Using Quantitative Models to Search for Appropriate Organizational Designs *

Bryan Horling† and Victor Lesser
Department of Computer Science
University of Massachusetts, Amherst, MA
{bhorling,lesser}@cs.umass.edu

Abstract

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 different roles, responsibilities and peers. These additional constraints can allow agents to operate more efficiently within the system by limiting the options they must consider. Different designs applied to the same problem will have different performance characteristics, therefore it is important to understand the behavior of competing candidate designs.

In this article, we describe a new representation for capturing such designs, and in particular we show how quantitative information can form the basis of a flexible, predictive organizational model. The representation is capable of capturing a wide range of multi-agent characteristics in a single, succinct model. We demonstrate the language's capabilities and efficacy by comparing a range of metrics predicted by detailed models of a distributed sensor network and information retrieval system to empirical results. These same models also describe the space of possible organizations in those domains and several search techniques are described that can be used to explore this space, using those quantitative predictions and context-specific definitions of utility to evaluate alternatives. The results of such a search process can be used to select the organizational design most appropriate for a given situation.

1 Introduction

The notion of 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 domain problem solving and control roles and responsibilities 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. For example, the design may specify a particular set of peers an agent should interact with, or prioritize the agent's responsibil-

*This material is based upon work supported in part by the National Science Foundation Engineering Research Centers Program under NSF Award No. EEC-0313747. Any opinions, findings, conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation. Effort also sponsored in part by the Defense Advanced Research Projects Agency (DARPA) and Air Force Research Laboratory Air Force Materiel Command, USAF, under agreement number F30602-99-2-0525. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Defense Advanced Research Projects Agency (DARPA), Air Force Research Laboratory or the U.S. Government.

†Currently employed at Google, Inc.

ities. Lacking such information, the agent would likely need to expend time and resources to reach the same operational state.

This additional structure becomes increasingly important as the system scales in number and scope (2). 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 agent that is responsible for controlling the sensor. Let us further assume that the sensor nodes must collaborate in some way to be successful. For example, 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 about which sensors to assign for specific tasks, 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 addressed in this paper revolve around finding a general way to determine the most appropriate organizational strategy for a given environment and set of organizational objectives 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 type and frequency of 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 tran-

sitions or tipping points require a deeper understanding and a more concrete representation to be addressed.

Understanding the basis for behaviors like those described above that occur within and between agents, 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. Although mathematical models for particular aspects of organizationally-driven agent behaviors have been created (34; 22; 32; 5; 12), none of these works have explored the utility of a general modeling language capable of incorporating arbitrary quantitative information. If we are to understand these organizational 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 quantitative characteristics each design will exhibit.

Many different representations have been created to describe agent organizations (40; 6; 19; 26; 7; 35; 9; 11; 21; 17; 36). 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 paper we will describe a representation, the Organizational Design Modeling Language (ODML), that is both flexible and quantitative. This language is designed to capture organizational information across many different domains at different levels of abstraction in a single unified, predictive structure. At the same time, it is able to integrate concrete numeric information in the form of expressions and predictive equations using a range 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. These models can then be used as part of a search process to evaluate and rank candidate organizational designs.

The following section will expand on the motivations behind ODML, and its relation to existing work. Section 3 will provide details describing the language itself and the rationale behind its design. In Section 4, we also discuss and evaluate modeling strategies for three domains to both demonstrate how the language can be used and provide examples of the detailed predictions that are possible in ODML. Section 4 will show how these models can be used as part of an organizational design search process. The complexity of such a search is determined, and several techniques described that can help cope with this complexity. Section 5 will summarize our conclusions and future directions.

2 Motivation and Related Work

The rationale behind ODML's design was to create an organizational representation with several key features. It should first be able to represent a wide range of designs across different domains possessing different relevant metrics. Second, an individual model should be able to define a class of designs, each with potentially different runtime characteristics. Finally, for each possible design, that same model should be able to predict concrete values for the metrics and characteristics deemed important by the designer. Together, these features allow a particular model to define the space of organizational possibilities and provide a means by which those possibilities can be compared.

An example ODML structure for a distributed sensor application can be seen in Figure 1. The details behind the structure will be covered in Section 3, it is sufficient here to note that ODML is used to create graph-based models consisting of interrelated *nodes*. Each node corresponds to a particular organizational component, such as an agent, a role. Nodes in this model include *sensor*, *sector*, and *track manager*, among others. Each node contains a set of *fields*, which use equations to describe the characteristics of that node. Several different types of interactions allow the fields of one node to affect those of another. Taken together, these fields and interactions produce a "web of equations" that can be used to predict the various characteristics of the functioning organization.

