can be constructed for the system in question. Like all templates, this would describe the range of organizational alternatives, and relate all relevant characteristics together to produce a high-level utility function. Further assume that the existing organization can be derived from the template as a single instance, by making appropriate choices for each embedded decision point.

As with the adaptation technique, the relationships embedded in the ODML model can be used to answer these questions. For example, to determine the relative utility of the existing instance, the utility of that instance can be directly computed as normal. A search of the template space can then find the optimal organizational structure, whose utility can also be determined and compared against the existing utility. One way to evaluate the robustness of an organizational design uses the instance model as a simple, manual exploration tool. The designer may change constants, sever relationships, eliminate participants, etc., and then recompute the utility. If the model has been designed correctly, the

severity of the observed change in utility should be indicative of the severity of the change being evaluated.

## APPENDIX A

### TRANSLATING ODML TO MATHEMATICA

As described previously in Section 4.2.3, it is possible to translate portions of the ODML search space into a representation that may be interpreted by a general mathematical solver. It is recommended that the reader read that section before continuing. This appendix will provide additional details of that process, including tables which define equivalent mappings into Mathematica's expression language.

A successful translation can take place on any ODML partially-instantiated model that does not have any unbound variables that can affect a has-a relationship and does not contain functions that lack an appropriate mapping. The floating point and list data types from Section 2.2.3 are supported, but discrete distributions currently are not and therefore should also not be present. Given such an instance, the translation process operates starting from the root node and continuing recursively to all nodes the initial node has a has-a relationship with.

The translation of a node  $N$  begins by listing its has-a relationships. Each relationship  $H$  the node has is specified as a list with the form  $N[H] = \{n_1, \dots, n_n\}$ , where  $n$  is the unique name of another node in the organization. The parameters passed to  $N$  when it was created are listed next, followed by the node's fields (constants, constraints, modifiers and variables). These are translated as shown in Table A.1 (cf. the elements list in Section 2.2.1). The expressions are differentiated in italics because they will undergo further translation, as outlined below.

Because symbolic references within nodes are considered local, ODML's namespace is broken into isolated parts. Mathematica's namespace is flat, so to avoid

**Figure A.1.** ODML model element to Mathematica translation table.

Type	Contents	Mathematica Equivalent
Parameter	$N.P = \langle s, e \rangle$	$N[s] = e$
Constant	$N.C = \langle s, e \rangle$	$N[s] = e$
Constraint	$K = \langle N[s], op, e \rangle$	$AppendTo[constraints, N[s] op e]$
Modifier	$N.M = \langle s, op, e \rangle$	$N[s] = N[s] op e$
Variable (bound)	$N.V = \langle s, e \rangle$	$N[s] = e$
Variable (unbound)	$N.V = \langle s, \{e_1, \dots, e_n\} \rangle$	$AppendTo[variables, N[s]]$ $AppendTo[constraints, (N[s] == e_1    \dots    N[s] == e_n)]$

introducing collisions in translation, all symbols must be rewritten to take into account their node of origin. This is done by prefixing them with the node's name, which is guaranteed to be unique. For example, the constant  $c$  in  $N$  would become  $N.c$ . In addition, Mathematica does not natively support a notion of an “object” (i.e., a structure with a set of named characteristics or methods), or the dot-notation syntax that ODML uses to reference nonlocal characteristics. To overcome this limitation, I abuse Mathematica's function notion by creating a set of functions for each node, where each function defines a single characteristic (e.g., parameter, field, etc.) for that node. For example, the node:

```
<node name="A">
  <constant name="x">1</constant>
  <constant name="y">x + 1</constant>
</node>
```

... would be defined with the functions:

```
A[x] = 1
A[y] = A[x] + 1
```

From Mathematica's perspective,  $A$  is actually the name of a function that has been overloaded to supply two different expressions depending on the parameter that is passed in. When called, the parameter is the name of a characteristic in the original node  $A$ , which allows it to simulate the desired behavior despite the fact that there is no true object backing the definition. Because function names may be stored and referenced directly in Mathematica, chained object references can be simulated as well. For example, if  $A[x] = 10$  and  $B[y] = A$ , then  $B[y][x]$  would produce the desired value of 10, just as if  $B[y]$  actually contained a reference to the node  $A$ .

This is an abuse of the function notation because it relies upon the fact that  $x$  and  $y$  are unbound symbols in the resulting set of equations. This has the side effect of causing Mathematica to maintain the intended definitions. If, however,  $x$  and  $y$  were subsequently set to some common value (say the numeric constant 5), then both definitions would resolve to  $A[5]$ , and one of the two definitions would be lost. Because all symbols are prefixed by node names during the translation and the global scope is cleared before beginning, this cannot occur during the maximization process outlined earlier.

The remainder of the translation revolves around the expressions themselves. Because most of ODML's expression syntax uses standard infix notation, the majority of a typical expression can be passed in unchanged. There are two important differences. The first is that numeric constants, such as the literal “25.0”, are wrapped with the `Rationalize` function (e.g., `Rationalize[25.0]`). This causes Mathematica to treat them as exact rational value, rather than reals, which are only approximations. This allows the output of the maximization process to respect the intuitive meaning of certain inequalities when used in a discrete variable context. This is demonstrated in the following two variants of the same maximization process:



**Figure A.2.** ODML built-in function to Mathematica translation table.

ODML Function	Mathematica Equivalent
$\min(x_1, \dots, x_n)$	<code>Min[x<sub>1</sub>, ..., x<sub>n</sub>]</code>
$\max(x_1, \dots, x_n)$	<code>Max[x<sub>1</sub>, ..., x<sub>n</sub>]</code>
$\text{abs}(x)$	<code>Abs[x]</code>
$\text{choose}(x, y)$	<code>((x)! / ((x-y)! y!))</code>
$\text{sqrt}(x)$	<code>Sqrt[x]</code>
$\text{round}(x)$	<code>Round[x]</code>
$\text{map}(x, y_1^v, y_1^e, y_2^v, y_2^e, \dots, *, *^e)$	Currently not supported.
$\text{list}(x_1, \dots, x_n)$	<code>Flatten[x<sub>1</sub>, ..., x<sub>n</sub>]</code>
$\text{listitem}(\bar{x}, i)$	<code>Part[<math>\bar{x}</math>, i]</code>
$\text{size}(\bar{x})$	<code>Length[Flatten[<math>\bar{x}</math>]]</code>
$\text{unique}(\bar{x})$	<code>Union[<math>\bar{x}</math>]</code>
$\text{forall}(y, \bar{x}, e)$	<code>Map[e' &amp;, <math>\bar{x}</math>],</code> where $e' = (e \sim s/y/\#/g)$
$\text{forrange}(y, x_l, x_h, e)$	<code>Map[e' &amp;, Range[x<sub>l</sub>, x<sub>h</sub> - 1]],</code> where $e' = (e \sim s/y/\#/g)$
$\text{forallsum}(\bar{x})$	<code>Total[<math>\bar{x}</math>]</code>
$\text{forallprod}(\bar{x})$	<code>Apply[Times, <math>\bar{x}</math>]</code>
$\text{forallavg}(\bar{x})$	<code>Mean[<math>\bar{x}</math>]</code>
$\text{forallstddev}(\bar{x})$	<code>StandardDeviation[<math>\bar{x}</math>]</code>
$E(D)$	Currently not supported.
$V(D)$	Currently not supported.
$\text{Pr}(D, op, x)$	Currently not supported.
$\text{mc}(D, seed)$	Currently not supported.

```
In := Maximize[u, (u == 9.0) || u == 10.0) && (u < 10.0), u]
Out := {10., {u -> 10.}}
```

```
In := Maximize[u, (u == Rationalize[9.0] || u == Rationalize[10.0])
&& (u < Rationalize[10.0]), u]
Out := {9, {u -> 9}}
```

Notice that in the first maximization the strict inequality was not respected, while in the second it was.

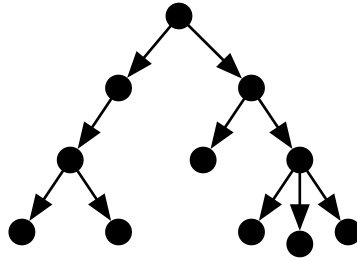
The second aspect of expression translation is the mapping of ODML's built-in functions into something equivalent using Mathematica's intrinsic functions. The currently implemented mappings are shown in Table A.2. Each ODML expression is scanned during translation, and when such a function is found it is converted as the table indicates. An example of this translation is shown in Figure 4.10.

## APPENDIX B

### A SURVEY OF MULTI-AGENT ORGANIZATIONAL PARADIGMS

The organization of a multi-agent system is the collection of roles, relationships, and authority structures which govern its behavior. All multi-agent systems possess some or all of these characteristics and therefore all have some form of organization, although it may be implicit and informal. Just as with human organizations, such agent organizations guide how the members of the population interact with one another, not necessarily on a moment-by-moment basis, but over the potentially long-term course of a particular goal or set of goals. This guidance might influence authority relationships, data flow, resource allocation, coordination patterns or any number of other system characteristics [75, 23]. This can help groups of simple agents exhibit complex behaviors and help sophisticated agents reduce the complexity of their reasoning. Implicit in this concept is the assumption that the organization serves some purpose – that the shape, size and characteristics of the organizational structure can affect the behavior of the system [62]. It has been repeatedly shown that the organization of a system can have significant impact on its short and long-term performance [23, 156, 51, 84, 128, 6, 180, 17], dependent on the characteristics of the agent population, scenario goals and surrounding environment. Because of this, the study of organizational characteristics, generally known as computational organization theory, has received much attention by multi-agent researchers.

It is generally agreed that there is no single type of organization that is suitable for all situations [91, 32, 112, 23]. In some cases, no single organizational style is appropriate for a particular situation, and a number of different, concurrently operating organizational structures are needed [63, 86]. Some researchers go so far as to say no perfect organization exists for any situation, due the inevitable tradeoffs that must be made and the uncertainty, lack of global coherence and dynamism present in any realistic population [150]. What is clear is that all approaches have different characteristics which may be more suitable for some problems and less suitable for others. Organizations can be used to limit the scope of interactions, provide strength in numbers, reduce or manage uncertainty, reduce or explicitly increase redundancy or formalize high-level goals which no single agent may be aware of [113, 61]. At the same time, organizations can also adversely affect computational or communication overhead, reduce overall flexibility or reactivity, and add an additional layer of complexity to the system [84]. By discovering and evaluating these characteristics, and then encoding them using an explicit representation [59], one can facilitate the process



**Figure B.1.** A hierarchical organization.

of organizational-self design [31] whereby a system automates the process of selecting and adapting an appropriate organization dynamically [112, 161]. This approach will ultimately enable suitably equipped agent populations to organize themselves, eliminating at least some of the need to exhaustively determine all possible runtime conditions a priori. Before this can occur, the space of organizational options must be mapped, and their relative benefits and costs understood.

These benefits and costs, and the potential advantages that could be provided by technologies able to make use of such knowledge, motivate the need to determine the characteristics of organizations and under what circumstances they are appropriate. While no two organizational instances are likely to be identical, there are identifiable classes of organizations which share common characteristics [150]. Several organizational paradigms suitable for multi-agent systems have emerged from this line of research [61]. These cover particularly common, useful or interesting structures that can be described in some general form. In this paper we will describe several of these paradigms, give some insight into how they can be used and generated, and compare their strengths and weaknesses. The vast amount of research which has been done in this field precludes a complete survey of any one technique; we hope to provide the reader with a concise description and a sample of the interesting work that has been done in each area.

In the following sections, we will describe the origin, form, function and characteristics of a typical structure for each organizational paradigm. Example applications will be presented, along with a discussion of techniques that have been employed to create the structures. By separating these concepts, we will distinguish between the characteristics of the organization generation process and those of the organizational structure itself, independently of how it was generated.

## B.1 Hierarchies

The *hierarchy* or *hierarchical organization* is perhaps the earliest example of structured, organizational design applied to multi-agent system and earlier distributed artificial intelligence architectures [60, 115, 61, 37, 11, 123, 132]. Agents are conceptually arranged in a tree-like structure, as seen in Figure B.1, where agents higher

in the tree have a more global view than those below them. In its strictest interpretation, interactions do not take place across the tree, but only between connected entities. More recent work [126] has explored starting with a strict hierarchy and augmenting it with cross links to allow more direct communication, which can reduce some of the latency that results from repeated traversals up and down the tree.

The data produced by lower-level agents in a hierarchy typically travels upward to provide a broader view, while control flows downward as the higher level agents provide direction to those below [11]. The simplest instance of this structure consists of a two-level hierarchy, where the lower level agents' actions are completely specified by the upper, which produces a global view from the resulting information [25]. More complex instances have multiple levels, while data flow, authority relations or other organizationally-dictated characteristics may not be absolute.

Fox [60] describes several different types of organizational hierarchies. The *simple* hierarchy endows a single apex member with the decision making authority in the system. *Uniform* hierarchies distribute this authority in different areas of the system to achieve efficiency gains through locality. Decisions are made by the agents which have both the information needed to reason about the decision, and the organizational authority to do make the decision. Each level acts as a filter, explicitly transferring information and implicitly transferring decisions up the hierarchy only when necessary. *Multi-divisional* hierarchies further exploit localization by dividing the organization along “product” lines, where products might represent different physical artifacts, services, or high-level goals. Each division has complete control over their product, which facilitates the decision making and resource allocation process by limiting outside influences. The divisions themselves may still be organized under a higher-level entity which evaluates their performance and offers guidance, but is strictly separated from the divisional decision process. These more sophisticated hierarchies look very much like like holarchical organizations, which are discussed in Section B.2.

## Characteristics

The applicability of hierarchical structuring comes from the natural decomposition possible in many different task environments. Indeed, task decomposition trees are a popular way of modeling individual agent plan recipes [42]; a hierarchical organization can be thought of as an assignment of roles and interconnections inspired by the global goal tree. The hierarchy's efficiency is also derived from this notion of decomposition, because the divide-and-conquer approach it engenders allows the system to use larger groups of agents more efficiently and address larger scale problems [213]. This type of organization can constrain agents to a number of interactions that is small relative to the total population size. This allows control actions and behavior decisions become more tractable, increased parallelism can be exploited, and because there is less potentially distracting data they can obtain a more cohesive view of the information pertinent to those decisions [132].

It is not sufficient to simply aggregate increasing amounts of information to obtain higher utility or better performance. This information must be matched with sufficient computational power and analysis techniques to make effective use of the

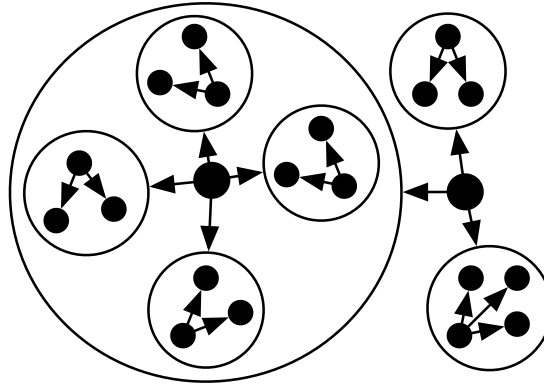
information [111]. Without this, the effort to transfer the data may be wasted and the excess information distract the agent from more important tasks. Alternatively, the information can be summarized, approximated or otherwise processed on its way up the tree to reduce the information load. However, in doing so, a new dimension of uncertainty is introduced because of the potential for necessary details to be lost. In this situation, the decision making authority should be correctly placed within the structure to maximize the tractable amount of useful information that is available that retains an acceptable level of uncertainty or imprecision [60, 113].

Using a hierarchy can also lead to an overly rigid or fragile organization, prone to single-point failures with potentially global consequences [130]. For example, if the apex agent were to fail the entire structure's cohesion could be compromised. Of course this agent could be replaced, but it may then prove costly to restore the concentrated information possessed by its predecessor. It is similarly susceptible to bottleneck effects if the scope of control decisions or data receipt is not effectively managed – consider what would happen if that apex agent received all the raw data produced by a large group of agents below it.

## Formation

Although the algorithm itself does not enforce a strict hierarchy such as the one described earlier, Smith's contract net protocol [178, 37] provides a straightforward mechanism to construct a series of connections with most of the same characteristics. In some of this early contract net work, the protocol was to explicitly form long-term organizational relationships, rather than the short-term contracts it has been typically used for more recently. The hierarchical structure that is produced by the process is implicitly based on the way the high-level goal can be decomposed. Upon receipt of a new task, an agent first chooses to perform the task itself, or search for agents willing to help complete the task. As part of this search process, the agent may decompose the task into subtasks or *contracts*. The agent, acting as a contractor, announces these contracts along with a bid specification to a subset of its peers who then decide if they wish to submit a bid. The bids which return to the contractor contain relevant information about the potential contractee which allows it to discriminate among competing offers. A contractee is selected and notified. Upon receipt of the new task, the contractee now faces the same question - should it perform the task itself or contract it out? Repeated invocations of this process produce a hierarchy of contractors and contractees. Because agents individually choose which contracts to bid on, and contractors choose which bids to accept, this strategy can effectively assign tasks among a population of agents without the need for a global view. The drawback to this approach is that it is myopic. Because the contracting agent does not necessarily take into account the needs of other contractors, it may bind scarce resource in suboptimal ways. For example, it may select a particular bid when viable alternatives exist, even though that particular bidder is critical to another agent [175].

As with most organizational structures, the shape of the hierarchy can affect the characteristics of both global and local behaviors. A very flat hierarchy where



**Figure B.2.** A holarchical organization.

agents have a high degree of connectivity can lead to overloading if agent resources are both limited and consumed as a result of these connections. Conversely, a very tall structure may slow the system's performance because of the delays incurred by passing information across multiple levels. One approach to making this tradeoff is the use of agent *cloning* [91, 41, 130]. An agent in such a system may opt to create a copy or clone of itself, possessing the same capabilities as the original, in response to overloaded conditions. If additional resources are available for this clone to use, this process allows the agent to dynamically create an assistant that can relieve excess burden from the original, reducing load-related errors or inefficiencies in the process. If the new agent is subordinate to the original, then a hierarchical organization will be formed in the process. Shehory [165] discusses using cloning when other task-reallocation strategies are not viable. In this work, an agent's overall load is a function of its local processing, free memory and communication. It uses a dynamic programming technique to compute an optimal time to clone, and an appropriately idle computational node to house the new agent. The clone receives a subset of the original task(s). The clones themselves require resources, and the results they produce may require an additional hop to get to their ultimate destination, so they may also be merged or destroyed when these costs outweigh their benefits.

## B.2 Holarchies

The term *holon* was first coined by Arthur Koestler in his book *The Ghost In The Machine* [103]. In this work, Koestler attempts to present a unified, descriptive theory of physical systems based on the nested, self-similar organization that many such systems possess. For example, biological, astrological and social systems are all comprised of multi-leveled, grouped hierarchies. A universe is comprised of a number of galaxies, which are comprised of a number of solar systems, and so on, all the way down to subatomic particles. Each grouping in these systems has a character derived but distinct from the entities that are members of the group. At the same

time, this same group contributes to the properties of one or more groups above it. The structure of each of these groupings is a basic unit of organization that can be seen throughout the system as a whole. Koestler called such units holons, from the Greek word *holos*, meaning “whole”, and *on*, meaning “part”. Each holon exists simultaneously as both a distinct entity built from a collection of subordinates and as part of a larger entity.

True to Koestler’s intent, this notion of a hierarchical, nested structure does accurately describe the organization of many systems. This concept has been exploited, primarily in business and manufacturing domains, to define and build structures called *holarchies* or *holonic organizations* which have this dual-nature characteristic. A sample such organization is shown in Figure B.2. In this diagram, hierarchical relationships are represented as directed edges, while circles represent holon boundaries. Enterprises, companies, divisions, working groups and individuals can each be viewed as a holons taking part in a larger holarchy. Fischer [56], Zhang [216], and Uliuru [194] have each organized agent systems by modeling explicit or implied divisions of labor in real-world systems as holons. In doing so, they create abstractions of these divisions, imparting capabilities to individual holons instead of individual agents. This layer of abstraction allows other entities in the system to make more effective use of these capabilities, by reasoning and interacting with the group as a single functional unit.

The defining characteristic of a holarchy is the partially-autonomous *holon*. Each holon is composed of one or more subordinate entities, and can be a member of one or more superordinate holons. Holons frequently have both a software and physical hardware component [216, 193], although this does not preclude their usage in purely computational domains. The degree of autonomy associated with an individual holon is undefined, and could differ between levels or even between similar holons at the same level. There is the presumption, however, that the level of autonomy is neither complete nor completely absent, as these extremes would lead to either a strict hierarchy or an unorganized grouping, respectively. Within the holarchy, the chain of command generally goes up – that is, subordinate holons relinquish some of their autonomy to the superordinate groupings they belong to. However, there is also the more heterarchical notion that individual holons determine how to accomplish the tasks they are given, since they are likely the locus of relevant expertise. Many holonic structures also support connections between holons across the organization, which can result in more amorphous, web-like organizational structures that can change shape over time [56, 216].

It would not be incorrect to conclude that a holarchy is just a particular type of hierarchy. If we relax our definition of hierarchy to allow some amount of cross-tree interactions and local autonomy, the two styles share many of the same features and can be used almost interchangeably. These richer models then begin to resemble and take on the characteristics of nearly-decomposable hierarchies [172], where lateral interactions are weak but still relevant. Very flat holarchies can also begin to resemble federations, which will be discussed in Section B.7.

## Characteristics

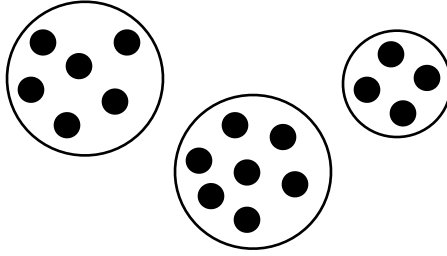
As with the conventional hierarchies from the previous section, holarchies are more easily applied to domains where goals can be recursively decomposed into subtasks that can be assigned to individual holons (although this is not essential). Given such a decomposition, or a capability map of the population, the benefits the holonic organizations provide are derived primarily from the partially autonomous and encapsulated nature of holons. Holons are usually endowed with sufficient autonomy to determine how best to satisfy the requests they receive. Because the requester need not know exactly how the request will be completed, the holon potentially has a great deal of flexibility in its choice of behaviors, which can enable it to closely coordinate potentially complementary or conflicting tasks. This characteristic reduces the knowledge burden placed on the requester and allows the holon's behavior to adapt dynamically to new conditions without further coordination, so long as the original commitment's requirements are met. A drawback to this approach is that, lacking such knowledge, it is difficult to make predictions about the system's overall performance [12].

## Formation

The challenge in creating a holonic organization revolves around selecting the appropriate agents to reside in the individual holons. The purpose of the holon must be useful within the broader context of the organization's high-level goals, and the holon's members must be effective at satisfying that purpose. Zhang [216] uses a model of static holons along with so-called mediator holons to create and adapt the organization. The static groups consist of product, product model and resource holons, each of which corresponds to a group of physical or information objects in the environment (e.g. manufacturing device, design plans, conveyors, etc.). The mediator holon ties these together, by managing orders, finding product data and coordinating resources in a manner similar to a federation, which will be discussed in Section B.7. Each new task is represented by a dynamic mediator holon (DMH), which is created by the mediator holon. The DMH is destroyed when the task is completed.

Another approach to holarchy construction uses *fuzzy entropy* minimization to guide the formation of individual holonic clusters [182, 193]. In this work, the collection of holons is assumed to be initially described with a set of source-plans, each of which describes a potential assignment of holons to clusters, along with a set of probabilities that describe the degree of occurrence of those clusters. From this initial uncertain information, one can derive the preferences which agents have to work with one another, and then choose the source plan which has the minimal entropy with respect to those preferences. The goal of this technique is to ensure that each holon has the necessary knowledge and expertise needed to perform its task. The preference that one agent has for another represents this knowledge or expertise requirement, so the minimally fuzzy set will satisfy this goal by clustering agents which have common preferences. In [193], Ulieru adds a genetic algorithm approach to this scheme to help explore the space of possible clustering assignments.





**Figure B.3.** A coalition-based organization.

## B.3 Coalitions

The notion of a *coalition* of individuals has been studied by the game theory community for decades, and has proved to be a useful strategy in both real-world economic scenarios and multi-agent systems. If we view the population of agents  $A$  as a set, then each subset of  $A$  is a potential *coalition*. Coalitions in general are goal-directed and short-lived; they are formed with a purpose in mind and dissolve when that need no longer exists, the coalition ceases to suit its designed purpose, or critical mass is lost as agents depart. Related research has extended this to longer-term agreements based on trust [15] and to the iterative formation of multiple coalitions in response to a dynamic task environment [131]. They may form in populations of both cooperative and self-interested agents.

A population of agents organized into coalitions is shown in Figure B.3. Within a coalition, the organizational structure is typically flat, although there may be a distinguished “leading agent” which acts as a representative and intermediary for the group as a whole [102]. Once formed, coalitions may be treated as a single, atomic entity. Therefore, although coalitions have no explicit hierarchical characteristic, it is possible to form such an organization by nesting one group inside another. Overlapping coalitions are also possible [164]. The agents in this group are expected to coordinate their activities in a manner appropriate to the coalition’s purpose. Coordination does not take place among agents in separate coalitions, except to the degree that their individual goals interact. For example, if one coalition’s goal depends on the results of another, these two groups might need to agree upon a deadline by which those results are produced. In this case, it would be the leading or representative agents forming the commitment, not arbitrary members of the coalition.

In addition to the problem of generating coalition structures, one must also determine how to solve the goal presented to the coalition. If the population is self-interested, a division of value to be apportioned to participants once that goal has been satisfied must also be generated and agreed upon [157].

## Characteristics

The motivation behind the coalition formation is the notion that the value of at least some of the participants may be superadditive along some dimension. Analogously, participants' costs may be subadditive. This implies that utility can be gained by working in groups – this is the same rationale behind buying clubs, co-ops, unions, public protests and the “safety in numbers” principle. For instance, in an economic domain, a larger group of agents might have increased bargaining strength or other monetary reward [190]. In computational domains we might expect more efficient task allocation, or the ability to solve goals with requirements greater than any single agent can offer [164]. In physically-limited systems, coalitions have been used to trade off the scope of agent interactions with the effectiveness of the system as a whole [175]. This last application directly affects the coordination costs incurred by the system; we will see that this capability and purpose are shared by congregations in Section B.5.

One could argue that all agents in the environment should always join to form the all-inclusive *grand coalition*. Indeed, under certain circumstances this is appropriate, since the structure would have the resources of all available agents at its disposal, which theoretically would provide the maximum value. There are costs associated with forming and maintaining such a structure however, and in real world scenarios this can be both an impractical and unnecessarily coarse solution [157]. Therefore, the problem of coalition formation becomes one of selecting the appropriate set(s)  $S \subseteq A$  which maximizes the utility (value minus costs) that coalition  $v_S$  can achieve in the environment. The value and cost of the coalition are generic terms, which may in fact be functions of other domain-dependent and independent characteristics of the structure.

## Formation

The complexity of the coalition formation task depends on the conditions under which the coalitions will exist, and the types of coalitions which are permitted. As with all organizations, operating in dynamic environments will be harder to maintain than in static ones. Additional complexity is also incurred if the partitioning of agents is not disjoint; that is, agents can have concurrent membership in more than one coalition. Uncertain rewards, self-interested agents and a potential lack of trust while coordinating add further obstacles to the process.

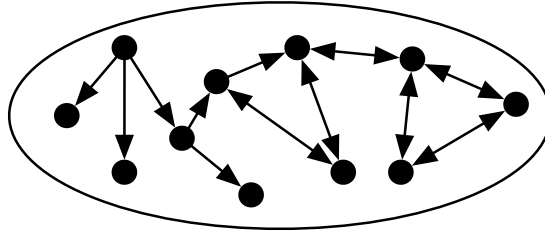
Sandholm [156] analyzes the worst case performance of forming exhaustive, disjoint coalitions over a static agent population from a centralized perspective. They show that by searching only the two lowest levels of a complete coalition structure graph, an  $a$ -approximate value solution can be found to the partitioning problem, where  $a = |A|$ . Although the search of  $2^{a-1}$  possible allocations still grows exponentially with  $a$ , the fraction of coalition structure needing to be searched approaches zero. They also present an anytime algorithm which can meet tighter bounds given additional time. Later work empirically evaluates the average-case performance of

three anytime search techniques [105]. The algorithms' performances varied by domain characteristics; and no single technique was best in all conditions.

Shehory [164] has studied how coalitions may be used to enable task achievement by a group of agents. In their scenario, a set of interdependent (precedence) tasks must be accomplished, some of which require multiple agents to perform. The agents are cooperative and potentially heterogeneous in their capabilities. The strategy they employ draws on techniques used by Chvatal's greedy set covering algorithm [27], which tries to find the minimum set of subsets that together contain each member of a target set. The initial values of all possible size-bounded coalitions are first computed and then iteratively refined in a distributed manner by the agents, taking into account task ordering and capability requirements. Once computed, the highest valued coalitions, either disjoint or overlapping depending on the selection algorithm, are instantiated. This algorithm was also augmented to support dynamically arriving tasks. A drawback to this addition is that, in the worst case, the organization process needs to be redone for each task, incurring a significant communication cost. Also limiting the potential scalability of this approach is the need for each agent to have full knowledge of the available agents and tasks.

Lerman [108] presents a scalable strategy where coalitions are formed between self-interested agents based only on local decision making. In this work agents operate in an electronic marketplace consisting of a number of extant purchase orders, with the objective of forming or joining a coalition of buyers that satisfied a need at the lowest price. Coalitions form around purchase orders, where agents form or join a coalition by adding a purchase request to an order, and can leave that coalition by removing their request. Agents in the system can move at will between purchase orders, searching for the one which offers the best value (lowest cost). An analysis based on differential equations shows that this strategy reaches equilibrium (later work [107] expands on these mathematical techniques to analyze other distributed behaviors). It also has low communication and computational requirements. However, it does not provide guarantees on the achievable value or convergence rate, which would be affected by scale, and does not have a notion of deadlines on the purchase orders.

Soh [181] presents a technique where coalitions are dynamically created in response to the recognition of tracking tasks in a distributed sensor network. In this work, agents are assumed to have incomplete, uncertain knowledge and must respond to events in real time for goal achievement to be possible. As such, coalitions are formed in a sacrificing, rather than optimal manner. An agent initiates coalition formation by first using local knowledge to select a subset of candidate partners that it believes will satisfy its requirements, both in terms of capabilities and willingness to cooperate. Next, it sequentially engages these candidates, in utility-ranked order, in argumentative negotiation, where offers and counteroffers are exchanged. This proceeds until satisfactory membership is decided, or the candidate list is exhausted. Agents are cooperative, so during this negotiation process agents explicitly decide what coalition(s) they are willing to join based on perceived gains in utility. This approach does not make any guarantees about coalition value, or even that a satisfactory coalition will be found, but given the relatively short time in which an allocation must be made it would seem to be a reasonable strategy. In addition, reinforcement learn-



**Figure B.4.** A team-based organization.

ing is used over the course of events to estimate candidate utility more accurately and select the most beneficial negotiation strategy, which should improve coalition value in the long run for reasonably stable environments. By storing preferences over multiple episodes, this learning also implicitly adds longevity to coalitions, giving organizational structures produced by this technique an interesting mix of dynamic and long-term characteristics.

## B.4 Teams

An agent *team* consists of a number of cooperative agents which have agreed to work together toward a common goal [61, 185, 8]. In comparison to coalitions, teams attempt to maximize the utility of the team (goal) itself, rather than that of the individual members. Agents are expected to coordinate in some fashion such that their individual actions are consistent with and supportive of the team's goal. Within a team, the type and pattern of interactions can be quite arbitrary, as seen in Figure B.4, but in general each agent will take on one or more roles needed to address the subtasks required by the team goal. Those roles may change over time in response to planned or unplanned events, while the high-level goal itself usually remains relatively consistent (although exception handling may promote the execution of previously dormant subtasks).

This description of agent teams is quite general, and nearly any cooperative agent system has characteristics that are similar to these, if only implicitly. However, systems that maintain an explicit representation of their teamwork or joint mental state are differentiated in their ability to reason more precisely about the consequences of their teamwork decisions [93, 70, 185]. For example, they will typically have representations of shared goals, mutual beliefs and team-level plans. This type of representation provides flexibility and robustness by allowing the agents to explicitly reason about team-level behaviors, where a less explicit system may rely on a set of assumptions that ultimately make the system brittle in the face of unexpected situations.

## Characteristics

The primary benefit of teamwork is that by acting in concert, the group of agents can address larger problems than any individual is capable of [72]. Other potential benefits, such as redundancy, the ability to meet global constraints, and economies of scale can also be realized [77]. However, it is the ability of the team (members) to reason explicitly about the ramifications of inter-agent interactions which gives the team the needed flexibility to work in uncertain environments under unforeseen conditions. The drawback to this tighter coupling is increased communication [141], so the team and joint goal representations, domain characteristics and task requirements are frequently used to determine what level of cooperation (and therefore communication) is needed [145].

Jennings [93] describes an electricity transportation management system which employs teamwork to organize the activities of diagnostic agents. Lacking such structure, the agents were prone to incoherent and wasteful activities, since they did not always share useful behavior information or propagate important environmental knowledge. By providing agents with an explicit representation of shared tasks and the means by which cooperation should progress, the agents were able to accurately reason about and resolve these interactions by employing team-level knowledge. Similarly, in [185], teamwork is used to provide the structure and coordination needed by agents to address interdependent goals in dynamic environments, such as tactical military exercises and competitive soccer games. These works demonstrate how pathological, but hard to predict failures can be addressed if the plans are backed up by a general model of teamwork.

## Formation

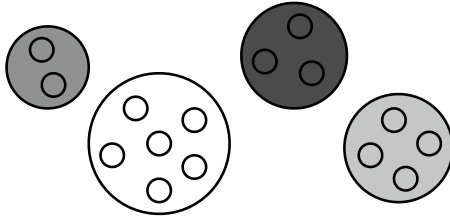
The challenges associated with team formation involve three principal problems: determining how agents will be allocated to address the high-level problem, maintaining consistency among those agents during execution, and revising the team as the environment or agent population changes [93, 125, 188].

The selection and role-assignment of agents that will work on the high-level problem depends on the goal's requirements, the capabilities of the candidate agents, and the knowledge of the selecting process itself [189, 8]. Initially, the process or agent performing the team construction must be aware of the agents which could potentially form the team. In the case of a static, reasonably sized agent population this can be done off-line as part of the system design, or the members can be dynamically discovered and assessed. This latter technique can be accomplished using well-known discovery mechanisms such as the contract net protocol [178] or matchmaker intermediaries [184]. Once a suitable pool has been found, the capabilities and preexisting responsibility of those agents must be evaluated relative to the needs of the goal. Typically, agents are each denoted to have a set of capabilities, while the goal's subtask(s) are of a particular type. If an agent's capabilities include that subtask's type, it can perform the task [189, 55]. The discovery mechanisms may include an implicit ranking technique, such as the bidding process employed in contract net, which makes the se-

lection process relatively straightforward. Tidhar [189] suggests a different technique where the agent characteristics are derived at compile time, either through designer input or automatic analysis of the agent’s plan library. Candidate teams comprised of a subset of those agents may also be specified, which also are marked with their characteristics. At runtime, these characteristics are matched with the goal requirements as part of the team allocation search. By including these characteristic labels, the number of possible team combinations can be greatly reduced.

Tambe’s STEAM [185] architecture provides a flexible method for representing and adapting team behaviors. It is based on the joint intentions framework [116], which formally defines how agents should reason over joint commitments and shared goals, and SharedPlans theory [71], which provides a formal way to encode and reason about joint plans, intentions and beliefs. Together, these help ensure a consistency of belief, or a desire to enact such a belief, across all team members. The commitments formed through the joint intentions process provide the explicit structure needed to reason about and monitor performance on a team level. Team plans are represented using a hierarchical decomposition tree, with nodes representing tasks for both teams and individuals, with associated preconditions, application and termination rules. Agents may simultaneously take part in several different tasks, and corresponding roles. The team’s cohesion is derived primarily from the joint intentions created as part of executing the team plans. Upon selecting a team task, agents first broadcast this intention to affected agents, and wait until a commitment to that task has been established between all participants. The existence of this commitment directs agents to propagate changes whenever the task is perceived to be achieved, unachievable or irrelevant, before taking local action itself. This trades off the potential reaction speed of the team and the cost of communication with group conformity. A decision theoretic approach is used to guide communication acts, which explicitly trades off the costs of communication with those of inconsistent beliefs. Nair [137] has also explored the possibility of using simulated emotions to provide the motivation to enforce team-level behaviors.

In STEAM, monitoring and repair of the team is accomplished with the use of role constraints [185]. Team members are assigned a role, based on the particular task they are working on. These roles are further constrained such that some particular combination of them (e.g. and, or) are needed to accomplish the task. One can then monitor if a task is achievable by monitoring the health of the individual agents, and using that information to evaluate the satisfiability of the role constraints. Such monitoring can be performed through explicit queries, environmental observations or by eavesdropping on communication, which can reduce the increased communication usually associated with teams. Kaminka [98] has demonstrated that the latter technique can perform well when coupled with a plan-recognition algorithm. Failures can thus be detected, and potentially resolved through an appropriate role-substitution, or the task abandoned if no substitution is possible. Alternately, one could use a diagnosis system [93, 78] to more precisely identify the root cause of the failure. Interestingly, this repair operation can itself be cast as a team task, so mutual agreement that a repair is necessary must be achieved before potentially drastic measures are taken. Nair [138] shows how an MDP incorporating team and role-allocation knowl-



**Figure B.5.** Congregations of agents.

edge can improve the system’s response in cases of multiple role failure. In this case, a suitable locally optimal policy for the reallocation decision problem can be found by analyzing the team’s plans, and then used to guide online responses to failures. This work showed that such policies can provide improved performance versus more heuristic and analytic techniques. A similar technique was also shown in that work to improve initial role allocation.

Tidhar [188] uses a similar hierarchical plan representation to represent teamwork in a tactical air mission scenario. Team membership and role assignment are performed by matching agent capabilities to one or more role’s requirements. As in STEAM, teams can be broken down into sub-teams, and agents may use both implicit (observation) and explicit (messaging) forms of coordination.

The Generalized Partial Global Planning (GPGP) framework also employs techniques that allow agents to act using team semantics [38, 109]. Where a STEAM-driven system will typically organize in an explicit, controlled fashion in response to a perceived goal, a GPGP-team is created in a more dynamic, emergent fashion. GPGP agents are provided with a set of individual plans which model a range of alternative ways that goals may be achieved. The sub-goals modeled in these plans may affect or be affected by other agents in the environment, although this may not be initially recognized. By communicating with one another and exchanging plans and schedules, these *non-local interrelationships* between tasks may be recognized. For example, the results from one agent’s activity may be a strict prerequisite for another agent’s task. They may alternately be a facilitating, but not required input to a task. By recognizing these interrelationships, and sharing knowledge of what goals are being pursued, agents gradually build an internal model of how their actions may affect others. This knowledge is similar to that created by the more formal joint intentions of STEAM, and allows agents to influence local behavior and communicate results as if they were members of a common team.

## B.5 Congregations

Similar to coalitions and teams, agent *congregations* are groups of individuals who have banded together into a typically flat organization in order to derive additional benefits. Unlike these other paradigms, congregations are assumed to be long-lived

and are not formed with a single specific goal in mind. Instead, congregations are formed among agents with similar or complementary characteristics to facilitate the process of finding suitable collaborators, as modeled in Figure B.5. The different shadings in this figure represent the potentially heterogeneous purpose behind each grouping, in comparison to the typically more homogeneous coalitions in Figure B.3. Individual agents do not necessarily have a single or fixed goal, but do have a stable set of capabilities or requirements which motivate the need to congregate [18, 69]. Analogous human structures include clubs, support groups, secretarial pools, academic departments and religious groups, from which the name is derived.

Congregating agents are expected to be individually rational, by maximizing their local long-term utility. Group or global rewards are not used in this formalism [18]. It is this desire to increase local utility which drives congregation selection, because it is the utility that can be provided by a congregation's (potential) members that determine how useful it is to the agent. Agents may come and go dynamically over the existence of the congregation, although clearly there must be a relatively stable number of participants for it to be useful. Agents must also take enough advantage of the congregation so that the time and energy invested in forming and finding the group is outweighed by the benefits derived from it. Since congregations are formed in large part to reduce the complexity of search and limit interactions, communication does not occur between agents in different congregations, although the groups are not necessarily disjoint (i.e., an agent can be a member of multiple congregations). The net result of the congregating behavior is an arrangement that can produce greater average utility per cycle spent computing or communicating [16].

## Characteristics

Although congregations can theoretically share many of the same benefits of coalitions, their function in current research has been to facilitate the discovery of agent partners by restricting the size of the population that must be searched. As a secondary effect these groupings can also increase utility or reliability by creating tighter couplings between agents in the same congregation, typically by imposing higher penalties for decommitment or increasing information sharing among congregating peers. The downside to this strategy is that the limited set may be overly restrictive, and not contain the optimal agents one might interact with given infinite resources. So, in forming the congregation, one is trading off quality and flexibility for a reduction in time, complexity or cost. If an appropriate balance can be found, this will result in a net gain in utility.

This hypothesis is borne out in the experiments from an information economy domain [16]. This work varied the number of congregations that agents were allowed to form. Since the population size was static, the average congregation size decreased as the number of congregations increased. The accumulated quality decreased proportionally because of less flexibility in agent interactions. However, these smaller congregations also incurred lower overhead, and thus had less cost. A median point was discovered in the space which produced maximum value.

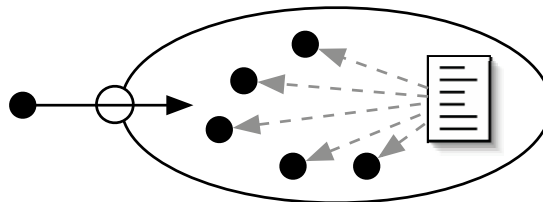


## Formation

Like coalition formation, congregation formation involves selecting or creating an appropriate group to join, and suffers from similar complexity problems as the agent population grows. Because congregations are more ideologically or capability driven, and there is usually no specific goal or task to unite them, one must first define how these groups may be differentiated. In [17] Brooks proposes using labels to address this problem. A label is a suitably descriptive tag assigned to each congregation which serves to both distinguish it from other groups and advertise the characteristics of its (desired) members. Assuming that agents have an ordered preference for such labels, the congregators' action is simply to move to the congregation for which it has the highest preference. The problem is then to create a number of logical points where agents may congregate and then decide upon the labels each congregation point will have; these labels help determine the makeup of the population which gathers there. Each agent was placed into one of several affinity groups, and a congregation is stable if and only if it contains only members of the same affinity group. Different numbers of labelers were then added which could attach labels to the congregation points. As with the congregators, the labelers were stable if and only if the congregation they provided the label to was homogeneous. The experimental and analytic results demonstrated that by increasing the number of labelers the system converged more quickly.

Brooks [16] presents a variation of this formation technique used in an information economy which also takes into account the costs associated with congregation size. In this scenario there are a set of buyers and sellers. Each buyer has an information preference, and each seller may choose what type of information to offer. The buyer's preference is soft – they have an optimal type, but are also willing to purchase related information, where similarity determines how much they are willing to pay. Instead of explicitly labeling congregation points, agents freely move through the system seeking groups that provide acceptable utility. The scenario is episodic, where during each episode agents elect to stay in place or randomly move to a new congregation. At the end of each episode an auction takes place from which buyers and sellers obtain their utility. The utility is based on the price of the goods bought and sold, combined with the costs incurred during the auction. This cost, divided uniformly among the congregation members, is proportional to the complexity of the auction, which is itself determined by the number of participants. Satisfied agents remain, while those which do not obtain enough utility move. This process results in an emergent population of congregations that trades off utility for computation time.

Griffiths' notion of a *clan* closely parallels the definition of a congregation [69]. He presents a technique where clans are formed as part of a self-interested activity to increase local utility or decrease the probability of failure. If a motivating factor is exhibited by the agent, such as a desire to increase information gain or decrease commitment failure, clan formation may be initiated. Clan formation begins with the agent identifying how large a clan it wishes to create, which is based on the competing utility (in value added) and cost (in computational complexity) that grow in proportion to clan size. A trust value is then used to determine what agents it



**Figure B.6.** An agent society.

could invite, while the perceived capabilities or benefits of those individual agents are used to determine the appropriately sized subset that it will invite. In lieu of a negotiation process or explicit reward, invitation recipients determine if they will accept the invitation based first on their trust in the sender, and second on the perceived local gain they would receive by joining. The sender includes information about itself in the invitation as a sort of capability advertisement to facilitate this determination. If a sufficient number of agents agree, the clan is formed, otherwise the attempt is abandoned.

Although it does not strictly deal with congregating agents, Sen’s work on reciprocal behavior [162] has some of the same characteristics. In this system, agents become more inclined to cooperate or assist another agent when it has a favorable history with that other agent. Specifically, agents track if others have cooperated with it in the past, or if it has cooperated with them, along with the approximate costs of those experiences. If an agent has a favorable balance of cooperation, it will be more inclined to give or receive assistance. The cooperation decision process is stochastic, enabling reciprocal relationships to be created or promoted even when a strictly positive balance does not exist. Weak groups may form between agents using this strategy who have complementary capabilities, which is similar to the notion of congregations we have presented. Because agents will more likely communicate with those that will help it, interactions can become implicitly confined within the group. These groupings are not formalized or well-defined, however, and communication is not necessarily restricted by the approximate boundaries that form. Sen showed that, among a group of self-interested agents operating in a package delivery domain, a population containing reciprocal agents outperformed a selfish population.