The idea of quantitatively modeling a system is not a new one. For example, Shen et al. (34) 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 (5) motivate the need for meta-level communication to

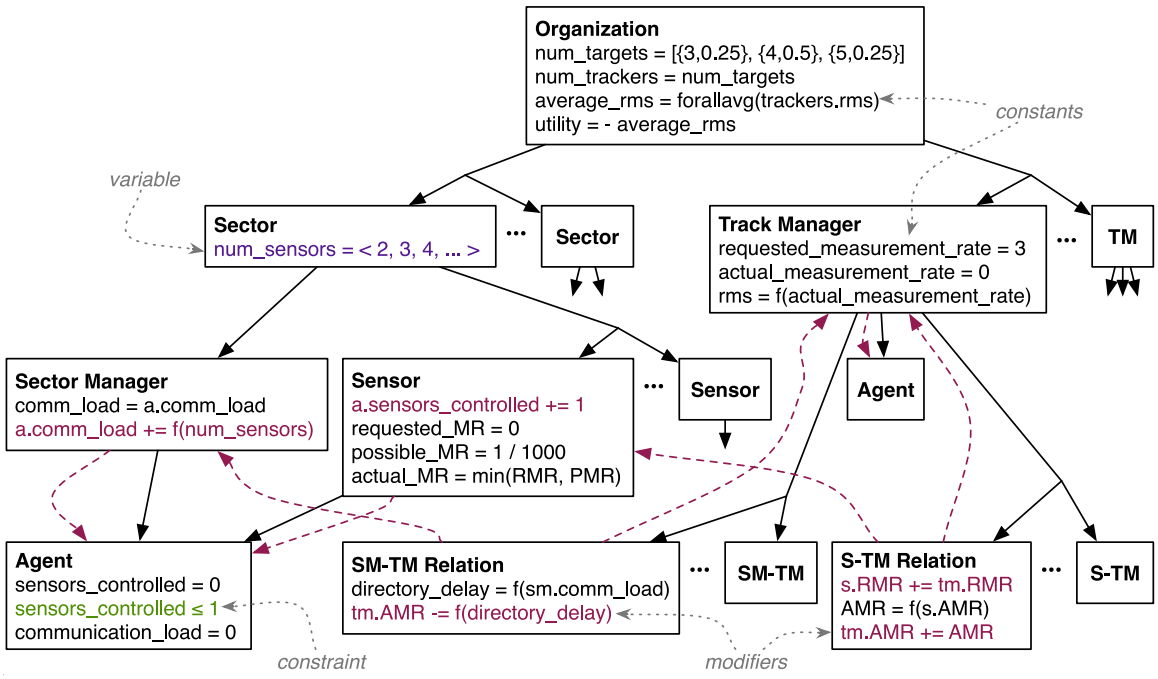


Figure 1: An example ODML structure capturing organizational interactions present in a distributed sensor network design. Rectangles represent ODML nodes, and the embedded expressions those nodes' fields. Solid edges represent has-a relationships, while dashed edges show relationships formed by nonlocal modifiers. The light gray labels with dotted edges give the type names for the reader, and are not part of the actual structure.

guide dynamic organizational adaptation when conditions are uncertain. So and Durfee (38) created a quantitative model of a tree-based organization, to predict how various types of hierarchies will behave under difference circumstances. Sen and Durfee (32) 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. (22), Gnanasambandam, et al. (12) and Schmitt and Roedig (31) 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 the underlying techniques they employ have been incorporated successfully into ODML models, they do not address the same class of problems as that representable in ODML.

More closely related to ODML's intent is the large set of existing organizational representations that have been created (40; 6; 19; 26; 7; 23; 35; 9; 11; 21; 17; 36). Table 1 contrasts the features supported by these different systems. As mentioned in the previous section, none of these exist in the flexible-but-quantitative space that ODML occupies. For example, OMNI (9) and $\mathcal{M}$OISE$^+$(17) 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. The MaSE system designed by Matson and DeLoach (23) does dynamically compute the quantitative utility of an organiza-

Framework	Organizational Alternatives	Organizational Knowledge	Heterogeneous Objectives	Explicit Task Structure	Current Structure	Quantitative Relations	Deductive Capability	Hard Constraints	Soft Constraints
ODML	✓	✓	✓		✓	✓	✓	✓	✓
OMNI		✓	✓		✓				
$\mathcal{M}$OISE$^+$	✓	✓	✓	✓	✓			✓	
MaSE	✓	✓	✓	✓	✓			✓	
SADDE		✓	✓		✓	✓		✓	✓
Process Handbook	✓	✓	✓	✓	✓	✓			
Sims	✓	✓	✓	✓	✓		✓	✓	✓

Table 1: A comparison of the characteristics and capabilities of several different organizational representations.

tion, but does so using only a single approximate quality statistic. The previous section provided an intuition of how many different interacting features affect the utility of an organizational design. Conversely, both SADDE (35) and MIT's Process Handbook (21) 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 (36) does incorporate quantitative information into a structured organizational model, but we 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.

A feature lacking in ODML but supported in all the contrasting systems is the ability to explicitly represent a task structure. Task structures are generally used to determine the responsibilities of the various components in the system, which ultimately affects the viability of alternative organizational designs. ODML captures such requirements and the effects they have on organization performance within the structure itself. For example, the structure shown in Figure 1 has both externally defined task responsibilities (*num_targets* in the Organization level) and organizational decisions that affect task allocation (*num_sensors* in Sector). This design choice provides for a variety of ways to model tasks and keeps the ODML language itself simple. Conversely, it also makes this knowledge somewhat less explicit, and prevents the language from offering much guidance on how to represent such information. This is a tradeoff that we have made consciously throughout the design of ODML, which we will return to in Section 3.1.

Each of the representations mentioned above has its strengths and ODML's goal is not to supplant these works, but to demonstrate another approach that makes different tradeoffs. As will be shown in Section 3, 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.