## B.6 Societies

Drawing from our own experiences with biological societies, a *society* of agents intuitively brings to mind a long-lived, social construct. Unlike some other organizational paradigms, agent societies are inherently open systems. Agents of different stripes may come and go at will while the society persists, acting as an environment through which the participants meet and interact. A canonical example of this paradigm is the electronic marketplace (discussed in more detail in Section B.8), consisting of buyers and sellers striving to maximize their individual utility [205, 4]. A more ambitious

example is the “agent world”, a permanent operating environment for agents that in some ways parallels our own [44, 208]. Agents will have different goals, varied levels of rationality, and heterogeneous capabilities; the societal construct provides a common domain through which they can act and communicate. Societies are also more ephemeral constructs than others paradigms we have seen so far. They impose structure and order, but the specific arrangement of interactions can be quite flexible. Within the society, agents may be sub-organized into other organizations, or be completely unrelated.

A second distinguishing characteristic of societies is the set of constraints they impose on the behavior of the agents, commonly known as *social laws*, *norms* or *conventions*. This arrangement is shown abstractly in Figure B.6, where the agents within the society have been provided with a set of specified norms. These are rules or guidelines by which agents must act, which provides a level of consistency of behavior and interface intended to facilitate coexistence. For example, it might constrain the type of protocol(s) agents can use to communicate, specify a currency by which they can transfer utility, or limit the behaviors the agent can exhibit in the environment. Penalties or sanctions may also exist to enforce these laws.

The set of laws embedded in a society must strike a balance among objectives [57]. It must be sufficiently flexible that goals are achievable, but not so much so that the beneficial constraints provided by the laws are lost. It must also be fair, such that the goals of one class of individuals are not incorrectly valued higher than those of another. These issues arise naturally in any structured, multiple participant system; Moses argues that most multi-agent systems have some form of social laws in place, if only implicitly [133].

## Characteristics

In [168], Shoham presents a grid world where robots must move from one location to another in accordance with a set of dynamically arriving tasks. Conflicts can arise when two or more agents attempt to occupy the same location at the same time along their chosen paths. They argue that a centralized solution is untenable, because of the potentially large number of interactions that must be continuously reasoned over in the heterogeneous population. Neither is a fully decentralized solution appropriate, because of the number of negotiation events that would need to take place at each time step. This motivates the need for “traffic laws”, a type of social law which does not eliminate such interactions, but should minimize the need for them. The traffic laws in this research are computed offline, and constrain the robots’ movement patterns in such a way that collisions do not occur, and destinations are reachable within a bounded amount of time. Vehicular traffic laws serve the same purpose in human societies. When driving a car there is no central authority which determines when and where we should go, and neither is there a free-for-all on the roads where one must talk to every other driver before proceeding. The challenge then is to design a set of laws that minimizes conflicts and encourages efficient solutions.

Although social laws were used to provide efficiency benefits in the work above, the purpose of an agent society is not always as quantitatively-driven as other organi-

zational constructs. Indeed, most research on agent societies is more concerned with how the concepts they embody can be used to facilitate the construction of large-scale, open agent systems in general. For example, Moses [133] argues that social laws can provide a formal structure upon which more complex inter-agent behaviors can be built. By limiting and enforcing these restrictions, agents can make simplifying assumptions about the behavior of other agents, which can make interaction and coordination more tractable.

In addition to formalizing normative behaviors, mechanisms may also be established to ensure or encourage that such laws are respected. One approach accomplishes this through explicit representations of reputation or trust [135, 152, 153]. An agent's behavior and interactions are observed by its peers and evaluated in the context of the norms it has agreed to. Deviation from those norms will result in a worsening reputation. This decreased reputation can in turn affect the utility the agent obtains, through increased decommitment penalties or competition from more reputable peers. In a rational agent this will serve as a deterrent to violating conventions. A different, but complementary approach instantiates and enforces social laws using social institutions provided in the environment [44, 30]. Agents are expected to formalize their interactions using contracts, which are independently verified by these institutions, thereby relocating some of the traditionally agent-centric complexity into a service available to the population as a whole. This reduces the burden placed on agent designers, and provides a mechanism where systemic (non-localized or long-term) failures may be detected more readily. This more rigorous enforcement of social laws also helps address the problem of unreliable, dishonest or malicious agents operating in the open environment.

Huhns [90] provides similar motivation for common communication languages, shared or interoperable ontologies and coordination and negotiation protocols, all of which may be specified as part of the society's structure. These beliefs can be supported by our own experiences in real life. It should be clear that complex human societies are founded upon the ability to interact with one another. Mutually understood and respected norms simplify many aspects of day-to-day existence. These principles can be used to the same effect in agent societies.

## Formation

There are two aspects to the society formation problem. The first is to define the roles, protocols and social laws which form the foundation of the society. Given such a definition, the second problem is to implement the more literal formation of the society, by determining how agents may join and leave it.

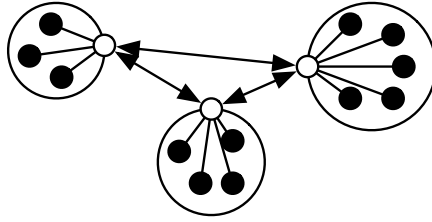
If the society is to be an open and flexible system, its structure must be formally encoded so that potential members may analyze it and determine compatibility. This description can be as simple as a set of common interfaces that must be implemented, or a complex description of permissible roles, high-level objectives and social laws. Dignum [49, 48] presents a three-part framework, consisting of organizational, social and interaction models. The organizational model defines the roles, norms, interactions and communication frameworks that are available in the environment. The

social model, instantiated at run-time, defines which roles agents have taken on. The interaction model, also created at run-time, encodes the interactions between agents that have been agreed-upon, including the potential reward and penalties. The latter two models are supported by contracts between the relevant entities. This formalism is similar to that proposed by Artikis [4], which provides additional details describing operators that can be used to encode social laws, roles and normative relations. Because the society is intended to be open, these structures do not involve the internal implementation of agents, but describe only the intended or expected externally observable characteristics of the participants and environment.

Assuming it is possible to encode the social laws in a way that makes them intelligible to agents, one still faces the challenge of determining what conventions should be enacted. Fitoussi [57] presents a notion of minimal social laws, where he argues that one should choose the smallest and simplest set of norms that address the needs of the society. This is consistent with the tradeoff between flexibility and complexity mentioned above. Work has also been done exploring the dynamic emergence of norms, for when social laws cannot specified off-line or if there is a desire for the corpus to be responsive to changing conditions [5, 76]. Walker and Wooldridge [200] propose and evaluate a number of ways that a group of agents can reach norm consensus based on locally available information.

Dellarocas defines the act of an agent entering a society to be the *socialization* process [44]. In that work, they suggest this can be accomplished through an explicit negotiation process between the agent and a representative of the society, as shown in the left side of Figure B.6. This exchange results in a *social contract*, or an explicit agreement made between the agent and the society indicating the conditions under which the agent may join that society. This allows the possibility of capable agents dynamically learning, and potentially negotiating over, the rules it must abide by in that society. López y López [212], present a framework in which the facilities for norm reasoning needed to support these behaviors can take place. A similar view is taken by Glaser in [66], with the additional stipulation that the joining agent must increase the utility of the society. This naturally extends to multi-society environments, where an agent's skills and goals define how good a fit it is with a particular society. Some of the challenges associated with operating in multi-society environments seem to be comparable, though larger in scale, to those encountered during coalition or congregation formation.

Because of their inherent flexibility, a great deal of additional complexity may be associated with social organizations. Sophisticated legal systems, communication bridges, ontologies, exception handling services, directories may all be part of the society model [44, 49, 100]. Some or all of these may be directly instantiated by trusted agents taking on so-called facilitation roles (differentiated from the operational roles taken on by worker agents). Of course, agents acting in the society must have a certain level of sophistication to know how and when to use such services. An interesting almost-paradox exists in this relationship. Although the society exists in part to reduce the complexity burden imposed on the participants, the participants must raise their level of complexity to take advantage of these benefits. In the case where interactions with some or all social services are mandatory (e.g. legal or arbitration



**Figure B.7.** An agent federation.

services), this additional complexity is similarly unavoidable and can act as a barrier to entry.

## B.7 Federations

*Agent federations*, or *federated systems*, come in many different varieties. All share the common characteristic of a group of agents which have ceded some amount of autonomy to a single delegate which represents the group [206, 65]. This organizational style is modeled on the governmental system of the same name, where regional provinces retain some amount of local autonomy while operating under a single central government. The delegate is a distinguished agent member of the group, sometimes called a facilitator, mediator or broker [184, 75]. Group members interact only with this agent, which acts as an intermediary between the group and the outside world, as shown in Figure B.7. In that figure each grouping is a federate, and the white agent situated at the edge of each federate is the delegated intermediary. Typically, the intermediate accepts skill and need descriptions from the local agents, which it uses to match with requests from intermediaries representing other groups. In this way the group is provided with a single, consistent interface. This level of indirection is similar to that seen in holons, and provides some of the same benefits.

### Characteristics

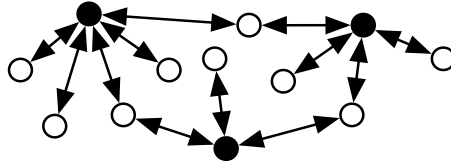
The capabilities provided by the intermediary are what differentiate a federation from other organizational types. The intermediary functions on one hand by receiving potentially undirected messages from its group members. These may include skill descriptions, task requirements, status information, application-level data and the like. These will typically be communicated using some general, declarative communication language which the facilitator understands [65]. Outside of the group, the intermediary sends and receives information with the intermediaries of other groups. This could include task requests, capability notifications and application-level data routed as part of a previously created commitment. Implicit in this arrangement is that, while the intermediary must be able to interact with both its local federation members and with other intermediaries, individual normal agents do not require a

common language as they never directly interact. This makes this arrangement particularly useful for integrating legacy or an otherwise heterogeneous group of agents [65, 167].

The intermediary itself can function in many different capacities. It may act as a translator, perform task allocation, or monitor progress, among other things. An intermediary which accepts task requests and allocates those tasks among its members is known as a broker or a facilitator. As part of the allocation, the broker may decompose the problem into more manageable subtasks. This allows agents to take advantage of all the capabilities of the (potentially changing) federation, without requiring knowledge of which agents perform a task or how they go about doing it. This reduces the complexity and messaging burden of the client, but also has the potential of making the broker itself a bottleneck [75] (a possibility common to all intermediaries). An intermediary acting as go-between among agents is known variously as a translator, embassy or mediator depending on its specific characteristics. Embassy agents provide a layer of security for members of their federation, by having the ability to deny communication requests. Mediator agents store representations of all related parties, reducing their individual complexity by providing a layer of abstraction. This capacity can be further exploited to arbitrate conflicts [120]. Intermediaries which provide the ability to track the state of one or more of its participants are known as monitors. For example, result information can be automatically propagated to interested parties. Of course, one or more of these roles may be combined into a single intermediary which offers several types of services.

## Formation

Singh and Genesereth in [177, 65] describe how a general federated system would work. All agents are expected to communicate using an Agent Communication Language (or ACL, a somewhat-generic term used by many researchers to describe their agents' communication protocol), which in this work is a combination of the first-order predicate calculus KIF with the KQML agent messaging language. Knowledge and statements sent between agents are encoded as KIF statements, which are then wrapped in KQML to provide a standard mechanism for specifying the sender, receiver, intent, and so forth. This provides a common language and set of behavioral constraints that will allow the various agents to interact. Not all agents must implement the entire class of concepts in the ACL, but the aspects they do use must be correct with respect to the ACL's specification. In addition, although they speak the same language, not all agents must use the same vocabulary to describe a particular situation, although to interact there must be an intermediary capable of translating the vocabularies. The system is initialized with a set of intermediaries called facilitators, which serve many of the roles outlined above, notably brokering. Agents connecting to the system start by sending their capabilities to the local facilitator. Implicit in this communication is the notion that the agent is willing to use those capabilities in service of requests posed by the facilitator. Needs are similarly routed to the facilitator, which then attempts to find other facilitators that can service that need. Each facilitator provides a yellow pages function which supports this search.



**Figure B.8.** A multi-agent marketplace.

Khedro’s Facilitators [99] and the jointly developed PACT project [34] have produced very similar systems that also use a common ACL and a community of intermediaries to produce a robust and dynamic task decomposition and allocation scheme among a group of heterogeneous participants.

The MetaMorph I [130] and II [167] architectures described by Maturana and Shen demonstrate a federated agent system for use in intelligent manufacturing. In this domain, agents are used to drive aspects of product design and manufacturing, contending with heterogeneous resources, dynamically changing conditions, and hard and soft constraints on behavior. MetaMorph’s name is derived from the fact that the system can continuously change shape, adapting to new conditions as they are perceived. This is accomplished in part through the use of intermediaries called mediators, which are responsible for brokering, recruiting and conflict resolution services. The recruiting service is similar to brokering, but is differentiated by the fact that the intermediary can remove itself from the relationship once the partners have been discovered. This weaker form of federation provides efficiency gains at the cost of less flexibility, both due to the loss of the layer of abstraction that exists in the brokered approach. The federations themselves are dynamically created in response to new task arrivals or requests from other groups using a contract net [178] approach, or are statically created from agents in a common subsystem (e.g. tools, workers, etc.).

## B.8 Markets

In a *market*-based organization, or *marketplace* as shown in Figure B.8, buying agents (shown in white) may request or place bids for a common set of items, such as shared resources, tasks, services or goods. Agents may also supply items to the market to be sold. Sellers (shown in black), or sometimes designated third parties called *auctioneers*, are responsible for processing bids and determining the winner. This arrangement creates a producer-consumer system that can closely model and greatly facilitate real-world market economies [204]. These latter systems fall into the more general category of *agent-mediated electronic commerce* [73]. Because of this similarity, a wealth of research results from human economics and business can be brought to bear on agent-based markets, creating a solid theoretical and practical foundation for creating such organizations [203, 205, 32].

Markets are similar to federated systems in that a distinguished individual or group of individuals is responsible for coordinating the activities of a number of



other participants. Unlike a federation, market participants are typically competitive. In addition, participants do not cede operational authority to those distinguished individuals, although they do trust the entities managing the market and abide by decisions they make. It is also common for markets to operate as open systems [204], allowing any agent to take part so long as it respects the system's specified rules and interface. As such, they share some of the benefits and drawbacks of societies.

When using the terms “buyer” and “seller”, one may implicitly assume that an artifact will eventually be transferred in exchange for some form of compensation [26, 191]. Although this paradigm is common, it is not always the case, and market-based organizations have been used in various projects to accomplish less obvious goals. For example, Wellman [201] proposes using a market-based approach to perform decentralized factory scheduling. In this work, each factory job is associated with a duration, deadline and value. The factory itself, acting as the seller, has a reserve price associated with the time slots it has available. Agents bid on a set of slots that have sufficient total time to satisfy the job duration and do not exceed the deadline, using the job value as a maximum bid price. Market forces will cause agents to seek out the most cost-effective time slots, while higher-valued jobs will naturally take precedence over lower ones. This should lead to an efficient allocation of (time) resources, while maximizing the factory's overall utility. Bussman [19] has developed an auction-based manufacturing control system with a similar purpose, where agents are used to represent workpieces, transportation conveyors and machines. In this work, machines bid for the right to work on workpieces, which act as sellers, by relating an expected time to completion. When a machine's bid is accepted, a series of additional negotiations between the workpiece and the conveyors move the piece to the appropriate location. Yet another example is the Mariposa distributed database system [183], which uses market-based techniques to optimize query processing. Individual nodes buy and sell fragments of information. Queries inserted into the system are associated with a bidding profile, indicating how much the user is willing to pay. A brokering process takes the query and requests bids from relevant nodes. who then submit bids in an effort to win the rights to process the query.

More generally, Wellman proposes the notion of *market-oriented programming* [203], which uses the marketplace paradigm as a general programming methodology that can efficiently address multi-commodity flow and resource allocation problems. His WALRAS framework that implements this concept has been used to create solutions for transportation logistics, product design and distributed information services. Many other marketplace frameworks have also been developed for general use [26, 149, 28, 29, 33]; Kurbel and Loutchko provide a comparative analysis of structure and function [104].

## Characteristics

Markets excel at the processes of allocation and pricing [205]. If agents bid correctly (i.e. make truthful bids according to their perceived utility gain if they win), the centralized arbitration provided by the auctioneer can result in an effective allocation of goods. The Kasbah system [26] is an example of an agent-based marketplace that

demonstrates many of the typical characteristics of this type of organization. Agents in Kasbah are segregated into two categories: buyers and sellers. Both types indicate the type of object they are interested in (buying or selling) with a feature vector, along with a desired price, a threshold price (lower or upper bound), and a negotiation strategy that controls how their offered price changes over time. A sale occurs when a seller's price matches what a buyer is willing to pay. The objects being sold in this system represent the targets of the allocation process, and the price is determined dynamically according to supply and demand. The mechanism that is employed in Kasbah corresponds to an intuitively fair way to allocate among competitors, at least from a self-interested point of view: all agents gradually compromise, and the agent willing to meet the seller's price first wins.

The behaviors embodied in a marketplace, namely the existence of buyers and sellers, a potential multitude of goods, and competition among participants, make such organizations intrinsically linked with the properties of *auctions*. Kasbah is an example of a *two-sided* auction, because both sides compromise. If one of the two parties maintained a fixed price, it would be *one-sided* auction. Many other types of auctions exist to service the different needs of different communities, each with their own characteristics [210, 104]. For example, in a *combinatorial* auction, participants bid on collections of goods, rather than single objects. In an *reverse* auction, sellers bid rather than buyers. In *sealed-bid* auctions, the participants do not see competing bids while the auction is in progress. In *continuous* auctions, a pool of items exist, exchanges occur as soon as two compatible bids are made, and the bidding process continues uninterrupted. The particular type of auction which is employed dictates the manner in which the participants interact. Much of the complexity involved in designing an effective market and marketplace agent revolves around understanding the subtleties of the auction's characteristics, and crafting an appropriate strategy based on that knowledge.

There are two drawbacks to market-based organizations. The first is the potential complexity required to both reason about the bidding process and determine the auction's outcome. The former computation may require a detailed approximation of competitors' beliefs, a practice known as *counterspeculation*, especially in single-shot or sealed bid auctions [191]. The latter computation, also known as *clearing* the trade, can be particularly difficult in the case of combinatorial auctions. This is known to be a NP-complete problem [154], although solutions have been devised that have good performance in practice [155]. The second is security; in addition to the practical network-related security issues inherent in any open system, one must also be able to verify the validity of the auction approach itself. For example, the bidding strategy used in the Kasbah system is vulnerable to a form of cheating known as *collusion*. If two or more bidders in the system agree to reduce their rate of compromise, they have a chance to artificially lower the final sale price. It is also important that the bidding process does not reveal information about the participants. For example, if a seller could determine the threshold prices of some of its buyers, it could simply wait until the maximum such price is reached, thereby artificially increasing the sale price. Some of these issues can be resolved by selecting an appropriate auction type. The Vickrey auction's structure [197], where the highest bidder wins but pays the second highest

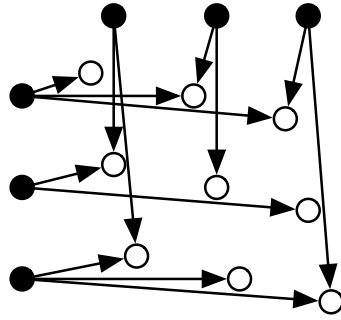
bid price, promotes truthful bidding and discourages counterspeculation. Enforcing anonymity and secure communication channels can also help avoid many common pitfalls.

## Formation

As is the case of many open systems, marketplaces are frequently static, pre-existing entities that do not require a formal creation process beyond starting the actual market process (if any) and allowing agents to connect. The well-known Trading Agent Competition market [202] operates in such a fashion, albeit for a limited amount of time. They may have certain barriers to entry, such as respecting a defined programming interface, implementing a particular transaction language, and respecting the rules of the market's auction type. These entry conditions are similar to those discussed earlier in the context of societies, although there is generally no formal negotiation or socialization process involved. Wellman [204] outlines a number of other practical characteristics that should be exhibited for a marketplace to be successful. They must maintain *temporal integrity*, meaning that the outcome of an auction depends on the arrival sequence of bids, and is independent of any delays internal to the market itself. Transactions performed by the market must be *atomic*, that is, they have no effect if they fail or are canceled prior to completion. As noted above, they also require attention to security risks, so that participant information is adequately protected and the auction process itself is kept safe from conventional attacks, particularly if there is an actual exchange of goods, information or currency in the market. Markets may also incorporate product discovery services, banking services, brokering middle-agents and negotiation support, to reduce the burden placed on the participants [191, 73].

Other works have explored dynamic formation of markets. As mentioned in Section B.5, Brooks has used the notion of congregations to dynamically form markets within a group of agents [16]. Recall that congregations are groups of agents which have banded together because of some common long-term interest or goal. In this work, that long term goal is the cost-effective exchange of goods or services. In a large population, it can be difficult to directly find suitable trading partners, and expensive to contact or broadcast to all possible partners. A suitably formed congregation serves to limit the scope of this search or broadcast, which in turn facilitates the marketplace creation.

A relatively new concept being exploited in both human [134] and agent [3, 58, 21] organization research is the *virtual organization* (VO). A virtual organization is one that has a fixed purpose (e.g., to provide a set of services) but a potentially transient shape and membership. The key characteristics of a VO are that they are formed by the grouping and collaboration of existing entities, and there is a separation between form and function that precludes the need to rigidly define how a behavior will take place. This provides flexibility in how a particular goal is satisfied, by allowing the system to adapt the set of participants to meet resource availability and service demand. The concept is similar to the coalition and congregation paradigms discussed earlier, and has many of the same benefits as a federation, although a virtual



**Figure B.9.** A matrix organization.

organization can generally be thought of as an entity in and of itself more so than an empty coalition or congregation.

The CONOISE project has explored the dynamic creation of virtual organizations within a larger marketplace environment [139]. In this context, the creation of a VO can be thought of as the creation of a new market entity (buyer or seller) from a group of existing participants. This can give those participants greater leverage, efficiency or reliability as they combine their producing or consuming power. The members of a VO may remain distinct when outside of the marketplace, but within the market they act as a single unit. For example, two producers might combine to offer a new joint product. Two consumers might combine to obtain greater buying power. In responding to bids, a VO will then be able to offer the union of services or goods over all its members. VOs may also split when the relationship is no longer beneficial or if levels of trust or reputation have been sufficiently degraded. In all cases, the shape of the market is affected as these changes are made, and thus the market as a whole will evolve over time based on the needs and capabilities of the participants, and the corresponding consolidation decisions they make.

## B.9 Matrix Organizations

We have seen that the strict hierarchical organization method is based on a tree-like structure of control. Agents or agent teams report to a single manager, which provides the agents with goals, direction and feedback. *Matrix organizations* relax the one-agent, one-manager restriction, by permitting many managers or peers to influence the activities of an agent. This forms a mixed-initiative environment, where successful agents reason about the effects their local actions can have on multiple entities. This is in some sense a closer approximation to how humans exist. A person may receive guidance or pressures from their manager, co-workers, spouse, children, colleagues, etc. Even in a purely business setting one might have to report to an immediate supervisor, project managers, vendors, and peers at cooperating

businesses. Interrelationships can come from many directions, each with its own objectives, relative importance and pertinent characteristics [199].

The term matrix organization comes from a grid based view of the participants. One can place managers (black) around a group of “worker” agents (white), and use a directed edge to indicate authority, as in Figure B.9. Alternately, agents are the rows and managers the columns (these sets may overlap), and a check is used to denote where an authority relationship exists. Like the hierarchy’s tree, the matrix provides a graphical way to depict which managers can influence the activities of each agent.

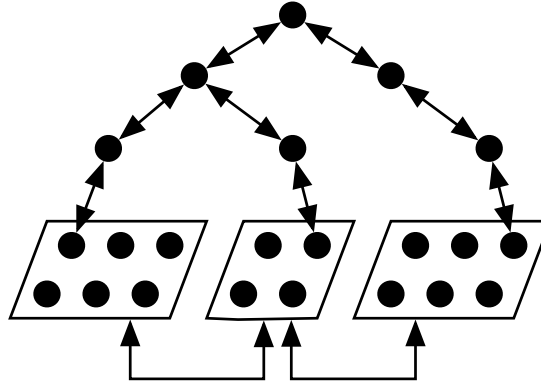
## Characteristics

Matrix organizations provide the ability to explicitly specify how the behaviors of an agent or agent group may be influenced by multiple lines of authority [43]. In this way, the agent’s capabilities may be shared, and the agent’s behaviors (hopefully) influenced so as to benefit all. This is particularly important if the agents themselves are viewed as a functional, limited resources. For example, if a particular skill is needed by two separate tasks, the agent can be used to address both, provided it has sufficient computational power. In the case where the agent has multiple ways of performing a task, it can also choose the method which best satisfies its peers.

This sharing come as a price, however, because the shared agent becomes a potential point of contention. If its managers disagree, the agent’s actions may become dysfunctional as it is pulled in too many directions at once [161, 150]. To operate effectively, the agent must have a commitment ranking mechanism and sufficient autonomy to resolve local conflicts, or the ability to promote conflicts to a higher level where they may be resolved [121]. Wagner’s *motivational quantities* framework [199] is one approach that addresses this problem. In that work, task valuation is performed by combining both the local intrinsic worth of the task with the perceived or specified worth that task will have on other entities. This valuation is quantified through the expected production and consumption of different motivational quantities (MQs), which act as a virtual resource or medium of exchange. The preference for particular MQs is specified with a set of utility curves that together determine the agent’s overall utility. By coupling the production of different types of MQs with the tasks associated with different managers, the framework is able to capture the quantitative motivation behind a particular course of action. This explicitly represents the type and state of the relationships the agent has with those managers, which can enable it to correctly balance its behavior in a matrix organization.

## Formation

Decker [43] describes the MACRON organizational architecture, in which agents form a matrix organization. The domain for their system is cooperative information gathering, where multiple agents search for relevant data in response to a user’s query. Individual agents are separated into predefined functional groups that contain agents able to access a particular type of information. These groups are under the control of a functional manager, who assigns agents to query tasks as they arrive. User query



**Figure B.10.** A compound organization.

agents generate those query tasks, and therefore use the functional managers to dynamically select agents to satisfy their own goals. Individual gathering agents report to two agents: a static functional manager, and a query manager which changes depending on the user's actions. This has the effect of assigning the minimal needed set of agents to the query, increasing efficiency when compared to a system employing a set of static teams where particular team members might go unused, depending on the query characteristics. At the same time, this approach uses fewer resources than one lacking functional groups, which would have to search through all available agents for each query.

In [86], Horling and Mailler describe a distributed sensor network application where a matrix organization is used to address a resource allocation problem. In this case, the sensors themselves were limited resources, since their heterogeneous locations and orientations made each one unique. The tracking process for each target was controlled by a different track manager, which was responsible for discovering and coordinating with the sensors needed to track its target. When multiple targets came in close proximity to the same sensor, a matrix organization is dynamically formed as the relevant managers interact with that sensor. At the same time, that sensor may have previously been given tasks by a regional manager responsible for detecting new targets. The result is an individual which may be under contention by three or more managers, and which must then decide how best to meet those demands. This was done using a combination of a predefined ranking scheme (tracking has higher priority than scanning for new targets), local autonomy (round robin scheduling) and conflict elevation (track managers negotiate directly once aware of the conflict).

## B.10 Compound Organizations

Not all organizational structures fit neatly into a particular category, and some architectures may include characteristics of several different styles. A system may have one organization for control, another for data flow, a third for discovery, and so on.

For example, Durfee’s PGP [53] incorporates one organization for interpretation, and another separate structuring of the same agents to manage coordination problems. Compound organizations can be overlapped, operating as virtual peers at the same conceptual level, or be nested, so that some subset of agents in a group are organized in a potentially different way within the larger context. A sample such organization is shown in Figure B.10, which combines a hierarchy with a set of coalitions. As with singular organizations, they may be created or adapted over time, or they may be instantiated as part of a transient form while a population shifts between organizational styles. Ideally, these compound architectures can use the most effective structure for the particular goal at hand, without limiting options that might be used elsewhere in the system. The tradeoff in this situation is usually one of complexity. Because an individual agent might take on different roles in response to different organizational demands, the agent itself must have sufficient sophistication to act efficiently and asynchronously in all those roles.

Some of the organizational paradigms which have been discussed so far are more amenable to coexistence than others. In much of the teamwork research, for example, a loose hierarchy of control was created among the agents after the team had formed [185, 189]. Hierarchical structures for interpreting and consolidating raw data are also a popular mechanism for handling scale that can augment a preexisting or lower-level structure [213]. Societies frequently have an internal organizational structure within the larger context defined by the social laws and norms [44, 49]. In other cases, researchers have exploited the characteristics of one type of organization to create another. Congregations, for example, have been used to facilitate the dynamic formation of markets [16], while both markets [108] and hierarchies [2] have been used to efficiently create coalitions. Societies can also be viewed as a common “pool” of agents, from which a range of other organizations can be constituted. In this type of compound organization, the society may exist in support of other, more dynamic structures created to address particular tasks [169]. This begins to touch on the notion of organization longevity, which will be addressed in Section B.12.

## Characteristics

The positive and negative characteristics of a compound organization are derived primarily from its constituent parts. However, the interplay between organizations can lead to unexpected consequences. For example, if the distinguished intermediary in a federated system plays a key role in a separate overlay organization, it may be unable to fulfill both roles adequately. Similar to a matrix organization, agents may be faced with conditions where it is not clear which of two competing objectives it should satisfy [150]. Conversely, its knowledge of the requirements of both organizations may enable it to make more globally effective decisions. The possible interactions and formation strategies among arbitrary coexisting organizations are difficult to characterize in a general manner; instead we will proceed with a discussion of example systems employing this technique.

## Example Compound Organizations

The distributed sensor network solution described by Horling and Mailler [86] uses several different overlapping organizational techniques. Agents are first partitioned into federations, called sectors, where membership is based on their geographic proximity. A distinguished member of each group is given the role of sector manager, who provides a form of recruiting service to other agents in the environment. This recruiting service supports the activities of track managers, who must discover and use the appropriate sensors as part of their tracking task. In forming the federations, the search time is reduced because only a subset of the population (the sector managers) needs to be interacted with, and communication requirements requirements are reduced because only the necessary subset of sensors will be returned. As discussed in Section B.9, both the sector and track managers provide tasks to individual sensors, forming a matrix organization in the process. This arrangement facilitates resource sharing by allowing the sensors to guide their local activities based on the needs of potentially several interested parties, but can also lead to conflicts caused by over-demand. Because the sensor is a finite resource, a cloning technique like the one discussed in Section B.1 cannot be used to address the conflict. Instead, a loose peer-to-peer relationship between track managers allows them to negotiate directly, alleviating the conflict through demand relaxation or by using alternate sensors. This resource allocation scheme employs a second, weaker form of federation through its use of mediators [120]. The conflicts, which may be potentially multi-linked and far-reaching, are partially centralized by a mediator agent which acts on the part of the relevant agents to find a suitable solution. In [84] the quantitative effects of these interactions are demonstrated through a set of experiments that vary the shape of the organizational structure.

Yadgar [213] describes a different approach in a distributed sensor environment. Groups of geographically-related sensors are first formed into sampler groups, which are essentially federations with a single agent called the sampler group leader acting as the intermediary. These groups then form the lowest level of a data aggregation hierarchy that exists above them. This arrangement is similar to the example organization shown in Figure B.10. The sampler group leader collects raw data from the members of its group, and passes the data to its parent agent in the hierarchy, known as a zone leader. It is this zone leader's responsibility to interpret the sensor data to the best of its ability, by building motion equations and combining data perceived to be from the same target. This more abstract view is then passed to the next level of the hierarchy, where the process repeats. This will eventually terminate at the apex agent which should be able to reconstruct a global view from the abstract pieces it receives. The hierarchy itself is strict, and communication is only permitted between connected agents, which reduces the level of sophisticated needed by the agents. The experimental results showed that this solution could scale to thousands of sensors and targets. The tradeoff they discovered was that shorter hierarchies produced more accurate results, because the fragmentation of the area was minimized, which in turn reduced the number of fusion processes data must survive before it is incorporated. Conversely, taller hierarchies dramatically reduced the computational load placed on



any one agent, because the area each agent was responsible for became relatively small. By weighing these characteristics against the domain requirements one can select an appropriate structure to use.

## B.11 Other Organizational Topics

In this survey we have focused entirely on particular organizational paradigms. However, there are a number of other topics related to organizational design which we will not cover in detail, but are sufficiently important to warrant mention. These are outlined below:

1. *Global Organizational Representation* Implicit in the concept of an intentional organizational design is an explicit representation of its structure. This is of use to designers, as a means of specification and exploration, and to the agents themselves, as a template and diagnostic tool. A number of general modeling representations have been proposed, notably by Fox [59], Tambe [187], Hübner [88], Pattison [143], Dignum [49], Sims [174], Horling [81] and Vázquez-Salceda [196].
2. *Local Organizational Representation* The organization's global view is not always the most appropriate vehicle to guide agents' behaviors. It can be too coarse in granularity, too qualitative or simply too large to be of practical use. Agents require a well-defined, quantitative mechanism that can be used to select appropriate local actions while respecting global organizational specifications. This process was originally described as *local elaboration* by March and Simon [124], where the activities performed by an agent are first constrained by its position in the organization, and then selected using local information and capabilities. The social consciousness model suggested by Glass and Grosz [67], Decker's TÆMS language [40], Shoham's social laws [168], and Wagner's MQ framework [199] provide ways to accomplish this.
3. *Organizational Performance* Other researchers have taken a different approach by creating formal analytic or statistical models that focus on the activities or behaviors of the organization, rather than representing the organization as a whole [123, 39, 132, 179, 107, 166, 68, 81, 160]. These typically more quantitative representations can provide insights into organizational performance that are largely absent from purely descriptive or logical representations. A different approach is to use experimental or simulation studies, which can offer a more general-purpose approach to analyze organizational performance that may not be amenable to modeling [114, 117, 171]. The drawback to using empirical analysis is the time required to run such tests, which is usually much greater than that needed for analytic techniques. Conversely, analytic models may require simplifying assumptions to be tractable, or otherwise fail to take into account the complexity real-world behaviors. Parunak [142] provides further discussion on the tradeoffs between these approaches. However they are obtained, such

predictions can play a critical role in the search and evaluation process, by allowing the designer to directly compare alternative organizational strategies before implementing a design. This can provide the foundation for a more prescriptive organizational tool.

4. *Generative Paradigms* In each section, we have presented different ways in which organizations may be formed. We have not, however, presented a unified discussion of specific generative paradigms – a classification of the techniques that may be used to produce organizations. These may be broadly separated into at least three classes: scripted, controlled and emergent. The first includes organizations that are produced from statically predefined instructions, possibly from an external third party or during start-up. The second includes those that are explicitly applied to a population by an individual or group of individuals in response to perceived conditions. The third captures techniques which have no central or global direction, but are instead self-directed or grown organically through the individual actions of agents. In practice, it may be difficult to clearly classify particular techniques. For example, congregations emerge from individual agent decisions using the technique described by Brooks [17]. However, the fact that it uses heuristics intended to simulate a controlled decision, along with agents which provide labels to guide the formation, gives the appearance of a controlled process.
5. *Organizational Adaptation* Although we have briefly touched on adaptation previously, an organization’s ability to adapt is a general concept that is critical in any dynamic environment. The organization must have the ability to detect and react to changes in a timely manner in realistic, open domains [22, 6, 78]. Any organizational change which occurs at runtime will have associated costs. These costs may be observed in direct consumption of resources, such as bandwidth or processing power, or indirectly because of inefficiencies or opportunities missed while in an intermediate state. The ability to adapt an organization depends on first recognizing potential problems, evaluating the costs and benefits of candidate solutions, and then implementing the selected changes. Related to adaptation is the notion of social pathologies, which occur when an organization adapts inappropriately [192, 94].
6. *Coordination and Negotiation* Many of the organizational styles that we have covered assume some that some sort of interaction or coordination will take place between agents. This is seen in the authority relationships of hierarchies, the joint intentions of teams, data routing protocols in federations, and negotiations of society members. The characteristics provided by these interactions are critical to the effective qualities of these paradigms. For example, aggregating nodes and managers in hierarchies and intermediaries in federations frequently take on responsibilities related to coordination, by assigning tasks or routing information in such a way that interrelationships among their subordinates can be avoided [62]. Argumentative negotiation has been shown to be effective in

resolving conflicts in team settings [97]. The techniques that are used can heavily influence the interactions and behaviors exhibited by the group, ultimately affecting the performance of the organizational structure. Work by Prasad [144], Lesser [109] and Toledo [54] have also explored the dynamic selection of coordination strategies, which in this context can be considered a form of organizational adaptation.

7. *Autonomy* The manner in which an agent behaves, and in particular how its motivations are determined, is intimately related to its position within the organization. Agents may be externally directed, self-directed or some combination of the two [113]. For example, we have seen that agents in hierarchies, federations and matrix organizations all generally have manager-supervisor relationships, implying that local actions are partially or completely decided by an external entity. Conversely, agents operating in markets are typically more autonomous, independently deciding how and when to bid. Like other characteristics, the level of autonomy can affect the performance of the system as a whole. Authoritarian structures can exploit centralization to make good decisions, while an organization of more autonomous entities offers better balance and parallelism. Because the needs and constraints exhibited by participants change over time, it can also be beneficial to dynamically adapt agents' levels of autonomy in response to changing events [6, 158, 217].
8. *Human Organizational Analogues* For much of the time that multi-agent organizations have been researched, attempts have been made to draw upon the large body of work that has been done on human organizations. The fields of sociology, anthropology, biology, economics, business management and formal organization theory (among others) contain a wealth of analytic and case study information describing how human organizations are structured and perform [61, 64]. Although on the surface much of this work is intimately tied to the human experience, attempts to extract concepts and abstractions have met with some success.
9. *Diversity* Although role assignment clearly plays a critical role in an organizational specification, the notion of agent diversity is rarely treated as or reasoned about as a first-class characteristic. As with stock portfolios, animal populations and security techniques, diversity can play an important role in agent systems susceptible to failure. Enforcing agent diversity through heterogeneous roles, agent types or division of labor, can impart semantic and capability fault-tolerance on the system as a whole [31, 147, 32, 118]. Diversity can be embedded in the organizational design to encourage such characteristics.

## B.12 Discussion

In this article we have presented a number of methods by which a multi-agent system could be organized. A brief comparison of the potential benefits and drawbacks of

Paradigm	Key Characteristic	Benefits	Drawbacks
Hierarchy	Decomposition	Maps to many common domains; handles scale well	Potentially brittle; can lead to bottlenecks or delays
Holarchy	Decomposition with autonomy	Exploit autonomy of functional units	Must organize holons; lack of predictable performance
Coalition	Dynamic, goal-directed	Exploit strength in numbers	Short term benefits may not outweigh organization construction costs
Team	Group level cohesion	Address larger grained problems; task-centric	Increased communication
Congregation	Long-lived, utility-directed	Facilitates agent discovery	Sets may be overly restrictive
Society	Open system	Public services; well defined conventions	Potentially complex, agents may require additional society-related capabilities
Federation	Middle-agents	Matchmaking, brokering, translation services; facilitates dynamic agent pool	Intermediaries become bottlenecks
Market	Competition through pricing	Good at allocation; increased utility through centralization; increased fairness through bidding	Potential for collusion, malicious behavior; allocation decision complexity can be high
Matrix	Multiple managers	Resource sharing; multiply-influenced agents	Potential for conflicts; need for increased agent sophistication
Compound	Concurrent organizations	Exploit benefits of several organizational styles	Increased sophistication; drawbacks of several organizational styles

**Figure B.11.** Comparing the qualities of various organization paradigms.

each strategy is summarized in Figure B.11. A more complete depiction of the range of relevant organizational characteristics in general has been compiled by Carley and Gasser [23], while Malone and Smith [123] provide a focused comparison of the characteristics of hierarchy and marketplace designs. It should be clear from this discussion that no single approach is necessarily better than all others in all situations. The selection made by a designer should be dictated by the needs imposed by the system's goals, the resources at hand, and the environment in which the participants will exist. That said, if one looks at the depth of available research and how frequently their concepts have been applied, it is the case that hierarchical, team-centric, coalition-based organizations and marketplaces have proved to be most popular among multi-agent researchers. These four paradigms seem to offer the most in terms of flexibility, ease of implementation and their innate ability to produce demonstrable, positive effects. Hierarchies are effective at addressing issues of scale, particularly if the domain can be easily decomposed along some dimension. Teamwork can be critical when working on large-grained tasks that require the coordinated capabilities of more than one agent. Coalitions allow agents to take advantage of economies of scale, without necessarily ceding authority to other agents. Markets take advantage of competition and risk to decide allocation problems in a fair, utility-centric manner. We also feel that if the broad vision of an agent-connected or agent-facilitated world that many proponents of multi-agent technology describe is to be realized, many of the characteristics of the agent society paradigm must be incorporated [64].

A popular approach not mentioned thus far is the (sparsely) connected graph structure, sometimes called a *network organization* or *adhocracy* [195, 13], where agents interact because of particular role-based requirements but no overarching design principle is explicitly applied. The connection pattern superficially resembles a team, but without a team's strong interaction semantics. Some aspects of the structure may be statically defined, but a more emergent, dynamic construction is more typical. If there is an absence of explicit control over the organizational structure, the set of interactions may change in response to every newly recognized goal. The network design is also a common basis for compound organizations in a manner similar to societies, where individual entities in the network are entire sub-organizations. These approaches can be effective and cost-efficient, but as the environment scales or the agent population becomes more dynamic a more structured organization can provide additional framework to address the more demanding context. Corkill and Lander [32] enumerate several other factors which motivate the need for explicit organization, including scarce resources, the potential for collaboration and the amount of repetition of work.

Other conditions may in fact preclude the use of particular paradigms. For instance, it can be difficult to generate optimal coalition or congregation structures when there is either limited time or a large population. When individual agent resources are constrained, particular instances of organizations which suffer from bottleneck effects, such as hierarchies, federations and holarchies, can become inefficient. We have also previously noted how some types of structures, such as matrices, societies, and certain compound organizations, require a somewhat higher level of sophistication of

the participating agents. As above, the operating context will guide, or in this case restrict, the choice of organizational design.

As research progresses in these areas, typically by adding features and relaxing assumptions, it can become difficult to precisely categorize a particular approach. For example, we noted how hierarchies and holarchies are closely related, as are coalitions and congregations. To a certain extent, we have focused on the extreme or most constrained examples of organizations in this paper to better delineate discrete classes, and it is frequently the case that the “rules” of a particular paradigm as we have presented them have been broken in an attempt to broaden its abilities or applicability. While this might frustrate one’s attempt at categorization, our opinion is that the convergent evolution of these strategies towards a common form lends additional credence to the applicability of that form.

A somewhat more elusive goal is to define what exactly constitutes an organization in general. At what level of abstraction in the system’s design should the influence of the organization diminish and more transient “operational” decisions become more important? Must a structure exist for some period of time or some number of iterations before it is considered an organization? We have looked at strategies that are generally short-lived, such as coalitions, while societies may outlast the lifetime of any of its participants. Teams may exist to satisfy only a single goal, while federations see a continuous stream of different tasks. In each of these cases, the pattern of interactions between the agents is a defining characteristic, influencing the behaviors and qualities exhibited by the system. If this same pattern exists in two different circumstances, is one an organization and the other not? To a certain extent, this is just a matter of semantics, and we could just as easily name it a “pattern of interactions” and leave it at that. However, maintaining a broad and flexible concept of organization allows one to more easily recognize that commonalities may exist between these architectures. In particular, characteristics observed in superficially different circumstances may be derived specifically from these interactions. Thus, we propose that under all circumstances this pattern can be interpreted as an organizational design. The fact that it may exist for a single moment or a single task certainly impacts its performance and construction, but much of the underlying purpose and qualities of the structuring remain the same, and should be recognized as such.

Whatever they are called, the type of short and long term patterns of interaction we have described in this article will become increasingly important as multi-agent technology is used to address more complex, real-world problems. Scale, real-time constraints and bounded rationality all conspire to create challenging environments to operate in. Because of their ability to regulate the increased complexity of the local problem solving process required in such domains, organizations should be a critical part of any comprehensive, multi-agent solution. By recognizing and understanding organizational paradigms such as those we have presented, we hope that the use of explicit organizational design is encouraged and facilitated.

## APPENDIX C

### DISTRIBUTED SENSOR NETWORK ODML MODEL

This appendix contains the complete textual definition of the ODML model created for the distributed sensor network domain introduced in Chapter 2 and described more completely in [110]. The graphical template first shown in Figure 2.10a is repeated above for convenience. The code itself is commented to facilitate inspection, and I will provide a brief outline of its contents below.

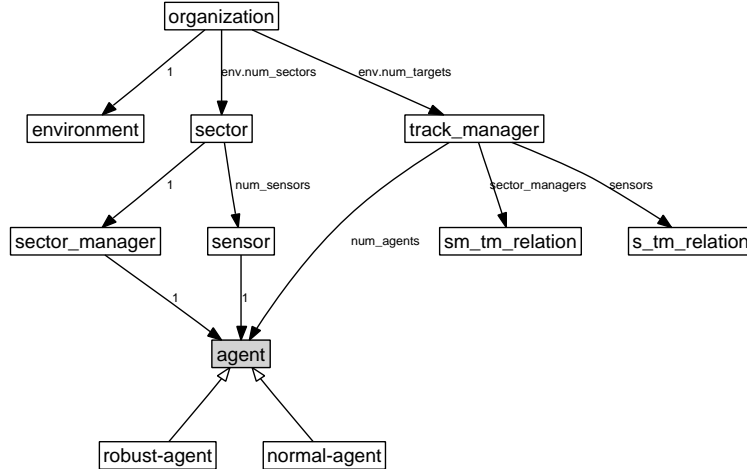
The code itself is divided into several different sections. The enclosing *organization* provides the structure in which the other node definitions exist. It begins by defining the high-level has-a relationships, which are nominally the root entities of the working organization. These consist of the *environment*, a set of *sectors* and a set of *track\_managers*. The high level *utility* calculation is also contained here, which is simply the negative average RMS error predicted by the model. The bulk of the remaining fields in *organization* are a set of constants that gather a range of organizational statistics for logging purposes. They are not used elsewhere.