3 Quantitative Organizational Models

As shown in Figure 1, ODML is used to create graph-based models, consisting of a set of nodes that represent organizational components and edges that represent interactions between those components. 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 O is defined as follows:

$$\begin{aligned} O &= \{\mathcal{N}, H, C, K, M, V\} \\ \mathcal{N} &= \{N_0, N_1, \ldots, N_n\} \\ N_i &= \{t, \ell, \bar{p}, I, H, C, K, M, V\} \end{aligned} \tag{1}$$

Examples of how these features are used in practice are given later in the section. 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 is an identifier that must be unique within the set of template nodes that make up the organization.

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

ℓ 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.

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

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.

$$N.I = \{\langle type \rangle, \ldots\}$$
$$\forall i \in N.I, \exists N' \in \mathcal{N} | N'.t = i \wedge N'.t \neq N.t \wedge$$
$$\forall i' \in N'.I, \exists N'' \in \mathcal{N} \ldots$$

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. The magnitude may be defined as a simple number or in terms of a constant or variable symbol defined elsewhere.

$$N.H = \{\langle symbol, type, magnitude \rangle, \ldots]$$
$$magnitude \in \mathbb{Z}^+ \cup \{s | \exists_c \langle s, c \rangle \in N.C \cup N.V\}$$

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., $x + y$).

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

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

$$N.K = \{\langle symbol, op, expression \rangle, \ldots\}$$
$$op \in \{<, >, \leq, \geq, =, \neq\}$$

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.

$$N.M = \{\langle symbol, op, expression \rangle, \ldots\}$$
$$op \in \{+, -, \times, \div\}$$

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 $[2.7, y^2, \pi z]$.

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

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. We will use dot notation to indicate characteristics

of particular objects. *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. ODML supports the use of floating point values, lists of floating point values, and discrete probabilistic distributions.

C, K, M, V are collectively known as a node's *fields*, and the quantitative state of a field as its *value*. For clarity, the names of nodes and fields are represented in italics. For example, Figure 1 shows that the *organization* node has a constant *average_rms* that is defined in terms of the *rms* fields of the *trackers* (i.e., the expected root-mean-squared triangulation error of each tracker). Note that the use of the term "constant" may initially be misleading. While the expression defining *average_rms* is fixed, the value for *average_rms* 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 (in this case, in the *rms* fields of the track managers). The top-level organization node O also contains the elements H, C, K, M, V, providing a location for the designer to embed additional global information and constraints.

Through modifiers or the assimilation of nonlocal values, the characteristics of one node may affect or be affected by those of another. ODML models are generally constructed by designing individual nodes, and linking them through nonlocal dependencies or modifiers. 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 correctly predict the characteristics of both individual nodes and the organization as a whole.

The manner in which field's values are determined is defined by the pseudocode in Figure 2 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 and distribution-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, $organization.get\_value(average\_rms)$ would return a prediction of the *organization*'s *average_rms*.

ODML instances are quite similar to templates. The difference is that where a template is a description of what

```
get_value(symbol s)
    r ← null
    if (s is of the form s1.s2)
        n ← N.get_value(s1)
        r ← n.get_value(s2)
    else if (∃ c ∈ C | c.symbol = s)
        r ← N.evaluate(c.expression)
    else if (∃ h ∈ H | h.symbol = s) r ← h
    else if (∃ v ∈ V | v.symbol = s)
        r ← N.evaluate(v.expression)
    else if (∃ p ∈ p̄ | p.symbol = s) r ← p
    else forall i ∈ I
        r ← i.get_value(s)
        if (r ≠ null) break
    forall m ∈ M
        if (m.symbol = s)
            r ← r m.op N.evaluate(m.expression)
    forall n ∈ 𝒩
        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 }
        v_s ← N.get_value(s)
        substitute all occurrences of s ∈ e with v_s
    r ← mathematical result of e
    return r
```

Figure 2: Pseudocode for the get_value function of a node N, used to quantify the characteristics of instance nodes.

could be, an instance is a description of what *is*. Where a template like that shown in Figure 3a might specify that a *sector_manager* role can be assigned to a single *agent* or distributed across multiple *agent* nodes, a corresponding 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 1, so we do 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 we indicate where necessary which is being considered.

The process of finding an appropriate organization revolves around first finding the set of *valid* instances, and selecting from that set the one that is most desirable. Section 4 will describe the search process used to traverse this space of instances. A valid design instance is one where the number of instances of a node type does not exceed the type's limit ℓ and all the constraints in each node's set K are satisfied. More formally, the validity of a particular organizational instance O is defined as:

$$\begin{aligned} &O \text{ is valid iff } \forall N \in O.\mathcal{N}, N \text{ is valid} \\ &N \text{ is valid iff } \textstyle\sum_{M \in O.\mathcal{N} | M.t = N.t} 1 \leq N.\ell \wedge \forall k \in N.K, \quad (2) \\ &\quad (N.get\_value(k.symbol)\ k.op \\ &\quad N.evaluate(k.expression)) = true \end{aligned}$$

The "desirability" of instance O can be quantified by defining a *utility* characteristic in the organization. This can then be computed using the existing machinery by calling $O.get\_value(utility)$. Because *utility* is defined like any other field, it may be defined as an arbitrary combination of the characteristics of other components of the organization. Therefore, the utility value may vary depending on the choices made to bind variables, the nodes used to satisfy is-a or has-a relations, and the quantitative interactions encoded in equations and modifiers. Once such a value has been computed for all candidate organizational designs, they may be ranked and the best selected.

3.1 Representational Flexibility

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. 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.