Following the *organization* is a series of node definitions, beginning with *environment*. It is here that the *num\_targets* and *sensors\_per\_sector* variables are defined, encoding two important organizational choices that must be made during instantiation. The remainder of *environment* consists of a number of constants defining the shape of the sensor area and characteristics of the sensors and activities entities can perform. These could be placed elsewhere (in the *sensor* node, for example), but are kept in *environment* just to combine all such external factors in one place.

The *sector* node follows. It provides two functions of interest. The first is its structural importance, as the root for a sector manager and a set of sensors. The second is the *effective\_area*, the constant used to specify the dimensions of the sector itself. As described in Section 2.3.1, a map function is used to determine this characteristic on a case-by-case basis.

The *sector\_manager* is the first role seen in the model. It is fairly simple, containing a has-a relationship with the *agent* that will take on the role, an is-a relationship with *entity* to acquire a set of common characteristics, and a set of modifiers used to pass communication load values to the agent. The communication values themselves are initialized to zero in *entity*, and are determined exclusively by the relationships the manager will take part in.

The *sensor* node also has a has-a relationship with *agent*, an is-a relationship with *entity*, and a set of modifiers to disseminate its communication load. In addition, it has a set of constants that are used to determine the sensing tasks it will perform. Its *requested\_measurement\_rate* is initialized to zero, and will be affected by the relationships it forms with track managers. The requested rate, along with the maximum



**Figure C.1.** Graphical view of the ODML DNS model.

possible sensing rate, is used to determine the *actual\_measurement\_rate*. A scaling factor *actual\_measurement\_ratio* is also calculated here, which is used to determine how much of the sensor’s efforts will go to each track manager.

The *track\_manager* role comes next, with similar relationships to *agent* and *entity* that the previous two roles have. Unlike those two roles, the has-a relationship with *agent* can have a non-unary size *num\_agents*. As described in Section 2.3.5, role migration is modeled by applying the effects of the role to one or more agents. A set of constants is used to estimate this *num\_agents*, which is used to specify the magnitude of the has-a relationship and to scale the communication load values passed on to those agents. Has-a relationships are also created to a set of *sm\_tm\_relations* and *s\_tm\_relations*, to model the track manager’s relationships to the available sector managers and sensors, respectively. Following this, a set of constants determine the *requested\_sensors*, which is used to determine the *requested\_measurement\_rate*. The *actual\_measurement\_rate*, which is initialized to zero, is determined by the aggregate effects of the track manager’s relationships. This is then used to calculate the *rms* error, which is estimated with a function derived from empirical data.

The first type of track manager relationship, the *s\_tm\_relation* that binds it to a *sensor*, is used to propagate the demand of the manager to the sensor in question. The specific *track\_manager* and *sensor* are themselves passed in as parameters during the initialization process. The relationship uses those references to first determine the requested measurement rate, and to pass that value to the sensor with a modifier. The actual measurement rate created by the relationship is calculated by scaling the requested measurement rate by the actual measurement ratio computed by the sensor. This is then provided to the track manager with another modifier. The communication load and messages types needed to support these acts are also calculated here, and passed along with modifiers.

The *sm\_tm\_relation* connects a track manager to a sector manager. The communication load of the sector manager is used to estimate a *directory\_delay*. This value can



reduce the requested measurement rate of the track manager as described in Section 2.3.7. Like the previous relationship, the messaging totals needed to support this relationship are also calculated here.

The node definitions for *agent*, *normal\_agent*, and *robust\_agent* follow. These contain default values the number of sensors under control and the number of roles taken on. It also has an is-a relationship with *entity*. As described in Section 2.3.4, these are used primarily to aggregate and store the effects of multiple role assignments. Constraints put in place here also control the number of sensors that may be controlled.

The final node definition is for *entity*, which is a simple base node that is extended by many of the nodes described above. It provides default values for the various message loads covered by the model, as well as a *communication\_load* constant that combines them into a single representative metric. This node is of little significance in the model, and has been omitted from previous descriptions for clarity.

The textual model itself follows below. Text surrounded by the normal XML comment syntax (`<!-- comment -->`) are inline comments.

```
<?xml version="1.0" encoding="UTF-8"?>
<organization name="sensor_organization">
  <!-- Members -->
  <has-a name="env">environment</has-a>
  <has-a name="sectors" size="env.num_sectors">sector(this,env)</has-a>
  <has-a name="trackers" size="env.num_targets">track_manager(this,env,sectors)</has-a>

  <!-- A few convenience values -->
  <constant name="sensor_density">forallavg(sectors.sensor_density)</constant>
  <constant name="total_sensors">forallsum(sectors.num_sensors)</constant>
  <constant name="total_targets">env.num_targets</constant>

  <!-- Calculate utility -->
  <constant name="average_rms">forallavg(trackers.rms)</constant>
  <constant name="utility">-1 * average_rms</constant>

  <!-- Some statistics to log -->
  <constant name="agents">unique(trackers.agent, sectors.manager.agent, sectors.sensors.agent)</constant>
  <constant name="stddev_agent_comm_load">forallstddev(agents.communication_messages)</constant>
  <constant name="total_agent_comm_load">forallsum(agents.communication_messages)</constant>
  <constant name="sector_managers">unique(sectors.manager)</constant>
  <constant name="sensors">unique(sectors.sensors)</constant>
  <constant name="roles">unique(trackers, sector_managers, sensors)</constant>
  <constant name="message_update">forallsum(roles.message_tb) + forallsum(roles.message_rb)</constant>
  <constant name="message_directory">forallsum(roles.message_drq) + forallsum(roles.message_drr)</constant>
  <constant name="message_measurement">forallsum(roles.message_rr)</constant>

  <log name="RMS">average_rms</log>
  <log name="data" file="organization.dat" append="true">[env.sensors_per_sector, env.num_sensors, env.num_targets, average_rms, stddev_agent_comm_load, total_agent_comm_load / 1000, message_update * env.time / 1000, message_directory * env.time / 1000, message_measurement * env.time / 1000]</log>
  <log name="data" file="directory.dat" append="true">[env.sensors_per_sector, forallavg(sector_managers.communication_load), forallavg(trackers.directory_delay), average_rms]</log>
</organization>
```

```

<log name="messages">agents.communication_messages</log>
<log name="organization_dot" file="results/organization.dot">todot(this)</log>
<log name="organization_dot" file="results/organization-SS-NT.dot">todot(this)</
log>
<log name="organization_template_dot" file="results/organization-template.dot">
todot(this.template)</log>
<log name="organization_depdot" file="results/organization-dependencies.dot">
todepdot(this.template)</log>

<!-- Environment -->
<node type="environment" name="environment" size="1">
  <!-- Organizational decisions, these control how many targets -->
  <!-- there will be and the size of the sectors -->
  <variable name="num_targets" decision="NT">1,2,4,6,8,12,24</variable>
  <variable name="sensors_per_sector" decision="SS">1,2,4,9,18,36</variable>

  <!-- Number of entities in the environment -->
  <constant name="num_sensors">36</constant>

  <!-- Area values -->
  <constant name="width">90</constant>
  <constant name="height">90</constant>
  <constant name="area">width * height</constant>
  <constant name="sensor_radius">20</constant>
  <constant name="num_sectors">num_sensors / sensors_per_sector</constant>

  <!-- Behavior constants -->
  <constant name="slot_length">900</constant>          <!-- ms -->
  <constant name="measurement_duration">900 / 1</constant>  <!-- ms /
measurement -->
  <constant name="measurement_rate">1 / slot_length</constant>  <!-- Rate that
measurements are requested -->
  <constant name="update_rate">1 / 2000 * 1.3</constant>      <!-- Rate that
SMs are given target updates -->
  <constant name="time">143840</constant>

  <log name="area">area</log>
  <log name="sensors_per_sector">sensors_per_sector</log>
</node>

<!-- Sector -->
<node type="sector" name="sector">
  <param>organization:org, environment:env</param>

  <!-- Populate the sector with a manager and sensors -->
  <has-a name="manager">sector_manager(org,env,this)</has-a>
  <has-a name="sensors" size="num_sensors">sensor(org,env,manager)</has-a>
  <constant name="num_sensors">env.sensors_per_sector</constant>

  <!-- Determine the dimensions of the sector -->
  <constant name="actual_area">env.area / env.num_sectors</constant>
  <constant name="actual_coverage">actual_area / env.area</constant>
  <constant name="influence_range">env.sensor_radius</constant>
  <constant name="influence_range">11</constant>
  <constant name="effective_area">map(env.num_sectors,
1, env.area,
2, env.width * (env.height / 2 + influence_range),
3, (1 / env.num_sectors) *
(2 * env.width * (env.height / 3 + influence_range) +
1 * env.width * (env.height / 3 + influence_range * 2)),
4, (env.width / 2 + influence_range) * (env.height / 2 + influence_range),
9, (1 / env.num_sectors) *
(4 * (env.width / 3 + influence_range) * (env.height / 3 + influence_range)
+
4 * (env.width / 3 + influence_range * 2) * (env.height / 3 +
influence_range) +
(env.width / 3 + influence_range * 2) * (env.height / 3 + influence_range *
2)),

```

```

18, (1 / env.num_sectors) *
  (4 * (env.width / 3 + influence_range) * (env.height / 6 + influence_range)
  +
  2 * (env.width / 3 + influence_range * 2) * (env.height / 6 +
  influence_range) +
  8 * (env.width / 3 + influence_range) * (env.height / 6 + influence_range *
  2) +
  4 * (env.width / 3 + influence_range * 2) * (env.height / 6 +
  influence_range * 2)),
36, (1 / env.num_sectors) *
  (4 * (env.width / 6 + influence_range) * (env.height / 6 + influence_range)
  +
  8 * (env.width / 6 + influence_range * 2) * (env.height / 6 +
  influence_range) +
  8 * (env.width / 6 + influence_range) * (env.height / 6 + influence_range *
  2) +
  16 * (env.width / 6 + influence_range * 2) * (env.height / 6 +
  influence_range * 2))
) </constant>
<constant name="effective_coverage">effective_area / env.area</constant>
<constant name="sensor_density">num_sensors / actual_area</constant>

<log name="actual_area">actual_area</log>
<log name="effective_area">effective_area</log>
</node>

<!-- Sector Manager -->
<node type="sector_manager" name="SM">
  <param>organization:org, environment:env, sector:sector</param>

  <is-a>entity</is-a>

  <!-- Role-Agent relationship -->
  <has-a name="agent" discriminator="roles,sensors_controlled" decision="new{
    normal-agent">agent(env)</has-a>

  <!-- Constants -->
  <constant name="role">"M"</constant>
  <constant name="target_updates">0</constant>

  <!-- Characteristics passed to the bound agent -->
  <modifier name="agent.message_rr" op="+">message_rr</modifier>
  <modifier name="agent.message_tb" op="+">message_tb</modifier>
  <modifier name="agent.message_rb" op="+">message_rb</modifier>
  <modifier name="agent.message_drr" op="+">message_drr</modifier>
  <modifier name="agent.message_drq" op="+">message_drq</modifier>
  <modifier name="agent.roles" op="+">1</modifier>
</node>

<!-- Sensor Manager -->
<node type="sensor" name="S">
  <param>organization:org, environment:env, sector_manager:manager</param>

  <is-a>entity</is-a>

  <!-- Role-Agent relationship -->
  <has-a name="agent" discriminator="sensors_controlled">agent(env)</has-a>

  <!-- Role constants -->
  <constant name="role">"A"</constant>
  <constant name="radius">env.sensor_radius</constant>
  <constant name="area">3.14 * radius^2</constant>

  <!-- Calculate actual measurement performance -->
  <constant name="requested_measurement_rate">0</constant>
  <constant name="measurement_duration">env.measurement_duration</constant>
  <constant name="requested_measurement_duration">requested_measurement_rate *
    1000 * measurement_duration</constant>

```

```

<constant name="actual_measurement_duration">min(requested_measurement_duration
, 1000)</constant>
<constant name="actual_measurement_rate">actual_measurement_duration / (
measurement_duration * 1000) </constant>
<constant name="actual_measurement_ratio">actual_measurement_rate /
requested_measurement_rate</constant>

<!-- DR Messages -->
<modifier name="message_drr" op="+">2 / env.time</modifier>

<!-- Characteristics passed to the bound agent -->
<modifier name="agent.message_rr" op="+">message_rr</modifier>
<modifier name="agent.message_tb" op="+">message_tb</modifier>
<modifier name="agent.message_rb" op="+">message_rb</modifier>
<modifier name="agent.message_drr" op="+">message_drr</modifier>
<modifier name="agent.message_drq" op="+">message_drq</modifier>
<modifier name="agent.sensors_controlled" op="+">1</modifier>
<modifier name="agent.roles" op="+">1</modifier>

<log name="req_rate">requested_measurement_rate * 1000</log>
<log name="ac_rate">actual_measurement_rate * 1000</log>
</node>

<!-- Track Manager -->
<node type="track_manager" name="TM">
  <param>organization:org,environment:env,[sector]:sectors</param>

  <is-a>entity</is-a>

  <!-- Agent(s) the role is bound to, and the relationships it has with other
  entities -->
  <has-a name="agent" size="num_agents" discriminator="roles" decision="roles
=1.0,roles=2.0,roles=3.0,roles=4.0,roles=5.0,roles=6.0">agent(env)</has-a>
  <has-a name="sm_relations">forall(sm, sector_managers):sm_tm_relation(org, this
, sm)</has-a>
  <has-a name="s_relations">forall(s, sensors):s_tm_relation(org, this, s)</has-a
  >

  <!-- Role constants -->
  <constant name="role">"T"</constant>
  <constant name="update_rate">env.update_rate</constant>
  <constant name="velocity">1 / 1000</constant>
  <constant name="sector_managers">list(sectors.manager)</constant>
  <constant name="sensors">list(sectors.sensors)</constant>

  <!-- Track migration effectively means the TM role for a target affects
  multiple agents -->
  <constant name="average_sector_area">forallavg(sectors.actual_area)</constant>
  <constant name="average_sector_path">0.8 * average_sector_area^0.5</constant>
  <constant name="migration_rate">1 / ((2 * average_sector_area)^0.5 / velocity)<
  /constant>
  <constant name="max_agents">5</constant>
  <constant name="num_agents">max(1, min(max_agents, migration_rate * env.time))<
  /constant>
  <constant name="sectors_seen">min(env.num_sectors, (1 / average_sector_path) *
velocity * max(1, uncertainty_radius / average_sector_path) * env.time)</
constant>
  <constant name="percent_sectors_seen">sectors_seen / env.num_sectors</constant>
  <constant name="percent_sectors_seen">1</constant>

  <constant name="uncertainty_radius">5</constant>
  <constant name="influence_radius">uncertainty_radius + 10</constant>
  <constant name="target_area">3.14 * influence_radius^2</constant>

  <constant name="desired_sensors">3</constant>
  <constant name="sensor_density">forallavg(sectors.sensor_density)</constant>
  <constant name="actual_sensors_available">target_area * sensor_density</
constant>

```

```

<constant name="requested_sensors">min(desired_sensors ,
    actual_sensors_available)</constant>

<constant name="directory_delay">forallavg(sm_relations.directory_delay)</
    constant>
<constant name="migration_delay">8000</constant>
<constant name="requested_measurement_rate">env.measurement_rate * 4.0^min(0,
    velocity * 1000 * ( (env.num_sensors / (requested_sensors * org.
    total_targets)) - 1.9)) * (1 - (velocity / average_sector_path) * (
    directory_delay + migration_delay/2))</constant>
<constant name="actual_measurement_rate">0</constant>
<constant name="rms">-1.8 + 1.59e-2 * (actual_measurement_rate + 1.35e-3)^-1</
    constant>

<!-- Distribute communication evenly -->
<modifier name="agent.message_rr" op="+">message_rr / num_agents</modifier>
<modifier name="agent.message_tb" op="+">message_tb / num_agents</modifier>
<modifier name="agent.message_rb" op="+">message_rb / num_agents</modifier>
<modifier name="agent.message_drr" op="+">message_drr / num_agents</modifier>
<modifier name="agent.message_drq" op="+">message_drq / num_agents</modifier>
<modifier name="agent.roles" op="+">1</modifier>

<log name="req_sensors">requested_sensors</log>
<log name="actual_msmnts">actual_measurement_rate</log>
<log name="sectors_seen">sectors_seen</log>
<log name="rms">rms</log>
</node>

<!-- Sensor-Track Manager Relationship -->
<node type="s_tm_relation" name="S-TM">
    <param>organization:org,track_manager:tm,sensor:s</param>

    <!-- Determine measurement rate -->
    <constant name="requested_sensor_rate">tm.requested_sensors / org.total_sensors
        </constant>
    <constant name="requested_measurement_rate">tm.requested_measurement_rate *
        requested_sensor_rate</constant>
    <modifier name="s.requested_measurement_rate" op="+">requested_measurement_rate
        </modifier>

    <!-- RR Messages -->
    <constant name="actual_measurement_rate">requested_measurement_rate * s.
        actual_measurement_ratio</constant>
    <modifier name="tm.actual_measurement_rate" op="+">actual_measurement_rate</
        modifier>
    <modifier name="s.message_rr" op="+">actual_measurement_rate</modifier>

    <!-- RB Messages -->
    <constant name="rebind_rate">requested_sensor_rate * (tm.velocity / (s.radius /
        4))</constant>
    <modifier name="tm.message_rb" op="+">rebind_rate</modifier>

    <log name="req_rate">requested_measurement_rate * 1000</log>
    <log name="ac_rate">actual_measurement_rate * 1000</log>
</node>

<!-- Sector Manager-Track Manager Relationship -->
<node type="sm_tm_relation" name="SM-TM">
    <param>organization:org,track_manager:tm,sector_manager:sm</param>

    <!-- TB messages -->
    <constant name="message_tb">sm.sector.effective_coverage * tm.update_rate</
        constant>
    <modifier name="sm.target_updates" op="+">message_tb</modifier>
    <modifier name="tm.message_tb" op="+">message_tb</modifier>

    <!-- DR Messages -->

```

```

    <constant name="directory_queries">tm.percent_sectors_seen * tm.num_agents * sm
      .sector.num_sensors / org.env.time</constant>
    <modifier name="tm.message_drq" op="+">tm.percent_sectors_seen / org.env.time</
      modifier>
    <modifier name="sm.message_drr" op="+">directory_queries</modifier>

    <!-- Directory Delay -->
    <constant name="directory_delay">3000</constant>
    <constant name="per_message_delay">0</constant>
    <modifier name="directory_delay" op="+">sm.communication_load * 1000 *
      per_message_delay</modifier>
  </node>

  <!-- Base agent type -->
  <node type="agent" abstract="true">
    <param>environment:env</param>
    <is-a>entity</is-a>

    <!-- Some defaults -->
    <constant name="sensors_controlled">0</constant>
    <constant name="roles">0</constant>
    <constraint name="sensors_controlled" op="&lt;=">1</constraint>

    <!-- Communication characteristics -->
    <constant name="communication_messages">communication_load * env.time</constant
      >
    <constant name="communication_profile">[message_rr * env.time, message_tb * env
      .time, message_rb * env.time, message_drr * env.time, message_drq * env.
      time]</constant>

    <log name="profile">communication_profile</log>
    <log name="messages">communication_messages</log>
    <log name="roles">parents.role</log>
  </node>

  <!-- Normal Agent -->
  <node type="normal-agent" name="a">
    <param>environment:env</param>
    <is-a>agent(env)</is-a>

    <constraint name="sensors_controlled" op="==">1</constraint>
    <constraint name="roles" op="&lt;=">6</constraint>
  </node>

  <!-- Robust Agent - example only, not used in experiments -->
  <node type="robust-agent" name="r">
    <param>environment:env</param>
    <is-a>agent(env)</is-a>

    <constraint name="roles" op="&lt;=">0</constraint>
  </node>

  <!-- Base entity type -->
  <node type="entity" abstract="true">
    <constant name="message_rr">0</constant>
    <constant name="message_tb">0</constant>
    <constant name="message_rb">0</constant>
    <constant name="message_drq">0</constant>
    <constant name="message_drr">0</constant>
    <constant name="communication_load">message_rr + message_tb + message_rb +
      message_drr + message_drq</constant>

    <log name="comm">communication_load</log>
  </node>
</organization>

```

## APPENDIX D

### INFORMATION RETRIEVAL ODML MODEL

This appendix contains the complete ODML listing for the information retrieval domain model described in Section 3.1 and inspired by work presented in [214, 215]. The original abstract model is shown graphically above in Figure D.1. The raw textual model given below contains additional structure not shown in that figure; a complete structural graph is shown in Figure D.2. The key differences include separating out the *user* and *other\_mediator* roles, as well as the addition of a *role* node. These changes are for modeling convenience only, and do not affect the discussion given in Section 3.3.

As with other ODML models, this one is structured with the *organization* characteristics defined first, followed by a series of node definitions for the other entities in the organization. The *environment* node specifies a set of scenario constants, and contains the two variables that decide the size of the search and query sets of the mediator. The *user* node is similarly used to store scenario information, in this case to define the rate at which queries will enter the system.

The *mediator* role follows the *user*, and begins by specifying the *agent* it will be bound to, and the number and type of *source* entities that will exist below it. The mediator's *rank* and *query\_probability* are computed next, which determine how likely it is the mediator will be selected to answer a query. From this the *work\_load* can be deduced, which is used to determine the pdf and cdf distributions described in Section 3.3.4. The service and response times are computed last.

The *other\_mediator* node is used to represent the mediators in the system that do not compete with the *mediator*, but are still a distraction because they must be searched during the first part of the query handling phase. They are nearly identical to the normal *mediator*, except that they have no *sources* below them.

The *aggregator* and *database* nodes are similar to mediators, except they do have the ranking and query probability computations. The local *query\_rate* of each is determined from the manager above it. This is used to determine the work load and response times of the entity.

The *agent* and *regular\_agent* nodes contain a small number of default characteristics. The *role*, *manager* and *sources* nodes do as well, although they serve a dual purpose in helping frame the structural decision problems by providing base types that the other entities may inherit.