One might go further to argue that, because nearly all concrete, organizational concepts have been shed to maximize flexibility, ODML may not properly be called an organizational modeling language at all. For example, a formalism consisting of quantitative relationships, a general graph-based structure and constraints can be used to model many domains outside of organizational design. We contend that although the nomenclature used by ODML may not explicitly be organizational, that the spirit in which it was designed is decidedly so. The features provided by ODML were chosen specifically to address the automated organizational design problem, where structural decisions have quantitative ramifications, and one must be able to both enumerate and rank the space of such possibilities. We believe it is precisely the conventional nomenclature used by other approaches, and the assumptions that generally underlie it, that forces designers down paths that may not be appropriate for the problem they face. Although having such built-in concepts 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 concepts. 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 computationally unwieldy. In another case, the agent characteristics supported by the language may be insufficient or inappropriate to capture the nature of the domain in question. Section 3.2 shows that important organizational characteristics are frequently domain-specific, and therefore unlikely to be present in a language that limits the representable features to some predefined set. Because ODML makes no such assumptions, the first designer could choose to omit an explicit representation of each agent, creating a more abstract but correspondingly more scalable model. The second designer has the freedom to incorporate those characteristics they deem appropriate, including those that are specific to their domain. So, although typical organizational constructs are largely absent from ODML, we feel that the benefits that such a representation offers are worth exploring the tradeoff.

The drawback to having a language lacking in high-level concepts is that it can make both the model design and search processes more difficult. For example, in languages that explicitly support agents and roles it is usually clear how a system employing those concepts should be modeled. In ODML, the designer is given the ability to capture those concepts in whatever manner is appropriate, but in doing so it does place additional demands on the designer. In general, "things" such as agents, resources and objects should be defined as either nodes or, if the inner details of those things are not needed or not relevant, as simple fields. Less tangible features such as roles and groups may also be modeled as nodes, while membership or organizational relationships can be defined with has-a relations. Interactions, communication or dependencies are generally defined as modifiers. State and knowledge are typically represented as constant fields, while the functional or practical limits are defined as constraints. Like system design, however, these are not hard-and-fast rules, and the exact mapping from system to model is a product of the needs and creativity of the designer. Section 3.2 will provide some concrete examples of how realistic systems have been modeled.

From a search perspective, higher-level languages can also embed into the design search process the idea that (for example) roles must be bound to agents or that tasks exist that must be scheduled, and create heuristics or strategies specifically designed to address those needs relationships (e.g., (36; 30; 24; 3)). Because ODML lacks such structural landmarks, its search process can only attempt to infer these situations. Given that such requirements might be modeled in different ways by different designers (or omitted altogether as above), the applicability of concept-specific techniques such as this 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. Section 4 returns to this issue, discussing the complexity of the search, and describe several techniques that have been employed to make the search process tractable and efficient.

3.2 Example Organizational Models

An important benefit from ODML's approach is the ability to model the domain-specific characteristics of a variety of organizational styles. This section briefly describes three such models, the first for a distributed sensor network, the second for an distributed information retrieval system and the last for a coalition. For each of these domains we will describe the high level objectives and the features and interactions we have found to be most important, and then outline how those concepts are captured by ODML models. The first two models, being based on running systems, will also be evaluated against those systems. We have also included the actual ODML model representing the information retrieval system in Appendix A.

The models presented in this section are included first as concrete examples of the language being used, to provide some insight into how various characteristics may be captured. They are also a demonstration of the types of features that can be accurately modeled, and the expressiveness of the language in general. They are not intended to show the limits of what is possible, but to show that complex, realistic interactions can be captured within the ODML framework.

The sensor network model has been described previously in (15), so only a summary is given here. More detail is given for the information retrieval model, while the coalition description discusses how a different types of coalitions might be modeled at a high level. Additional information for these models can be found in (13), which also contains descriptions of ODML models that have been created for several other organizational characteristics and paradigms not covered by this article, including teams, federations, marketplaces, and characteristics that are affected by location or the passage of time.

3.2.1 Distributed Sensor Network

The distributed sensor network (DSN) application described in this section is a more complex version of that presented in Section 1, although it retains the same objective to track targets that move through an environment. It was designed prior to the existence of ODML, making it an ideal platform to gauge ODML's ability to accurately depict the characteristics of a real-world system. The system employs an explicit organizational design that is composed of several different elements. This begins by dividing the environment into a series of logical *sectors*, which are intended to explicitly limit the interactions needed between sensors. There are also three types of responsibilities, or *roles*, that agents may take on: *sector manager*, *track manager* and *sensor*. Each role specifies behaviors, responsibilities and interactions that must be enacted by the agent it is assigned to. Agents can take on multiple roles.

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. The DSN architecture, comprising roughly 40,000 lines of Java code and described in detail in (20), has been demonstrated in both simulation and real-world experiments.

An abbreviated view of the ODML model for this domain that includes field expression information is shown in Figure 1, and a complete structural view of the template and instance models is shown in Figure 3. Space precludes showing the complete textual specification, which is roughly 300 lines long (it can be found in (13)). Vertices in the graph, such as *sensor* and *track_manager*, directly map to the organizational components described above. Note that nodes can represent both tangible (e.g. *agent*) and intangible (e.g. *sector*) entities. Edges in the graph represent the interaction between those components. For example, the solid edge from *sector_manager* to *agent* models the role-agent relationship. The dashed edge from *s_tm_relation* shows how a modifier is used to pass the demand for measurements from the *track_manager* to the *sensor*.