The code for the model follows below.

```
<?xml version="1.0" encoding="UTF-8"?>
```

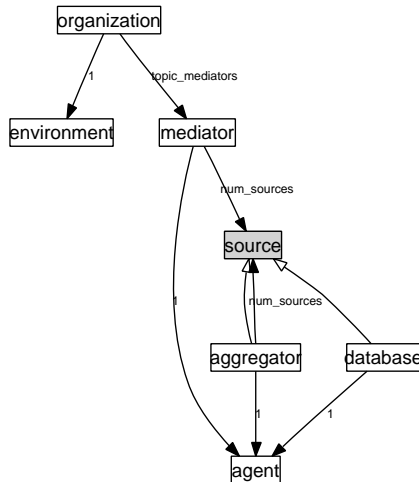


Figure D.1. Graphical view of the ODML information retrieval model.

```

<organization name="content_organization">
  <!-- Create the root level participants -->
  <has-a name="env">environment(this)</has-a>
  <has-a name="users" size="num_users">user(env, this)</has-a>
  <has-a name="mediators" size="num_topic_mediators">mediator(env, this)</has-a>
  <has-a name="other_mediators" size="num_other_mediators">other_mediator(env, this)
  </has-a>

  <!-- Some scenario values -->
  <variable name="num_topic_mediators">1,2,3,4</variable>
  <constant name="num_users">1</constant>
  <constant name="num_other_mediators">3</constant>
  <constant name="total_mediators">num_topic_mediators + num_other_mediators</constant>

  <!-- Gross organizational characteristics -->
  <constant name="response_time">max(mediators.response_time)</constant>
  <constant name="response_recall">forallsum(mediators.recall_portion) / env.topic_size</constant>
  <constraint name="response_recall" op=">=">0.70</constraint>
  <constant name="utility">response_recall * 10 - response_time / 100</constant>

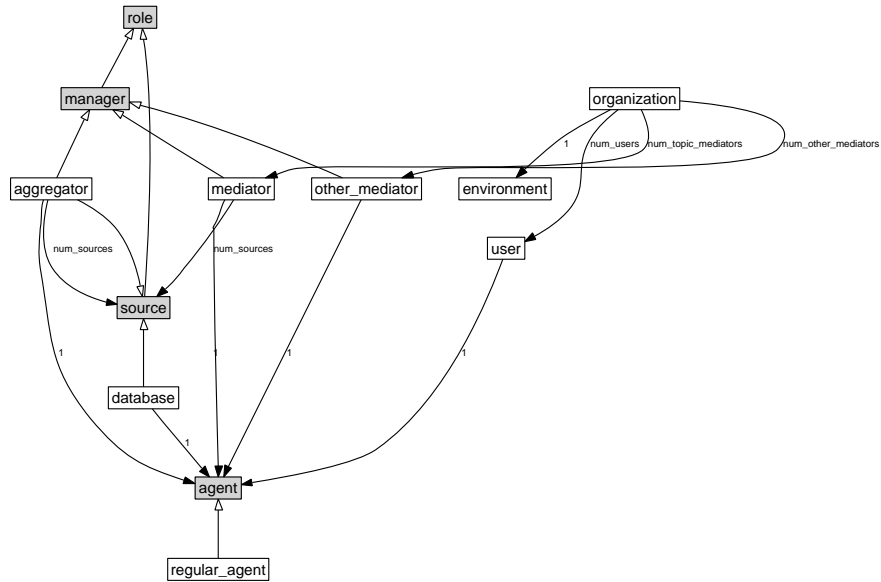
  <!-- Data to log -->
  <log name="organization_dot" file="organization-shape.dot" append="false">todot(this, "true", "false")</log>
  <log name="utility">utility</log>
  <log name="other_mediators">num_other_mediators</log>
  <log name="response_recall">response_recall</log>
  <log name="response_time">response_time</log>

  <!-- Environment -->
  <node type="environment">
    <param>organization:org</param>
    <constant name="topic_size">700</constant> <!-- Total amount of
      topic data -->

    <constant name="topic_query_rate">forallsum(org.users.topic_query_rate)</constant>
    <constant name="nontopic_query_rate">forallsum(org.users.nontopic_query_rate)
    </constant>
  </node>

```





**Figure D.2.** Graphical view of the complete ODML information retrieval model obtained from the source in this appendix.

```

<constant name="message_latency">20</constant>           <!-- Time to send a
    message -->
<constant name="query_service_rate">25/25</constant>     <!-- Service rate to
    interpret query -->
<constant name="process_service_rate">1/200</constant>   <!-- Service rate to
    perform query -->
<constant name="response_service_rate">1/50</constant>  <!-- Service rate to
    interpret response -->

<variable name="search_size">1,2,3,4,5,6</variable>     <!-- How far
    searches propogate -->
<variable name="query_size">1,2,3,4,5,6</variable>      <!-- How far
    queries propogate -->

<constraint name="search_size" op="&lt;=">org.total_mediators</constraint>
<constraint name="query_size" op="&lt;=">org.num_topic_mediators</constraint>

<constant name="search_set_size">search_size</constant>
<constant name="query_set_size">query_size</constant>

<constant name="search_probability">search_set_size / org.total_mediators</
    constant>
<constant name="mediator_query_rate">topic_query_rate * search_probability
    * min(1, query_set_size / org.num_topic_mediators)</constant>

<log name="topic_query_rate">topic_query_rate</log>
<log name="topic_size">topic_size</log>
<log name="query_set_size">query_set_size</log>
<log name="search_set_size">search_set_size</log>
</node>

<!-- Users -->
<node type="user">
    <param>environment:env, organization:org</param>
    <has-a name="agent">agent</has-a>

    <constant name="query_limit">10000</constant>

```

```

    <constant name="topic_query_rate">2/1000</constant>
    <constant name="nontopic_query_rate">0/1000</constant>
</node>

<!-- Top Level Topic Mediators -->
<node type="mediator">
  <param>environment:env, organization:org</param>
  <is-a>manager</is-a>
  <has-a name="agent">agent</has-a>
  <has-a name="sources" size="num_sources">source(this,env)</has-a>
  <variable name="num_sources">1,2,3,4,5,6,7,8</variable>

  <constant name="search_set_size">env.search_set_size</constant>
  <constant name="query_set_size">env.query_set_size</constant>

  <!-- Determine the mediators rank, based on its perceived recall -->
  <constant name="rank">1 + forallsum(forall(s, org.mediators.
    perceived_response_size,
    0^max(perceived_response_size - s, 0) - 0^abs(perceived_response_size - s
    )))</constant>
  <constant name="rank_ties">forallsum(forall(r, org.mediators.rank,
    0^abs(r - rank)))</constant>

  <!-- Determine the probability the mediator will be queried -->
  <constant name="query_probability">(search_set_size / org.total_mediators) *
    (1 / choose(org.total_mediators - 1, search_set_size - 1)) *
    forallsum(forrange(r, 0, query_set_size,
      forallsum(forrange(g, 0, min(search_set_size, rank_ties),
        choose(org.total_mediators - rank - rank_ties + 1,
          search_set_size - r - g - 1)
        * choose(rank - 1, r)
        * choose(rank_ties - 1, g)
        * min(1, (query_set_size - r) / (g + 1))
      )))
    )</constant>

  <constant name="data_size">forallsum(sources.data_size)</constant>
  <constant name="topic_size">forallsum(sources.topic_size)</constant>
  <constant name="topic_percentage">topic_size / data_size</constant>

  <constant name="actual_response_size">topic_size</constant>
  <constant name="perceived_response_size">topic_size</constant>
  <constant name="recall_portion">query_probability * actual_response_size</
    constant>

  <!-- Determine the work load the mediator will see -->
  <constant name="query_rate">query_probability * env.topic_query_rate
    + (1 - topic_percentage) * env.nontopic_query_rate</constant>
  <constant name="response_rate">0</constant>

  <constant name="arrival_rate">query_rate</constant>
  <constant name="service_rate">env.response_service_rate / num_sources</
    constant>
  <constant name="effective_service_rate">arrival_rate / agent.work_load</
    constant>
  <constant name="poisson_rate">effective_service_rate - arrival_rate</constant
  >
  <constraint name="arrival_rate" op="<=">effective_service_rate</constraint
  >
  <modifier name="agent.work_load" op="+>arrival_rate / service_rate</modifier
  >

  <constant name="local_pdf_list">forrange(x, 0, (dist_range / dist_step),
    poisson_rate * e^(- poisson_rate * (x) * dist_step) /* Exp pdf f(x)
    */
  )</constant>
  <constant name="local_cdf_list">forrange(x, 0, (dist_range / dist_step),

```

```

    1 - e^(- poisson_rate * (x+1) * dist_step)           /* Exp cdf F(x)
    */
  )</constant>

<constant name="source_pdf_list">forrange(x, 0, (dist_range / dist_step),
  forallprod(forall(s, sources, listitem(s.cdf_list, x)))
  * forallsum(forall(s, sources, listitem(s.pdf_list, x) / listitem(s.
  cdf_list, x)))
)</constant>
<constant name="source_cdf_list">forrange(x, 0, (dist_range / dist_step),
  forallprod(forall(s, sources, listitem(s.cdf_list, x)))
)</constant>

<constant name="pdf_list">forrange(x, 0, (dist_range / dist_step),
  forallsum(forrange(i, 0, x+1,
  listitem(source_pdf_list, i) * listitem(local_pdf_list, x - i) *
  dist_step
  ))
)</constant>
<constant name="cdf_list">forrange(x, 0, (dist_range / dist_step),
  forallsum(forrange(i, 0, x+1,
  listitem(source_pdf_list, i) * listitem(local_cdf_list, x - i) *
  dist_step
  ))
)</constant>
<log name="pdf" file="pdf.dat">forrange(x, 0, (dist_range / dist_step),
  [(x * dist_step) + overhead_time, listitem(pdf_list, x)]
)</log>

<!-- Determine service and response times -->
<constant name="service_time">forallsum(forrange(x, 1, (dist_range /
  dist_step),
  (x * dist_step) * (listitem(pdf_list, x) * dist_step)
  ))</constant>

<constant name="overhead_time">
  env.message_latency           /* Query down from user */
+ env.message_latency          /* Search to mediators */
+ env.message_latency          /* Search reply from mediators */
+ env.message_latency          /* Query to mediators */
+ env.message_latency          /* Query down to sources */
+ max(sources.overhead_time)   /* Subordinate overhead */
+ env.message_latency          /* Response from mediators */
+ env.message_latency          /* Response up to user */
</constant>
<constant name="response_time">overhead_time + service_time</constant>

<log name="topic_size">topic_size</log>
<log name="rank">rank</log>
<log name="query_probability">query_probability</log>
<log name="response_time">response_time</log>
<log name="poisson_rate">poisson_rate</log>
<log name="query_rate">query_rate</log>
<log name="service_rate">service_rate</log>
<log name="response_rate">response_rate</log>
<log name="service_time">service_time</log>
</node>

<!-- Non-Topic Mediators -->
<node type="other_mediator">
  <param>environment:env, organization:org</param>
  <is-a>manager</is-a>
  <has-a name="agent" discriminator="work_load">agent</has-a>
  <constant name="sources">list()</constant>
  <constant name="num_sources">0</constant>

  <constant name="search_set_size">env.search_set_size</constant>
  <constant name="query_set_size">env.query_set_size</constant>

```

```

<constant name="query_rate">query_probability * env.topic_query_rate
  + (1 - topic_percentage) * env.nontopic_query_rate</constant>
<constant name="response_rate">0</constant>
<constant name="overhead_time">
  env.message_latency +                               /* Query down */
  env.message_latency                                 /* Response up */
</constant>
<constant name="response_time">overhead_time</constant>
<constant name="service_rate">1</constant>
<constraint name="query_rate" op="&lt;=">service_rate</constraint>

<constant name="rank">1 + forallsum(forall(s, org.mediators.
  perceived_response_size,
  0^max(perceived_response_size - s, 0) - 0^abs(perceived_response_size - s
  )))</constant>
<constant name="rank_ties">forallsum(forall(r, org.mediators.rank,
  0^abs(r - rank)))</constant>
<constant name="query_probability">(search_set_size / org.total_mediators) *
  (1 / choose(org.total_mediators - 1, search_set_size - 1)) *
  forallsum(forrage(r, 0, query_set_size,
    forallsum(forrage(g, 0, min(search_set_size, rank_ties),
      choose(org.total_mediators - rank - rank_ties + 1,
        search_set_size - r - g - 1)
      * choose(rank - 1, r)
      * choose(rank_ties - 1, g)
      * min(1, (query_set_size - r) / (g + 1))
    ))
  ))</constant>

<constant name="data_size">forallsum(sources.data_size)</constant>
<constant name="topic_size">forallsum(sources.topic_size)</constant>
<constant name="topic_percentage">topic_size / data_size</constant>

<constant name="actual_response_size">topic_size</constant>
<constant name="perceived_response_size">topic_size</constant>
<constant name="recall_portion">query_probability * actual_response_size</
  constant>

<log name="topic_size">topic_size</log>
<log name="rank">rank</log>
<log name="query_probability">query_probability</log>
</node>

<!-- Mid-Level Aggregation Nodes -->
<node type="aggregator" recurse="5">
  <param>manager:manager,environment:env</param>
  <is-a>source(manager,env)</is-a>
  <is-a>manager</is-a>
  <has-a name="agent" discriminator="work_load">agent</has-a>
  <has-a name="sources" size="num_sources">source(this,env)</has-a>
  <variable name="num_sources">2,3,4</variable>

  <constant name="query_rate">manager.query_rate</constant>
  <constant name="response_rate">0</constant>

  <!-- Determine the work load the aggregator will see -->
  <constant name="arrival_rate">query_rate</constant>
  <constant name="service_rate">env.response_service_rate / num_sources</
    constant>
  <constant name="effective_service_rate">arrival_rate / agent.work_load</
    constant>
  <constant name="poisson_rate">effective_service_rate - arrival_rate</constant
  >
  <constraint name="arrival_rate" op="&lt;=">effective_service_rate</constraint
  >
  <modifier name="agent.work_load" op="+">arrival_rate / service_rate</modifier
  >

```

```

<constant name="local_pdf_list">forrange(x, 0, (dist_range / dist_step),
    poisson_rate * e^(- poisson_rate * (x) * dist_step)    /* Exp pdf f(x)
    */
)</constant>
<constant name="local_cdf_list">forrange(x, 0, (dist_range / dist_step),
    1 - e^(- poisson_rate * (x+1) * dist_step)            /* Exp cdf F(x)
    */
)</constant>

<constant name="source_pdf_list">forrange(x, 0, (dist_range / dist_step),
    forallprod(forall(s, sources, listitem(s.cdf_list, x)))
    * forallsum(forall(s, sources, listitem(s.pdf_list, x) / listitem(s.
    cdf_list, x))))
)</constant>
<constant name="source_cdf_list">forrange(x, 0, (dist_range / dist_step),
    forallprod(forall(s, sources, listitem(s.cdf_list, x)))
)</constant>

<constant name="pdf_list">forrange(x, 0, (dist_range / dist_step),
    forallsum(forrange(i, 0, x+1,
        listitem(source_pdf_list, i) * listitem(local_pdf_list, x - i) *
        dist_step
    )))
)</constant>
<constant name="cdf_list">forrange(x, 0, (dist_range / dist_step),
    forallsum(forrange(i, 0, x+1,
        listitem(source_pdf_list, i) * listitem(local_cdf_list, x - i) *
        dist_step
    )))
)</constant>

<!-- Determine the service and response times -->
<constant name="service_time">forallsum(forrange(x, 1, (dist_range /
    dist_step),
    (x * dist_step) * (listitem(pdf_list, x) * dist_step)
))</constant>

<constant name="overhead_time">
    env.message_latency                                /* Query down to sources
    */
    + max(sources.overhead_time)                       /* Subordinate overhead
    */
    + env.message_latency                               /* Response up to manager
    */
</constant>
<constant name="response_time">overhead_time + service_time</constant>

<constant name="data_size">forallsum(sources.data_size)</constant>
<constant name="topic_size">forallsum(sources.topic_size)</constant>

<modifier name="manager.response_rate" op="+">response_rate / num_sources</
    modifier>

<log name="data_size">data_size</log>
<log name="topic_size">topic_size</log>
<log name="response_time">response_time</log>
<log name="poisson_rate">poisson_rate</log>
<log name="service_time">service_time</log>
</node>

<!-- Leaf Source Nodes -->
<node type="database">
    <param>manager:manager, environment:env</param>
    <is-a>source(manager, env)</is-a>
    <has-a name="agent" discriminator="work_load">agent</has-a>

    <!-- Determine the work load the aggregator will see -->

```

```

<constant name="query_rate">manager.query_rate</constant>

<constant name="arrival_rate">query_rate</constant>
<constant name="service_rate">env.process_service_rate</constant>
<constant name="effective_service_rate">arrival_rate / agent.work_load</
constant>
<constant name="poisson_rate">effective_service_rate - arrival_rate</constant
>
<constraint name="arrival_rate" op="<=">effective_service_rate</constraint
>
<modifier name="agent.work_load" op="+">arrival_rate / service_rate</modifier
>

<constant name="local_pdf_list">forrange(x, 0, (dist_range / dist_step),
    poisson_rate * e^(- poisson_rate * (x) * dist_step)    /* Exp pdf f(x)
    */
)</constant>
<constant name="local_cdf_list">forrange(x, 0, (dist_range / dist_step),
    1 - e^(- poisson_rate * (x+1) * dist_step)            /* Exp cdf F(x)
    */
)</constant>
<constant name="pdf_list">local_pdf_list</constant>
<constant name="cdf_list">local_cdf_list</constant>

<constant name="service_time">forallsum(forrange(x, 1, (dist_range /
    dist_step),
    (x * dist_step) * listitem(pdf_list, x) * dist_step
))</constant>

<constant name="overhead_time">
    env.message_latency                                     /* Response up to
    manager */
</constant>
<constant name="response_time">overhead_time + service_time</constant>

<constant name="data_size">100</constant>
<constant name="topic_percentage">0.8</constant>
<constant name="topic_size">data_size * topic_percentage</constant>

<modifier name="manager.response_rate" op="+">query_rate</modifier>

<log name="data_size">data_size</log>
<log name="topic_size">topic_size</log>
<log name="response_time">response_time</log>
<log name="poisson_rate">poisson_rate</log>
</node>

<!-- Agents -->
<node type="agent" abstract="true">
    <constant name="work_load">0</constant>
</node>
<node type="regular_agent" name="agent" size="50">
    <is-a>agent</is-a>
</node>

<!-- Types -->
<node type="manager" abstract="true">
    <is-a>role</is-a>
    <constant name="response_time">0</constant>
    <constant name="query_rate">0</constant>
</node>

<node type="source" abstract="true">
    <is-a>role</is-a>
    <param>manager:manager , environment:env</param>
    <constant name="response_time">0</constant>
    <constant name="topic_size">0</constant>
    <constant name="data_size">0</constant>

```

```
</node>

<node type="role" abstract="true">
  <constant name="e">2.71828183</constant>
  <constant name="dist_step">10</constant>
  <constant name="dist_range">4000</constant>
</node>

</organization>
```

# APPENDIX E

## TILING REDUCTION MODEL

This appendix contains the complete ODML listing for the TILING reduction model described in Section 4.1.1.

```
<?xml version="1.0" encoding="UTF-8"?>
<organization name="tiling">
  <has-a name="rows" size="N" ordered="true">row(N)</has-a>
  <has-a name="columns" ordered="true">forall(i, N_range):column(N, i, rows)</has-a>
  >

  <!-- Problem Grid Size -->
  <constant name="N">3</constant>
  <constant name="N_range">forrange(r, 0, N, r)</constant>

  <!-- Cell (0,0) Constraint -->
  <constant name="toprow">listitem(rows, 0)</constant>
  <constant name="origin">listitem(toprow.tiles, 0)</constant>
  <constant name="origintype">origin.type</constant>
  <constraint name="origintype" op="==">0</constraint>

  <log name="insdot" file="results/organization.dot">todot(this)</log>
  <log name="tmpdot" file="results/organization-template.dot">todot(this.template)<
  /log>

  <node type="row">
    <param>float:N</param>
    <has-a name="tiles" size="N" ordered="true">tile</has-a>
    <has-a name="relations">forall(i, N-1):horizontal_relation(tiles, i)</has-a>
    <constant name="N-1">forrange(r, 0, N-1, r)</constant>
  </node>

  <node type="column">
    <param>float:N,float:c,[row]:rows</param>
    <has-a name="relations">forall(i, N-1):vertical_relation(tiles, i)</has-a>
    <constant name="tiles">forall(r, rows, listitem(r.tiles, c))</constant>
    <constant name="N-1">forrange(r, 0, N-1, r)</constant>
  </node>

  <!-- Abstract Structures -->

  <node type="tile" abstract="true">
    <param>float:type</param>
  </node>

  <node type="relation" abstract="true">
    <param>float:t1t,float:t2t,[tile]:tiles,float:i</param>
    <constant name="t1">listitem(tiles, i)</constant>
    <constant name="t2">listitem(tiles, i+1)</constant>
    <constraint name="t1t" op="==">t1.type</constraint>
    <constraint name="t2t" op="==">t2.type</constraint>
  </node>

  <node type="horizontal_relation" abstract="true">
```



```

        <param>float:t1t,float:t2t,[tile]:tiles,float:i</param>
        <is-a>relation(t1t,t2t,tiles,i)</is-a>
    </node>
    <node type="vertical_relation" abstract="true">
        <param>float:t1t,float:t2t,[tile]:tiles,float:i</param>
        <is-a>relation(t1t,t2t,tiles,i)</is-a>
    </node>

    <!-- Tiles -->

    <node type="t_0" name="0">
        <is-a>tile(0)</is-a>
    </node>
    <node type="t_1" name="1">
        <is-a>tile(1)</is-a>
    </node>

    <!-- Compatibility Relations -->

    <node type="h_01" name="h_01">
        <param>[tile]:tiles,float:i</param>
        <is-a>horizontal_relation(0,1,tiles,i)</is-a>
    </node>
    <node type="h_10" name="h_10">
        <param>[tile]:tiles,float:i</param>
        <is-a>horizontal_relation(1,0,tiles,i)</is-a>
    </node>

    <node type="v_01" name="v_01">
        <param>[tile]:tiles,float:i</param>
        <is-a>vertical_relation(0,1,tiles,i)</is-a>
    </node>
    <node type="v_10" name="v_10">
        <param>[tile]:tiles,float:i</param>
        <is-a>vertical_relation(1,0,tiles,i)</is-a>
    </node>

</organization>

```

## APPENDIX F

### SUBSET-SUM REDUCTION MODEL

This appendix contains the complete ODMML listing for the SUBSET-SUM reduction model described in Section 4.2.1.

```
<?xml version="1.0" encoding="UTF-8"?>
<organization name="subset_sum">
  <has-a name="N" size="5">number(this)</has-a>
  <constant name="sum">0</constant>
  <constant name="t">0</constant>
  <constraint name="sum" op="==">t</constraint>

  <node type="number" abstract="true">
</node>

  <node type="number_empty" size="4">
    <param>organization:sum</param>
    <is-a>number()</is-a>
  </node>

  <node type="number-1" size="2">
    <param>organization:sum</param>
    <is-a>number()</is-a>
    <modifier name="sum.sum" op="+">-1</modifier>
  </node>

  <node type="number1" size="3">
    <param>organization:sum</param>
    <is-a>number()</is-a>
    <modifier name="sum.sum" op="+">1</modifier>
  </node>
</organization>
```

## BIBLIOGRAPHY

- [1] Sherief Abdallah, Nevin Darwish, and Osman Hegazy. Monitoring and synchronization for teamwork in gpgp. In *SAC '02: Proceedings of the 2002 ACM symposium on Applied computing*, pages 288–293. ACM Press, 2002.
- [2] Sherief Abdallah and Victor Lesser. Organization-Based Cooperative Coalition Formation. *Proceedings of the IEEE/WIC/ACM International Conference on Intelligent Agent Technology, IAT*, pages 162–168, September 2004.
- [3] Manju Ahuja and Kathleen M. Carley. Network structure in virtual organizations. *Organization Science*, 10(6):741–757, 1999.
- [4] Alexander Artikis. *Executable Specification of Open Norm-Governed Computational Systems*. PhD thesis, Department of Electrical & Electronic Engineering, Imperial College London, November 2003.
- [5] Robert Axelrod. An evolutionary approach to norms. *The American Political Science Review*, 80(4):1095–1111, 1986.
- [6] K. Suzanne Barber and Cheryl E. Martin. Dynamic adaptive autonomy in multiagent systems: Representation and justification. *International Journal of Pattern Recognition and Artificial Intelligence*, 15(3):405–433, 2001.
- [7] J. A. C. Baum. Organizational ecology. In S. R. Clegg, C. Hardy, and W. R. Nord, editors, *Handbook of Organization Studies*, pages 77–114. Sage Publications, London, U.K., 1996.
- [8] G. Beavers and H. Hexmoor. Teams of agents. In *Proceedings of the IEEE Systems, Man, and Cybernetics Conference*, pages 574–582, 2001.
- [9] Raphen Becker, Shlomo Zilberstein, Victor Lesser, and Claudia V. Goldman. Solving Transition Independent Decentralized Markov Decision Processes. *Journal of Artificial Intelligence Research*, 22:423–455, December 2004.
- [10] Daniel S. Bernstein, Robert Givan, Neil Immerman, and Shlomo Zilberstein. The complexity of decentralized control of markov decision processes. *Mathematics of Operations Research*, 27(4):819–840, 2002.
- [11] Alan H. Bond and Les Gasser. An analysis of problems and research in DAI. In Alan H. Bond and Les Gasser, editors, *Readings in Distributed Artificial Intelligence*, pages 3–35. Morgan Kaufmann, 1988.

- [12] L. Bongaerts. *Integration of Scheduling and Control in Holonic Manufacturing Systems*. PhD thesis, Katholieke Universiteit Leuven, Belgium, 1998.
- [13] Stephen P. Borgatti and Pacey C. Foster. The network paradigm in organizational research: A review and typology. *Journal of Management*, 29(6):991–1013, 2003.
- [14] Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, and Eve Maler. *Extensible Markup Language (XML) 1.0 (Second Edition)*. W3C, 1.1 edition, October 2000. URL: <http://www.w3c.org/TR/REC-xml>.
- [15] S. Breban and J. Vassileva. Long-term coalitions for the electronic marketplace. In *Proceedings of Canadian AI Workshop on Novel E-Commerce Applications of Agents*, pages 6–12, 2001.
- [16] C. Brooks and E. Durfee. Congregating and market formation. In *Proceedings of the first international joint conference on Autonomous agents and multiagent systems*, pages 96–103. ACM Press, 2002.
- [17] C. Brooks and E. Durfee. Congregation formation in multiagent systems. *Journal of Autonomous Agents and Multiagent Systems*, 7(1-2):145–170, 2003.
- [18] C. Brooks, E. Durfee, and A. Armstrong. An introduction to congregating in multiagent systems. In *Proceedings of the Fourth International Conference on Multiagent Systems*, pages 79–86, 2000.
- [19] Stefan Bussmann and Klaus Schild. Self-organizing manufacturing control: An industrial application of agent technology. In *Proceedings of the 4th International Conference on Multi-Agent Systems (ICMAS 2000)*, pages 87–94. IEEE Computer Society, July 2000.
- [20] T. Camp, J. Boleng, and V. Davies. A survey of mobility models for ad hoc network research. *Wireless Communications & Mobile Computing (WCMC): Special issue on Mobile Ad Hoc Networking: Research, Trends and Applications*, 2(5):483–502, 2002.
- [21] Henrique Lopes Cardoso and Eugénio Oliveira. Virtual enterprise normative framework within electronic institutions. In *Proceedings of the 5th Int. Workshop on Engineering Societies in the Agents World (ESAW 04)*, October 2004.
- [22] Kathleen Carley. Organizational adaptation. *Annals of Operations Research*, 75:25–47, 1998.
- [23] Kathleen M. Carley and Les Gasser. Computational organization theory. In Gerhard Weiss, editor, *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*, pages 299–330. MIT Press, 1999.

- [24] Kathleen M. Carley and Les Gasser. Computational organization theory. In Gerhard Weiss, editor, *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*, pages 299–330. The MIT Press, 1999.
- [25] B. Chandrasekaran. Natural and social system metaphors for distributed problem solving: Introduction to the issue. *IEEE Transactions on Systems, Man, and Cybernetics*, 11(1):1–5, January 1981.
- [26] A. Chavez and P. Maes. Kasbah: An agent marketplace for buying and selling goods. In *First International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology (PAAM'96)*, pages 75–90, London, UK, 1996. Practical Application Company.
- [27] V. Chvatal. A greedy heuristic for the setcovering problem. *Mathematics of Operations Research*, 4(3), August 1979.
- [28] J. Collins, M. Tsvetovat, B. Mobasher, and M. Gini. MAGNET: A multi-agent contracting system for plan execution. In *Proceedings of Workshop on Artificial Intelligence and Manufacturing: State of the Art and State of Practice*, pages 63–68. AAAI Press, August 1998.
- [29] J. C. Collins and L. C. Lee. Building electronic marketplaces with the ZEUS agent tool-kit. *Agent Mediated Electronic Commerce*, 1571:1–24, 1998.
- [30] Marco Colombetti, Nicoletta Fornara, and Mario Verdicchio. A social approach to communication in multiagent systems. In João Alexandre Leite, Andrea Omicini, Leon Sterling, and Paolo Torroni, editors, *Declarative Agent Languages and Technologies*, volume 2990 of *Lecture Notes in Artificial Intelligence*, pages 191–220. Springer-Verlag, May 2004.
- [31] Daniel Corkill and Victor Lesser. The use of meta-level control for coordination in a distributed problem solving network. *Proceedings of the Eighth International Joint Conference on Artificial Intelligence*, pages 748–756, August 1983.
- [32] Daniel D. Corkill and Susan E. Lander. Diversity in Agent Organizations. *Object Magazine*, 8(4):41–47, May 1998.
- [33] Guifre Cuni, Marc Esteva, Pere Garcia, Eloi Puertas, Carles Sierra, and Teresa Solchaga. MASFIT: Multi-Agent System for FIsh Trading. In *Proceedings of the 16th European Conference on Artificial Intelligence, (ECAI'2004)*, pages 710–714. IOS Press, August 2004.
- [34] Mark R. Cutkosky, Robert S. Englemore, Richard E. Fikes, Michael R. Genesereth, Thomas R. Gruber, William S. Mark, Jay M. Tenenbaum, and Jay C. Weber. PACT: An experiment in integrating concurrent engineering systems. In Michael N. Huhns and Munindar P. Singh, editors, *Readings in Agents*, pages 46–55. Morgan Kaufmann, San Francisco, CA, USA, 1997.

- [35] Mehdi Dastani, Virginia Dignum, and Frank Dignum. Role-assignment in open agent societies. In *Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 489–496. ACM Press, 2003.
- [36] Herbert Aron David. *Order Statistics, 2nd Ed.* Wiley, 1981.
- [37] R. Davis and R. G. Smith. Negotiation as a metaphor for distributed problem solving. *Artificial Intelligence*, 20(1):63–109, January 1983.
- [38] K. Decker and V. Lesser. Generalizing the Partial Global Planning Algorithm. *International Journal on Intelligent Cooperative Information Systems*, 1(2):319–346, June 1992.
- [39] K. Decker and V. Lesser. An Approach to Analyzing the Need for Meta-Level Communication. *International Joint Conference on Artificial Intelligence*, 1, January 1993.
- [40] K. Decker and V. R. Lesser. Quantitative Modeling of Complex Environments. *International Journal of Intelligent Systems in Accounting, Finance and Management. Special Issue on Mathematical and Computational Models and Characteristics of Agent Behaviour.*, 2:215–234, January 1993.
- [41] K. Decker, Katia Sycara, and M. Williamson. Cloning for intelligent adaptive information agents. In C. Zhang and Lukose D, editors, *Multi-Agent Systems*, pages 63–75. Springer Verlag, 1997.
- [42] Keith Decker. TAEMS: A Framework for Environment Centered Analysis & Design of Coordination Mechanisms. In *Foundations of Distributed Artificial Intelligence, Chapter 16*, pages 429–448. G. O’Hare and N. Jennings (eds.), Wiley Inter-Science, January 1996.
- [43] Keith Decker, Victor Lesser, Nagendra Prasad, and Thomas Wagner. MACRON: An Architecture for Multi-Agent Cooperative Information Gathering. *Proceedings of the CIKM Workshop on Intelligent Information Agents*, December 1995.
- [44] Chrysanthos Dellarocas and Mark Klein. Civil agent societies: Tools for inventing open agent-mediated electronic marketplaces. In *Agent Mediated Electronic Commerce (IJCAI Workshop)*, pages 24–39, 1999.
- [45] Scott DeLoach. Modeling organizational rules in the multi-agent systems engineering methodology. In *Proceedings of the 15th Conference of the Canadian Society for Computational Studies of Intelligence on Advances in Artificial Intelligence*, pages 1–15. Springer-Verlag, 2002.
- [46] Scott DeLoach and Eric Matson. An Organizational Model for Designing Adaptive Multiagent Systems. In *Proceedings of the AAAI-04 Workshop on Agent Organizations: Theory and Practice*, pages 66–73, San Jose, California, July 2004. AAAI Press, California.

- [47] Jay L. Devore. *Probability and Statistics for Engineering and the Sciences*. Wadsworth, Inc., Belmont, CA, 1995.
- [48] V. Dignum, J-J. Meyer, H. Weigand, and F. Dignum. An organizational-oriented model for agent societies. *Proceedings of International Workshop on Regulated Agent-Based Social Systems: Theories and Applications (RASTA'02)*, July 2002.
- [49] Virginia Dignum. *A model for organizational interaction: based on agents, founded in logic*. PhD thesis, University of Utrecht, Utrecht, The Netherlands, 2003.
- [50] Virginia Dignum, Javier Vazquez-Salceda, and Frank Dignum. Omni: Introducing social structure, norms and ontologies into agent organizations. In *Second International Workshop on Programming Multi-Agent Systems at the Third International Joint Conference on Autonomous Agents and Multi-Agent Systems*, pages 91–102, New York, NY, July 20 2004.
- [51] Edmund Durfee, Victor Lesser, and Daniel Corkill. Coherent Cooperation Among Communicating Problem Solvers. *IEEE Transactions on Computers*, C36(11):1275–1291, November 1987.
- [52] Edmund H. Durfee and Thomas A. Montgomery. Coordination as distributed search in a hierarchical behavior space. *IEEE Transactions on Systems, Man, and Cybernetics*, 21(6):1363–1378, 1991.
- [53] E.H. Durfee and V.R. Lesser. Partial Global Planning: A Coordination Framework for Distributed Hypothesis Formation. *IEEE Transactions on Systems, Man, and Cybernetics*, 21(5):1167–1183, September 1991.
- [54] Cora Beatriz Excelente-Toledo and Nicholas R. Jennings. The dynamic selection of coordination mechanisms. *Autonomous Agents and Multi-Agent Systems*, 9(1-2):55–85, July - September 2004.
- [55] S. Shaheen Fatima and Michael Wooldridge. Adaptive task and resource allocation in multi-agent systems. In Jörg P. Müller, Elisabeth Andre, Sandip Sen, and Claude Frasson, editors, *Proceedings of the Fifth International Conference on Autonomous Agents*, pages 537–544, Montreal, Canada, 2001. ACM Press.
- [56] Klaus Fischer. Agent-based design of holonic manufacturing systems. *Journal of Robotics and Autonomous Systems*, 27(1-2):3–13, 1999.
- [57] David Fitoussi and Moshe Tennenholtz. Choosing social laws for multi-agent systems: minimality and simplicity. *Artificial Intelligence*, 119(1-2):61–101, 2000.
- [58] Ian T. Foster, Nicholas R. Jennings, and Carl Kesselman. Brain meets brawn: Why grid and agents need each other. In *Proceedings of the 3rd International*

*Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2004)*, pages 8–15. IEEE Computer Society, August 2004.

- [59] Mark Fox, Mihai Barbuceanu, Michael Gruninger, and Jinxin Lin. An Organizational Ontology for Enterprise Modeling. In Michael J. Prietula, Kathleen M. Carley, and Les Gasser, editors, *Simulating Organizations: Computational Models of Institutions and Groups*, pages 131–152. AAAI Press / MIT Press, 1998.
- [60] Mark S. Fox. Organization structuring: Designing large complex software. Computer Science Technical Report CMU-CS-79-155, Carnegie-Mellon University, December 1979.
- [61] Mark S. Fox. An organizational view of distributed systems. *IEEE Transactions on Systems, Man, and Cybernetics*, 11(1):70–80, January 1981.
- [62] J. Galbraith. *Organization Design*. Addison-Wesley, Reading, MA, 1977.
- [63] Les Gasser. Social conceptions of knowledge and action: DAI foundations and open systems semantics. *Artificial Intelligence*, 47(1-3):107–138, 1991.
- [64] Les Gasser. Perspectives on organizations in multi-agent systems. In *Multi-agents systems and applications*, pages 1–16. Springer-Verlag New York, Inc., 2001.
- [65] Michael R. Genesereth. An agent-based framework for interoperability. In Jeffrey Bradshaw, editor, *Software agents*, pages 317–345. MIT Press, 1997.
- [66] N. Glaser and P. Morignot. Societies of autonomous agents and their reorganisation. In W. Tschacher and J.P. Dauwalder, editors, *Dynamics, Synergetics, Autonomous Agents — Nonlinear Systems Approaches to Cognitive Psychology and Cognitive Science*. World Scientific, 1998.
- [67] Alyssa Glass and Barbara Grosz. Socially conscious decision-making. In Carles Sierra, Maria Gini, and Jeffrey S. Rosenschein, editors, *Proceedings of the Fourth International Conference on Autonomous Agents*, pages 217–224, Barcelona, Catalonia, Spain, 2000. ACM Press.
- [68] N. Gnanasambandam, S. Lee, N. Gautam, S. R. T. Kumara, W. Peng, V. Manikonda, M. Brinn, and M. Greaves. Reliable MAS performance prediction using queueing models. In *Proceedings of the IEEE Multi-agent Security and Survivability Symposium (MASS)*, 2004.
- [69] Nathan Griffiths. Supporting cooperation through clans. In *Cybernetic Intelligence, Challenges and Advances – Proceedings of the 2nd IEEE Systems, Man and Cybernetics, UK & RI Chapter Conference*, 2003.
- [70] Barbara Grosz and Sarit Kraus. Collaborative plans for complex group action. *Artificial Intelligence*, 86(2):269–357, 1996.



- [71] Barbara Grosz and Sarit Kraus. The Evolution of SharedPlans. In *Foundations and Theories of Rational Agencies*, pages 227–262. Kluwer Academic Publishers, 1999.
- [72] B.J. Grosz and C.L. Sidner. Plans for discourse. In P.R. Cohen, J. Morgan, and M.E. Pollack, editors, *Intentions in Communication*, pages 417–444. MIT Press, 1990.
- [73] Robert H. Guttman, Alexandros G. Moukas, and Pattie Maes. Agent-mediated electronic commerce: a survey. *Knowl. Eng. Rev.*, 13(2):147–159, 1998.
- [74] S. Hayden, C. Carrick, and Q. Yang. Architectural design patterns for multi-agent coordination. In *Proceedings of the International Conference on Agent Systems '99*, May 1999.
- [75] Sandra C. Hayden, Christian Carrick, and Qiang Yang. A catalog of agent coordination patterns. In *AGENTS '99: Proceedings of the third annual conference on Autonomous Agents*, pages 412–413, New York, NY, USA, 1999. ACM Press.
- [76] Carl Hewitt. Offices are open systems. *ACM Transactions on Office Information Systems*, 4(3):271–287, July 1986.
- [77] H. Hexmoor and G. Beavers. Towards teams of agents. In *Proceedings of the International Conference in Artificial Intelligence (IC-AI'2001)*. CSREA Press, 2001.
- [78] Bryan Horling, Brett Benyo, and Victor Lesser. Using Self-Diagnosis to Adapt Organizational Structures. *Proceedings of the 5th International Conference on Autonomous Agents*, pages 529–536, June 2001.
- [79] Bryan Horling and Victor Lesser. Data Dissemination Techniques For Distributed Simulation Environments. In R. G. Ingalls, M. D. Rossetti, J. S. Smith, and B. A. Peters, editors, *Proceedings of the 2004 Winter Simulation Conference*, pages 792–800, Washington, D.C., December 2004. Omnipress.
- [80] Bryan Horling and Victor Lesser. A Survey of Multi-Agent Organizational Paradigms. *Knowledge Engineering Review*, 2005.
- [81] Bryan Horling and Victor Lesser. Analyzing, Modeling and Predicting Organizational Effects in a Distributed Sensor Network. *Journal of the Brazilian Computer Society, Special Issue on Agents Organizations*, pages 9–30, July 2005.
- [82] Bryan Horling and Victor Lesser. Quantitative Organizational Models for Large-Scale Agent Systems. In Toru Ishida, Les Gasser, and Hideyuki Nakashima, editors, *Massively Multi-Agent Systems I*, volume LNAI 3446, pages 121–135. Springer-Verlag, Berlin, Kyoto, Japan, 2005.

- [83] Bryan Horling, Victor Lesser, Regis Vincent, Tom Wagner, Anita Raja, Shelley Zhang, Keith Decker, and Alan Garvey. The TAEMS White Paper, January 1999.
- [84] Bryan Horling, Roger Mailler, and Victor Lesser. A Case Study of Organizational Effects in a Distributed Sensor Network. In *Proceedings of the International Conference on Intelligent Agent Technology (IAT 2004)*, pages 51–57, Beijing, China, September 2004.
- [85] Bryan Horling, Roger Mailler, and Victor Lesser. Farm: A Scalable Environment for Multi-Agent Development and Evaluation. In Alessandro Garcia Carlos Lucena, Jaelson Castro Alexander Romanovsky, and Paulo Alencar, editors, *Advances in Software Engineering for Multi-Agent Systems*, pages 220–237. Springer-Verlag, Berlin, February 2004.
- [86] Bryan Horling, Roger Mailler, Jiaying Shen, Regis Vincent, and Victor Lesser. Using Autonomy, Organizational Design and Negotiation in a Distributed Sensor Network. In Victor Lesser, Charles Ortiz, and Milind Tambe, editors, *Distributed Sensor Networks: A multiagent perspective*, pages 139–183. Kluwer Academic Publishers, 2003.
- [87] Gábor Horváth. Approximate waiting time analysis of priority queues. In *Proceedings of the Fifth International Workshop on Performability Modeling of Computer and Communication Systems (PMCCS5)*, 2001.
- [88] Jomi Fred Hübner, Jaime Simão Sichman, and Olivier Boissier. A model for the structural, functional, and deontic specification of organizations in multiagent systems. In *Proceedings of the Brazilian Symposium on Artificial Intelligence (SBIA '02)*, pages 118–128, 2002.
- [89] Jomi Fred Hübner, Jaime Simão Sichman, and Olivier Boissier. Using the moise+ for a cooperative framework of MAS reorganization. In A.L.C Bazzan and S. Labidi, editors, *Proceedings of the 17th Brazilian Symposium on Artificial Intelligence (SBIA '04)*, pages 506–515, 2004.
- [90] Michael N. Huhns and Larry M. Stephens. Multiagent systems and societies of agents. In Gerhard Weiss, editor, *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*, pages 79–120. The MIT Press, Cambridge, MA, USA, 1999.
- [91] T. Ishida, L. Gasser, and M. Yokoo. Organization self-design of distributed production systems. *IEEE Transactions on Knowledge and Data Engineering*, 4(2):123–134, 1992.
- [92] Alan Julian Izenman. Recent developments in nonparametric density estimation. *Journal of the American Statistical Association*, 86(413):205–224, March 1991.

- [93] N.R. Jennings. Controlling cooperative problem solving in industrial multi-agent systems. *Artificial Intelligence*, 75(2):195–240, 1995.
- [94] David Jensen and Victor Lesser. Social Pathologies of Adaptive Agents. In M. Barley & H. Guesgen, editor, *Safe Learning Agents: Papers from the 2002 AAAI Spring Symposium.*, volume TR SS-02-07. AAAI Press, August 2002.
- [95] David D. Jensen and Paul R. Cohen. Multiple comparisons in induction algorithms. *Mach. Learn.*, 38(3):309–338, 2000.
- [96] Java Toolkit: J/Link. <http://www.wolfram.com/solutions/mathlink/jlink/>.
- [97] Hyuckchul Jung, Milind Tambe, and Shriniwas Kulkarni. Argumentation as distributed constraint satisfaction: applications and results. In *AGENTS '01: Proceedings of the fifth international conference on Autonomous agents*, pages 324–331, New York, NY, USA, 2001. ACM Press.
- [98] Gal A. Kaminka, David V. Pynadath, and Milind Tambe. Monitoring teams by overhearing: A multi-agent plan-recognition approach. *Journal of Artificial Intelligence Research*, 17:83–135, 2002.
- [99] T. Khedro and M. Genesereth. Facilitators: A networked computing infrastructure for distributed software interoperation. In *Working Notes of the IJCAI-95 Workshop on Artificial Intelligence in Distributed Information Networks.*, 1995.
- [100] M. Klein, J.A. Rodriguez-Aguilar, and C. Dellarocas. Using domain-independent exception handling services to enable robust open multi-agent systems: The case of agent death. *Journal of Autonomous Agents and Multi-Agent Systems*, 7(1-2):179–189, 2003.
- [101] L. Kleinrock. *Queueing Systems. Volume I: Theory.* John Wiley & Sons, New York, 1975.
- [102] M. Klusch and A. Gerber. Dynamic coalition formation among rational agents. *IEEE Intelligent Systems*, 17(3):42–47, 2002.
- [103] Arthur Koestler. *The Ghost In The Machine.* Hutchinson Publishing Group, Arkana, London, 1967.
- [104] Karl Kurbel and Iouri Loutchko. Towards multi-agent electronic marketplaces: What is there and what is missing? *The Knowledge Engineering Review*, 18(1):33–46, 2003.
- [105] Kate Larson and Tuomas Sandholm. Anytime coalition structure generation: An average case study. *Journal of Experimental and Theoretical AI*, 11:1–20, 2000.
- [106] James H Lawton. *Distributed Sensor Networks: A Multiagent Perspective*, chapter The Radsim Simulator, pages 11–20. Kluwer Academic Publishers, 2003.