A range of characteristics have been incorporated into the model, including the physical and task environment, agent interactions, single and multiple role assignments, dynamic role assignment, heterogeneity, geographic heterogeneity, potential conflicts, and both hard and soft constraints. Each was successfully modeled, suggesting that the relatively modest set of primitives offered by ODML is capable of representing a wide range of complex and relevant organizational factors. Although ODML lacks a true general notion of time, we have been able to incorporate the notion of a bounded temporal window into this model. This allows one to inspect organizational characteristics at different points in time, so that performance predictions can depend on features that may change over time (e.g., the movement pattern of targets). This is important because organizations may respond differently to different sequences of events. In the DSN domain, for example, you might require a different organizational design depending on whether the targets are consistently diffuse or periodically concentrated. Additional details describing this can be found in (13).

Our previous work analyzed the effects that this organization had on performance across a range of metrics (15). In those tests, the number of agents in each sector was varied to demonstrate how changing the organization can have far-reaching consequences. Because the total number of agents remained constant, varying the number of agents per sector altered the number of sector managers. This in turn affected the number of roles each agent took on, the number of relationships they formed, the load incurred by those responsibilities and the overall performance of the system.

To gauge the representational efficacy of ODML, we have used the model described in the previous section to create organizational instances that match those prior test runs. Characteristics defined in the ODML model make predictions for the same metrics that were originally tested, allowing us to calculate values that can be compared against the empirical results. These predictions were obtained using the *get_value* function described in Figure 2. This exercise both demonstrates how ODML can be used as a predictive tool for different operating contexts, and evaluates how well a specific model was able to capture real-world behaviors.

The comparative results are shown in Figure 4. Note that the behavioral details behind these results are beyond the scope of this document (they can be found in (15)). In this context, we are exploring only the accuracy of the ODML model's predictions (i.e., the ability of

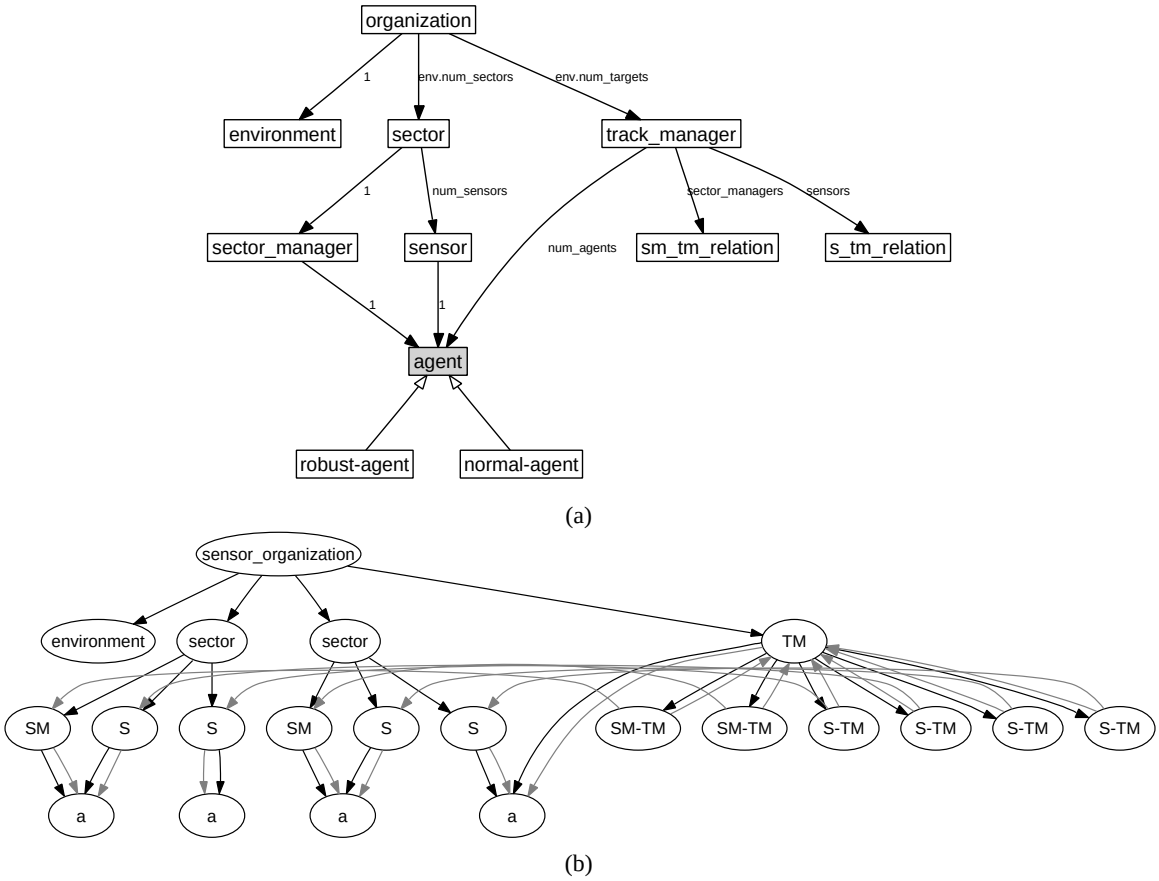
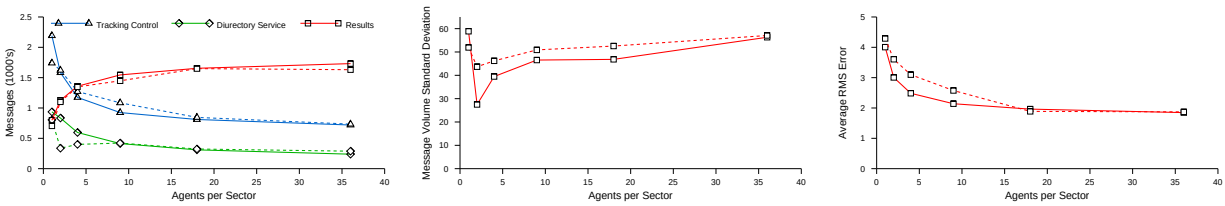