- [107] K. Lerman and A. Galstyan. A general methodology for mathematical analysis of multiagent systems. Technical Report ISI-TR-529, University of California, Information Sciences Institute, 2001.
- [108] K. Lerman and O. Shehory. Coalition formation for large-scale electronic markets. In *Proceedings of the International Conference on Multi-Agent Systems (ICMAS'2000)*, 2000.
- [109] V. Lesser, K. Decker, T. Wagner, N. Carver, A. Garvey, B. Horling, D. Neiman, R. Podorozhny, M. NagendraPrasad, A. Raja, R. Vincent, P. Xuan, and X.Q. Zhang. Evolution of the GPGP/TAEMS Domain-Independent Coordination Framework. *Autonomous Agents and Multi-Agent Systems*, 9(1):87–143, July 2004.
- [110] V. Lesser, C. Ortiz, and M. Tambe, editors. *Distributed Sensor Networks: A Multiagent Perspective (Edited book)*, volume 9. Kluwer Academic Publishers, May 2003.
- [111] Victor Lesser. A Retrospective View of FA/C Distributed Problem Solving. *IEEE Transactions on Systems, Man, and Cybernetics*, 21(6):1347–1363, November 1991.
- [112] Victor Lesser. Reflections on the Nature of Multi-Agent Coordination and Its Implications for an Agent Architecture. *Autonomous Agents and Multi-Agent Systems*, 1:89–111, January 1998.
- [113] Victor Lesser and Daniel. Corkill. Functionally Accurate, Cooperative Distributed Systems. *IEEE Trans. on Systems, Man, and Cybernetics*, SMC-11(1):81–96, January 1981.
- [114] V.R. Lesser and D.D Corkill. The Distributed Vehicle Monitoring Testbed: A Tool for Investigating Distributed Problem Solving Networks. *AI Magazine*, 4(3):15–33, 1983.
- [115] V.R. Lesser and L.D Erman. Distributed Interpretation: A Model and an Experiment. *IEEE Transactions on Computers Special Issue on Distributed Processing*, C-29(12):1144–1163, December 1980.
- [116] Hector J. Levesque, Philip R. Cohen, and José H. T. Nunes. On acting together. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, pages 94–99, July 1990.
- [117] Zhiang Lin and Kathleen Carley. DYCORP: A computational framework for examining organizational performance under dynamic conditions. *Journal of Mathematical Sociology*, 20(2-3):193–218, 1995.
- [118] David Lybäck. Transient diversity in multi-agent systems. Master's thesis, Department of Computer and Systems Sciences, Stockholm University and the Royal Institute of Technology, September 1999.

- [119] Roger Mailler, Bryan Horling, Victor Lesser, and Regis Vincent. The Control, Coordination, and Organizational Design of a Distributed Sensor Network. 2003.
- [120] Roger Mailler and Victor Lesser. Solving Distributed Constraint Optimization Problems Using Cooperative Mediation. In *Proceedings of Third International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2004)*, pages 438–445. IEEE Computer Society, 2004.
- [121] Roger Mailler, Victor Lesser, and Bryan Horling. Cooperative Negotiation for Soft Real-Time Distributed Resource Allocation. In *Proceedings of Second International Joint Conference on Autonomous Agents and MultiAgent Systems (AAMAS 2003)*, pages 576–583, Melbourne, July 2003. ACM Press.
- [122] Thomas W. Malone, Kevin Crowston, Jintae Lee, Brian Pentland, Chrysanthos Dellarocas, George Wyner, John Quimby, Charles S. Osborn, Abraham Bernstein, George Herman, Mark Klein, and Elissa O’Donnell. Tools for inventing organizations: Toward a handbook of organizational processes. *Management Science*, 45(3):425–443, 1999.
- [123] Thomas W. Malone and Stephen A. Smith. Modeling the performance of organizational structures. *Operations Research*, 36(3):421–436, 1988.
- [124] J. G. March and H. A. Simon. *Organizations*. Wiley, New York, 1958.
- [125] Stacy Marsella, Milind Tambe, Jafar Adibi, Yaser Al-Onaizan, Gal A. Kaminka, and Ion Muslea. Experiences acquired in the design of robocup teams: A comparison of two fielded teams. *Autonomous Agents and Multi-Agent Systems*, 4(1/2):115–129, 2001.
- [126] P. Mathieu, J. C. Routier, and Y. Secq. Dynamic organization of multi-agent systems. In *Proceedings of the First International Joint Conference on Autonomous Agents and Multi-Agent Systems*, pages 451–452. ACM Press, 2002.
- [127] Eric Matson and Scott DeLoach. Organization-based adaptive information systems for battlefield situational analysis. In *Proceedings of the IEEE International Conference on Integration of Knowledge Intensive Multi-Agent Systems (IEEE KIMAS '03)*, October 2003.
- [128] Eric Matson and Scott DeLoach. Using dynamic capability evaluation to organize a team of cooperative, autonomous robots. In *Proceedings of The 2003 International Conference on Artificial Intelligence (IC-AI'03)*, pages 744–749, 2003.
- [129] Eric Matson and Scott A. DeLoach. Autonomous organization-based adaptive information systems. In *Proceedings of the IEEE International Conference on Knowledge Intensive Multiagent Systems (KIMAS '05)*, April 2005.

- [130] F. Maturana, W. Shen, and D. Norrie. Metamorph: An adaptive agent-based architecture for intelligent manufacturing. *International Journal of Production Research*, 37(10):2159–2174, 1999.
- [131] Carlos Mérida-Campos and Steven Willmott. Modelling coalition formation over time for iterative coalition games. In *Proceedings of the 3rd International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2004)*, pages 572–579. IEEE Computer Society, August 2004.
- [132] Thomas Montgomery and Edmund Durfee. Search reduction in hierarchical distributed problem solving. *Group Decision and Negotiation*, 2:301–317, 1993.
- [133] Yoram Moses and Moshe Tennenholtz. Artificial social systems. *Computers and AI*, 14(6):533–562, 1995.
- [134] Abbe Mowshowitz. On the theory of virtual organization. *Systems Research and Behavior Science*, 14(6):373–384, 1997.
- [135] Lik Mui, Mojdeh Mohtashemi, and Ari Halberstadt. Notions of reputation in multi-agent systems: A review. In *Proceedings of the First International Conference on Autonomous Agents and MAS*, pages 280–287, Bologna, Italy, July 2002. ACM.
- [136] Prasad Nagendra, Decker M.V., Garvey K., Lesser A., and V. Exploring Organizational Designs with TAEMS: A case study of distributed data processing. *Proceedings of the Second International Conference on Multi-Agent Systems*, pages 283–290, January 1996.
- [137] R. Nair, T. Ito, M. Tambe, and S. Marsella. The role of emotions in multiagent teamwork: A preliminary investigation. In *Who needs emotions: the brain meets the robot*. Oxford University Press, 2003.
- [138] R. Nair, M. Tambe, and S. Marsella. Role allocation and reallocation in multiagent teams: Towards a practical analysis. In *Proceedings of Second International Joint Conference on Autonomous Agents and Multi-agent Systems (AAMAS-03)*, pages 552–559, 2003.
- [139] T. J. Norman, A. Preece, S. Chalmers, N. R. Jennings, M. Luck, V. D. Dang, T. D. Nguyen, V. Deora, J. Shao, W. A. Gray, and N. J. Fiddian. Conoise: Agent-based formation of virtual organisations. In *Proceedings of the 23rd SGAI International Conference on Innovative Techniques and Applications of Artificial Intelligence*, pages 353–366. Springer-Verlag, 2003.
- [140] Christos H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.
- [141] L. E Parker. Designing control laws for cooperative agent teams. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 582–587, 1993.

- [142] H. Van Dyke Parunak, Robert Savit, and Rick L. Riolo. Agent-based modeling vs. equation-based modeling: A case study and users' guide. In *Proceedings of the First International Workshop on Multi-Agent Systems and Agent-Based Simulation*, pages 10–25, London, UK, 1998. Springer-Verlag.
- [143] H. Edward Pattison, Daniel D. Corkill, and Victor R. Lesser. Instantiating Descriptions of Organizational Structures. *Distributed Artificial Intelligence, Research Notes in Artificial Intelligence*, I:59–96, 1987.
- [144] M.V. Nagendra Prasad and V Lesser. Learning Situation Specific Coordination in Cooperative Multi-Agent Systems. *Autonomous Agents and Multi-Agent Systems*, 2:173–207, 1999.
- [145] David V. Pynadath and Milind Tambe. The communicative multiagent team decision problem: Analyzing teamwork theories and models. *Journal of AI research*, 16:389–423, 2002.
- [146] V. Nageshwara Rao and Vipin Kumar. Parallel depth first search, part i: Implementation. *International Journal of Parallel Programming*, 16(6):479–499, 1987.
- [147] S. Reed and V.R Lesser. Division of Labor in Honey Bees and Distributed Focus of Attention. *University of Massachusetts/Amherst Computer and Information Science Department Technical Report 80-17*, November 1980.
- [148] Reiss, R.D. *Approximate Distributions of Order Statistics*. Springer-Verlag, New York, NY, 1989.
- [149] J. Rodriguez, P. Noriega, C. Sierra, and J. Padget. FM96.5 A Java-based Electronic Auction House. In *Proceedings of 2nd Conference on Practical Applications of Intelligent Agents and MultiAgent Technology (PAAM'97)*, pages 207–224, London, UK, April 1997.
- [150] Pierre Romelaer. Organization: A Diagnosis Method. Technical Report 78, University Paris IX Dauphine, Crepa Laboratory, June 2002.
- [151] S. Ross. *Introduction to Probability Models*. Academic Press, Boston, MA, 5th edition, 1993.
- [152] D. Huynh S. D. Ramchurn and N. R. Jennings. Trust in multiagent systems. *The Knowledge Engineering Review*, 19(1):1–25, 2004.
- [153] Jordi Sabater and Carles Sierra. Social regret, a reputation model based on social relations. *SIGecom Exch.*, 3(1):44–56, 2002.
- [154] Tuomas Sandholm. Algorithm for optimal winner determination in combinatorial auctions. *Artificial Intelligence*, 135(1-2):1–54, 2002.

- [155] Tuomas Sandholm. Optimal winner determination algorithms. In *Combinatorial Auctions*. MIT Press, 2005.
- [156] Tuomas Sandholm, Kate Larson, Martin Andersson, Onn Shehory, and Fernando Tohme. Coalition structure generation with worst case guarantees. *Artificial Intelligence*, 111(1-2):209–238, 1999.
- [157] Tuomas Sandholm and Victor Lesser. Coalitions Among Computationally Bounded Agents. *Artificial Intelligence, Special Issue on Economic Principles of Multi-Agent Systems*, 94(1):99–137, January 1997.
- [158] P. Scerri, D. Pynadath, and M. Tambe. Towards adjustable autonomy for the real world. *Journal of Artificial Intelligence Research*, 17:171–228, 2002.
- [159] Paul Scerri, Alessandro Farinelli, Stephen Okamoto, and Milind Tambe. Allocating roles in extreme teams. In *Proceedings of the 3rd International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 1502–1503. IEEE Computer Society, 2004.
- [160] Jens Schmitt and Utz Roedig. Sensor Network Calculus - A Framework for Worst Case Analysis. In *Proceedings of the International Conference on Distributed Computing in Sensor Systems (DCOSS05), Marina del Rey, USA*. IEEE Computer Society Press, June 2005.
- [161] Markus Schwaninger. A theory for optimal organization. Technical Report 38, Institute of Management at the University of St. Gallen, Switzerland, 2000.
- [162] Sandip Sen. Reciprocity: a foundational principle for promoting cooperative behavior among self-interested agents. In *Proc. of the Second International Conference on Multiagent Systems*, pages 322–329, 1996.
- [163] Sandip Sen and Edmund H. Durfee. A formal study of distributed meeting scheduling. *Group Decision and Negotiation*, 1998.
- [164] Onn Shehory and Sarit Kraus. Methods for task allocation via agent coalition formation. *Artificial Intelligence*, 101(1-2):165–200, 1998.
- [165] Onn Shehory, Katia Sycara, Prasad Chalasani, and Somesh Jha. Agent cloning: an approach to agent mobility and resource allocation. *IEEE Communications Magazine*, 36(7):58–67, 1998.
- [166] Jiaying Shen, Xiaoqin Zhang, and Victor Lesser. Degree of Local Cooperation and its Implication on Global Utility. In *Proceedings of Third International Joint Conference on Autonomous Agents and MultiAgent Systems (AAMAS 2004)*, volume 2, pages 546–553, New York, New York, July 2004. IEEE Computer Society.



- [167] W. Shen and D. Norrie. A hybrid agent-oriented infrastructure for modeling manufacturing enterprises. In *Proceedings of the Knowledge Acquisition Workshop (KAW'98)*, pages 1–19, 1998.
- [168] Yoav Shoham and Moshe Tennenholtz. On social laws for artificial agent societies: Off-line design. *Artificial Intelligence*, 73(1-2):231–252, 1995.
- [169] J. Sichman and Y. Demazeau. On social reasoning in multi-agent systems. *Revista Ibero-Americana de Inteligencia Artificial*, 13:68–84, 2001.
- [170] Carles Sierra, Jordi Sabater, Jaume Agusti-Cullell, and Pere Garcia. Evolutionary computation in MAS design. In *Proceedings of ECAI 2002*, pages 188–192, 2002.
- [171] Carles Sierra, Jordi Sabater, Jaume Augusti, and Pere Garcia. SADDE: Social agents design driven by equations. In F. Bergenti, M.P. Gleizes, and F. Zambonelli, editors, *Methodologies and software engineering for agent systems*. Kluwer Academic Publishers, 2004.
- [172] Herbert A. Simon. *The Sciences of the Artificial*. MIT Press, Cambridge, MA, 1968.
- [173] Mark Sims, Daniel Corkill, and Victor Lesser. Separating Application-Specific and Organizational Coordination Issues during Multi-Agent Organizational Design and Instantiation. Computer Science Technical Report 04-98, University of Massachusetts, November 2004.
- [174] Mark Sims, Daniel Corkill, and Victor Lesser. Separating Domain and Coordination in Multi-Agent Organizational Design and Instantiation. In *Proceedings of the International Conference on Intelligent Agent Technology (IAT 2004)*, pages 155–161, Beijing, China, September 2004.
- [175] Mark Sims, Claudia Goldman, and Victor Lesser. Self-Organization through Bottom-up Coalition Formation. In *Proceedings of Second International Joint Conference on Autonomous Agents and MultiAgent Systems (AAMAS 2003)*, pages 867–874, Melbourne, AUS, July 2003. ACM Press.
- [176] Mark Sims, Jim Kurose, and Victor Lesser. Streaming versus Batch Processing of Sensor Data in a Hazardous Weather Detection System. In *Proceedings of Second Annual IEEE Communications Society Conference on Sensor and Ad Hoc Communications and Networks (SECON 2005)*, September 2005.
- [177] Narinder Singh, Michael R. Genesereth, and M. Syed. A distributed and anonymous knowledge sharing approach to software interoperation. *International Journal of Cooperative Information Systems*, 4(4):339–368, 1995.
- [178] Reid G. Smith. The contract net protocol: High-level communication and control in a distributed problem solver. *IEEE Transactions on Computers*, 29(12):1104–1113, 1980.

- [179] Young-pa So and Edmund H. Durfee. Designing tree-structured organizations for computational agents. *Computational and Mathematical Organization Theory*, 2(3):219–246, 1996.
- [180] Young-pa So and Edmund H. Durfee. Designing organizations for computational agents. In *Simulating Organizations*, pages 47–64. AAAI Press/ MIT Press, 1998.
- [181] Leen-Kiat Soh, Costas Tsatsoulis, and Huseyin Sevay. A Satisficing, Negotiated, and Learning Coalition Formation Architecture. In Victor Lesser, Charles Ortiz, and Milind Tambe, editors, *Distributed Sensor Networks: A multiagent perspective*, pages 109–138. Kluwer Academic Publishers, 2003.
- [182] Dan Stefanoiu, Mihaela Ulieru, and Douglas Norrie. Fuzzy modeling of multi-agent systems behavior. vagueness minimization. In *Proceedings of World Multiconference on Systemics, Cybernetics and Informatics (SCI'2000)*, volume III, pages 118–123, July 2000.
- [183] Michael Stonebraker, Paul M. Aoki, Witold Litwin, Avi Pfeffer, Adam Sah, Jeff Sidell, Carl Staelin, and Andrew Yu. Mariposa: A wide-area distributed database system. *VLDB Journal: Very Large Data Bases*, 5(1):48–63, 1996.
- [184] Katia Sycara, K. Decker, and M. Williamson. Middle-agents for the internet. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence*, pages 578–583, January 1997.
- [185] Milind Tambe. Towards flexible teamwork. *Journal of Artificial Intelligence Research*, 7:83–124, 1997.
- [186] Milind Tambe. Representational conventions for using steam. March 1997.
- [187] Milind Tambe, Jafar Adibi, Y. Alonazon, Ali Erdem, Gal A. Kaminka, Stacy Marsella, and Ion Muslea. Building agent teams using an explicit teamwork model and learning. *Artificial Intelligence*, 110(2):215–239, 1999.
- [188] G. Tidhar, C. Heinze, and M. Selvestrel. Flying together: Modelling air mission teams. *Journal of Applied Intelligence*, 8(3):195–218, May 1998.
- [189] Gil Tidhar, Anand Rao, and Liz Sonenberg. Guided team selection. In *Proceedings of the 2nd International Conference on Multi-agent Systems (ICMAS-96)*, pages 369–376, Kyoto, Japan, 1996.
- [190] Maksim Tsvetovat and Katia Sycara. Customer coalitions in the electronic marketplace. In Carles Sierra, Maria Gini, and Jeffrey S. Rosenschein, editors, *Proceedings of the Fourth International Conference on Autonomous Agents*, pages 263–264, Barcelona, Catalonia, Spain, 2000. ACM Press.

- [191] M. Tsvetovatyy, M. Gini, B. Mobasher, and Z. Wieckowski. MAGMA: An agent-based virtual market for electronic commerce. *Journal of Applied Artificial Intelligence*, 11(6):501–523, 1997.
- [192] R. M. Turner. The tragedy of the commons and distributed AI systems. In *Proceedings of the 12th International Workshop on Distributed Artificial Intelligence*, pages 379–390, Hidden Valley, Pennsylvania, 1993.
- [193] Mihaela Ulieru. Emergence of holonic enterprises from multi-agent systems: A fuzzy-evolutionary approach. In Vincenzo Loia, editor, *Soft Computing Agents: A New Perspective on Dynamic Information Systems*, pages 187–215. IOS Press, 2002.
- [194] Mihaela Ulieru, Scott Walker, and Robert Brennan. Holonic enterprise as a collaborative information ecosystem. In *Proceedings of the Workshop on Holons: Autonomous and Cooperating Agents for Industry 2001*, pages 1–14, May 2001.
- [195] Mashall van Alstyne. The state of network organization: a survey in three frameworks. *Journal of Organizational Computing and Electronic Commerce*, 7(2&3):83–151, 1997.
- [196] Javier Vázquez-Salceda, Virginia Dignum, and Frank Dignum. Organizing multiagent systems. Technical Report UU-CS-2004-015, Institute of Information & Computing Sciences, March 2004.
- [197] William Vickrey. Counterspeculation, auctions, and competitive sealed tenders. *Journal of Finance*, 16(1):8–37, 1961.
- [198] Eberhard O. Voit, W. Leonard Balthis, and Ronald A. Holser. Hierarchical monte carlo modeling with s-distributions: Concepts and illustrative analysis of mercury contamination in king mackerel. *Environment International*, 21(5):627–635, 1995.
- [199] Thomas Wagner and Victor Lesser. Relating Quantified Motivations for Organizationally Situated Agents. In *Intelligent Agents VI — Proceedings of the Sixth International Workshop on Agent Theories, Architectures, and Languages, Lecture Notes in Artificial Intelligence*, pages 334–348. N. R. Jennings and Y. Lesperance (eds.), Springer-Verlag, Berlin, April 1999.
- [200] A. Walker and M. J. Wooldridge. Understanding the emergence of conventions in multi-agent systems. In *Proceedings of the First International Conference on Multi-Agent Systems*, pages 384–389, San Francisco, CA, 1995.
- [201] M. Wellman, W. Walsh, P. Wurman, and J. MacKie-Mason. Auction protocols for decentralized scheduling. Technical report, University of Michigan, July 1998.

- [202] M. Wellman and P. Wurman. A trading agent competition for the research community. In *Proceedings of the IJCAI-99 Workshop on Agent-Mediated Electronic Trading*, August 1999.
- [203] Michael Wellman. A market-oriented programming environment and its application to distributed multicommodity flow problems. *Journal of Artificial Intelligence Research*, 1:1–23, 1993.
- [204] Michael Wellman. Online marketplaces. In Munindar P. Singh, editor, *Practical Handbook of Internet Computing*. Chapman Hall & CRC Press, Baton Rouge, 2004.
- [205] Michael Wellman and Peter Wurman. Market-aware agents for a multiagent world. *Robotics and Autonomous Systems*, 24:115–125, 1998.
- [206] G. Wiederhold, P. Wegner, and S. Cefi. Toward megaprogramming. *Communications of the ACM*, 33(11):89–99, 1992.
- [207] Wikipedia. Bogomips — wikipedia, the free encyclopedia, 2005. [Online; accessed 18-Aug-2005].
- [208] S. Willmott, J. Dale, B. Burg, P. Charlton, and P. O’Brien. Agentcities: A worldwide open agent network. *Agentlink Newsletter*, 8:13–15, November 2001.
- [209] Wolfram Research, Inc. *Mathematica*. Wolfram Research, Inc., Champaign, Illinois, 2004.
- [210] P. Wurman, M. Wellman, and W. Walsh. A parameterization of the auction design space. *Games and Economic Behavior*, 35(1-2):304–338, 2001.
- [211] Fabiola López y López and Michael Luck. Modelling norms for autonomous agents. In *Proceedings of the Fourth Mexican International Conference on Computer Science (ENC’03)*, pages 238–245. IEEE Computer Society Press, 2003.
- [212] Fabiola López y López, Michael Luck, and Mark d’Inverno. Normative agent reasoning in dynamic societies. In *Proceedings of the 3rd International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2004)*, pages 732–739. IEEE Computer Society, 2004.
- [213] O. Yadgar, S. Kraus, and C. Ortiz. Scaling up distributed sensor networks: cooperative large-scale mobile-agent organizations. In V. Lesser, C. Ortiz, and M. Tambe, editors, *Distributed Sensor Networks: a multiagent perspective*, pages 185–218. Kluwer publishing, 2003.
- [214] Haizheng Zhang, Bruce Croft, Brian Levine, and Victor Lesser. A Multi-agent Approach for Peer-to-Peer based Information Retrieval System. In *Proceedings of Third International Joint Conference on Autonomous Agents and MultiAgent Systems*, pages 456–464, NYC, July 2004.

- [215] Haizheng Zhang and Victor Lesser. A Dynamically Formed Hierarchical Agent Organization for a Distributed Content Sharing System . In *Proceedings of the International Conference on Intelligent Agent Technology (IAT 2004)*, pages 169–175, Beijing, September 2004. IEEE Computer Society.
- [216] X. Zhang and D. Norrie. Holonic control at the production and controller levels. In *Proceedings of IMS 99*, pages 215–224, 1999.
- [217] XiaoQin Zhang, Victor Lesser, and Thomas Wagner. Integrative Negotiation in Complex Organizational Agent Systems. In *Proceedings of the 2003 IEEE/WIC International Conference on Intelligent Agent Technology (IAT 2003)*, pages 140–146, Halifax, Canada, 2003. IEEE Computer Society.