Figure 3: Example ODML (a) template and (b) instance structures for the sensor network organization. As previously, edges with solid heads are has-a relationships, while hollow-headed edges represent is-a relationships. Grey edges are modifiers.

a parameterized web of equations to correctly predict the performance of a class of domain-specific organizations). Solid lines represent the values predicted by the ODML model, while dashed are those obtained through the previous empirical testing. Figure 4a shows communication totals by type. Figure 4b shows the communication disparity, which measures how well or poorly the communication load is distributed in the population. Figure 4c shows the average root-mean-squared (RMS) error of the tracking tasks. Although there are some points of difference, in most cases the model does a good job predicting performance. One difference can be seen in Figure 4b, 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. The model as described here does not capture these geographic differences, and will therefore generally have a lower estimated deviation. However, as shown in (13) it is possible to model the movement patterns of targets within the ODML model, and use that more detailed knowledge to predict load imbalances caused by geographic location if the additional accuracy is needed. This is a good example of the type of tradeoff that ODML permits the designer to make between the simplicity of the model and the level of detail present in the model's subsequent predictions. We



(a) (b) (c)

Figure 4: ODML DSN model predictions versus empirical observations for a) Message totals by type, b) Messaging disparity and c) RMS error. Predicted lines are solid, empirical are dashed.

(a) (b) (c)

Figure 5: Comparison of the ODML DSN model's role-specific predictions.

will revisit this ability in the context of search in Section 4.2.4.

To evaluate how this model captures finer-grained details, we compared the communication profiles of individual roles, as seen in Figure 5. In addition to communication totals, these graphs also include role counts, indicating how many agents took on the specified role. 'A' represents the *sensor* role, 'M' is the *sector_manager*, while 'T' is the *track_manager*. 'AM' describes agents acting as both sensors and sector managers. Predictions at this more detailed level are also accurate. Many of the differences that do exist can be attributed to geographic variances in a small sample size. For example, the 36- and 18-size scenarios had only one or two sector managers. Their individual geographic locations can affect performance, and these variations are not reflected in the predicted values.

3.2.2 Information Retrieval

As a second example, we will describe a model of a peer-to-peer information retrieval system. A general peer-to-peer information retrieval (IR) system is composed of a number of interconnected databases, controlled by a set of entities (agents, in this case). 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.

Zhang and Lesser (42) propose that a hierarchical organization can be used to address this problem. Their solution organizes information sources into a set of hierarchies, allowing queries to quickly propagate to data sources, and results be routed back to a single agent in the network. At the top level of each hierarchy is a *mediator*. Each mediator is responsible for providing a concise and accurate description, known as a collection signature, of the data available in the information sources present in the hierarchy below it. An information source may be an in-

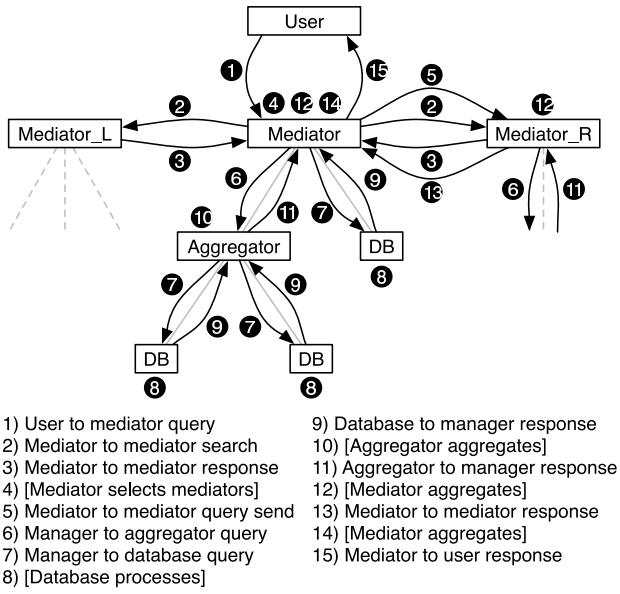


Figure 6: The control and communication sequence involved in handling a query in the information retrieval organization.

dividual *database*, or an *aggregator* which manages other sources. Mediators are responsible for handling the user queries, by first using the signatures of other mediators to compare data sources, then routing the query to those mediators that seem appropriate, and finally collecting and delivering the resulting data.

The sequence of messages and actions performed by the organization handling a query is summarized in Figure 6. In that example, Mediator receives the user query and searches all three nearby mediators. The search response (3) from Mediator_L indicates it has no relevant information, so only Mediator and Mediator_R are selected to handle the query.

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 parallelism into the query distribution process. These same hierarchies also distribute the communication and processing load.

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. Because each agent is a bounded resource, 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.

An organizational model for this system was created using ODML. The structural designs of the IR template and instance models are shown in Figure 7. The underlying textual specification for the template can be found in Appendix A, and we will describe some of the details of this model more thoroughly below. Like the sensor network model above, this model uses notions of roles, a task environment and performance constraints. However, other, more domain-specific phenomena that must be captured are significantly different, and drive the shape of the organization in different directions. These include predicting the quality of the result that will be returned for a user query and determining how load distribution affects the system's response time.

The system's response quality, defined in the model as *response_recall*, measures what proportion of the relevant information was returned to the user. This value depends on what mediators are used to service the query, which can be determined by estimating the query load at each mediator. 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 searched for and used to answer the query. To estimate this, we must first determine the relative rank ordering m_r in terms of information content of the mediator in question m, and the number of mediators R_r that share that ranking.

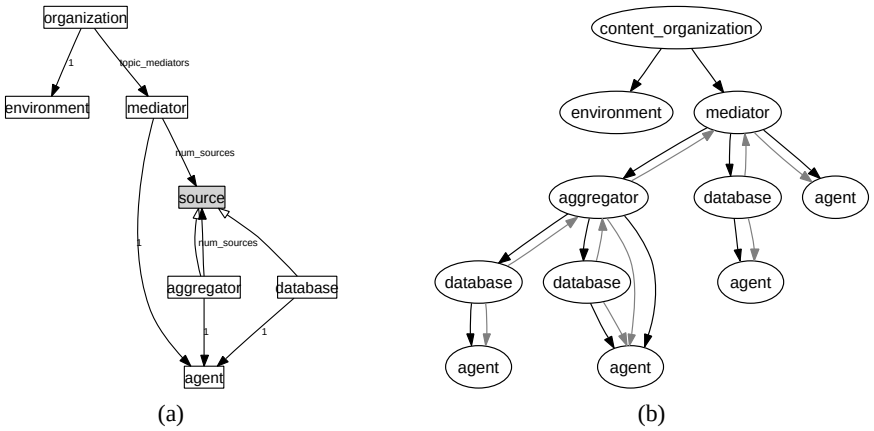


Figure 7: a) An ODML template structure for the information retrieval domain. b) A small organizational instance produced from that template.

$$m_r = 1 + \Big(\sum_{k \in O.mediators} 0^{max(m.prs - k.prs, 0)} - 0^{abs(m.prs - k.prs)} \Big) \quad (3)$$

$$R_r = \sum_{k \in O.mediators} 0^{abs(m_r - k_r)} \quad (4)$$

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. Because this rank ordering is used to select which mediators will be used to service a query, it is possible to use this to compute the probability $P(m)$ that mediator m will be selected.

$$P(m) = \frac{s}{|M|} \frac{1}{\binom{|M|-1}{s-1}} \Big(\sum_{i=0}^{q-1} \sum_{j=0}^{\min(s,R_r)-1} \binom{m_r - 1}{i} \binom{R_r - 1}{j} \binom{|M| - m_r - R_r + 1}{s - i - j - 1} \min\Big(1, \frac{q-i}{j+1}\Big) \Big) \quad (5)$$

Where $|M|$ is the total number of mediators, s is the number of mediators that will be searched and compared (*search_set_size*), and q is the number of mediators that will be given the query (*query_set_size*). Equation (5) models the search process and subsequent mediator selection that will take place when a query is received by the system. Equations 3, 4, and 5 all exist in the *mediator* node of the ODML model. They can be found in the *rank*, *rank_ties*, and *query_probability* constants, respectively, of the *mediator* definition in Appendix A.

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 and team-based systems (39; 16; 41), so it is worth discussing the equation in greater detail.

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 $\frac{s}{|M|}$. 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 possible mediator combinations from the search $\binom{|M|-1}{s-1}$ provides the final desired probability. The summations iterate 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. There are $\binom{R_r-1}{j}$ equal valued mediators competing for the available query slots, and the final ratio calculates the fraction of those that might contain m.

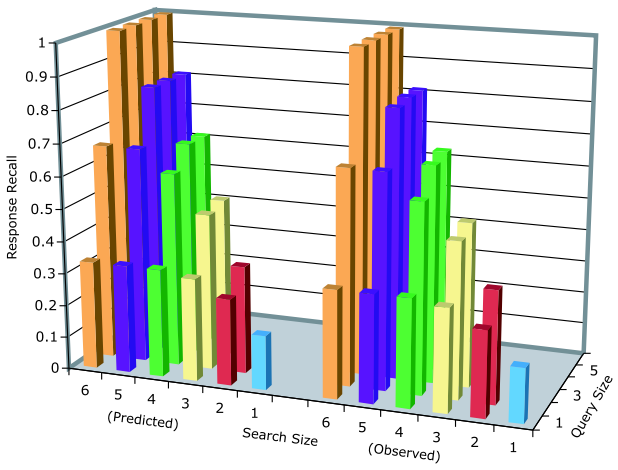


Figure 8: A comparison of the predicted and empirical response recall values as the search and query sizes are varied.

These equations are used to determine the final *query_rate* for a particular mediator, and its *query_probability* which is used elsewhere in the model.

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 (described in more detail in (13)) 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 with no appropriate data sources served as distractions. The *perceived_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. A graph comparing the values predicted by the ODML model and the empirical results are shown in Figure 8. As expected, when the search size 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. This shows that the predictions were quite accurate in most cases, with an average of 0.9% error over all cases. Experiments not shown here with other designs produced similar results.

The second characteristic that we will describe, the response time of the IR system, is the amount of time that elapses between a user query and the system's response. This 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 of the organization. Several characteristics affect this value. For example, each communication event incurs some message transit latency. The query processing by the databases, and the aggregation performed by both the aggregator and mediator will take some variable amount of time. Because queries are reflected to all subordinates, the latter two entities must also wait for slowest of their information sources before they can themselves respond. Finally, because multiple queries can exist simultaneously in the network, additional delays at individual agents can be incurred when a query must wait in a queue for existing processing to complete. The ODML model draws upon existing techniques from probability theory, queuing theory and order statistics to capture these aspects of the system. In particular, the model predicts an entire probability density function for the response time at each level in the organization, not just a single mean statistic. As will be shown below, this detailed represented is needed to correctly predict the performance of managers that are dependent on subordinates that act as queuing systems.

The rate at which an individual mediator will receive user queries it its *arrival_rate*. Responses will be returned back at this same rate, on average. More concretely, the existing model assumes that queries have a Poisson arrival distribution and mean rate *arrival_rate*. After the query makes its way down through any aggregators, the leaf databases will receive the query, also at rate *arrival_rate*. Each database has a *service_rate*, defining how quickly it can process queries. At any given time there may be previously received queries already being processed or waiting at the database. We assume FIFO processing, so the amount of time any new query must wait will depend in part on these existing queries.

Queuing theory (18; 29) can be used to analyze how long the wait will be, by modeling a database service as a *M/M/1* queue with Poisson task arrival rate and service rate. The pdf $f_M(x,\lambda)$ and cdf $F_M(x,\lambda)$ of the *M/M/1* queue's waiting time distribution are given below, where $x \geq 0$ and $\lambda = service\_rate - arrival\_rate$.

$$f_M(x,\lambda) = \lambda e^{-\lambda x} \quad (6)$$

$$F_M(x,\lambda) = 1 - e^{-\lambda x}, \quad (7)$$

The model maintains this information as a discrete list of sampled points, which are calculated dynamically from the two underlying functions.

The mediator waits for responses from all its information sources before progressing, so its expected service time will be dependent on the maximum service time of those below it. This value can be found by computing the maximum order statistic (the expected maximum value) of those source service times. The model makes a simplifying assumption that the source service time distributions are independent, but does not assume they are identical. The model generates the pdf $f_{(n)}$ and cdf $F_{(n)}$ distributions of this independent, not-identically distributed (inid) max order statistic using the following functions (4; 28):

$$f_{(n)}(x) = \Big[\prod_{i=1}^{n} F_i(x)\Big] \sum_{i=1}^{n} \Big(\frac{f_i(x)}{F_i(x)}\Big) \quad (8)$$

$$F_{(n)}(x) = \prod_{i=1}^{n} F_i(x), \quad (9)$$

where f_i and F_i represent the pdf and cdf of the ith sample, respectively (i.e., the service time distribution of the ith source).

The mediator itself is not simply a pass-through, but must process and aggregate the resulting data as well, introducing additional latency. Thus, the mediator can also be viewed and modeled as a *M/M/1* queue. The service rate of the mediator depends on the number of responses it receives, which depends on the number of information sources below it. The mediator's pdf and cdf can also be produced using Equations 6 and 7, with $service\_rate = response\_service\_rate/num\_sources$, and a Poisson rate of $\lambda = arrival\_rate - service\_rate$.

The model determines the total service time by combining these two activities, modeled as the sum of the times exhibited by these two random variables. 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:

$$f_C(x) = \sum_{i=0}^{d\_range/d\_step} f_s(i)\, f_l(x-i)\, d\_step \quad (10)$$

$$F_C(x) = \sum_{i=0}^{d\_range/d\_step} f_s(i) F_l(x-i)\, d\_step \quad (11)$$

where $x_n = n$ and $(0 \leq n < \frac{d\_range}{d\_step})$. d_range represents the upper bound on the sampled points, while d_step is the stride length between points. For the mediator, f_s would be the aggregate information source pdf given in Equation 8, while f_l and F_l would be the pdf and cdf of the waiting time for the local *M/M/1* queuing process.

Note that Equations 8-11 are recursive, in that they rely upon both the pdf and cdf distributions of the sources below the mediator. The equations make no assumptions about the form of those distributions, so they can be used both when the information source is a single database or an arbitrarily complex aggregator hierarchy. This same assumption also allows Equations 8 and 9 to be used to compute the pdf and cdf distributions for the aggregator itself. The definition terminates in the exponential response distribution exhibited by the databases. By incorporating the cumulative overhead incurred by the message transit times of the query and result propagation process the model can now compute the expected *service_time* of the mediator, allowing prediction of the *response_time* distribution of the organization as a whole.

The model's predictions are shown in Figure 9. Each scenario measures the response time performance of a different IR organizational design, by submitting 1000 queries to it in a Poisson fashion. The organizational design of each scenario is depicted on the left, along with the predicted (solid) and empirical (dashed) response time distribution data on the right. These performance graphs show the ODML model does a good job of predicting the response time distribution of the different organizational designs. Additional trials were performed for organizations with three agents [1 Mediator, 0 Aggregators, 2 Databases], five agents [1M,0A,4D], 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 (8). R^2 was greater than 0.8 for all tested scenarios, where a value of 0.7 or above is considered good for this statistic.

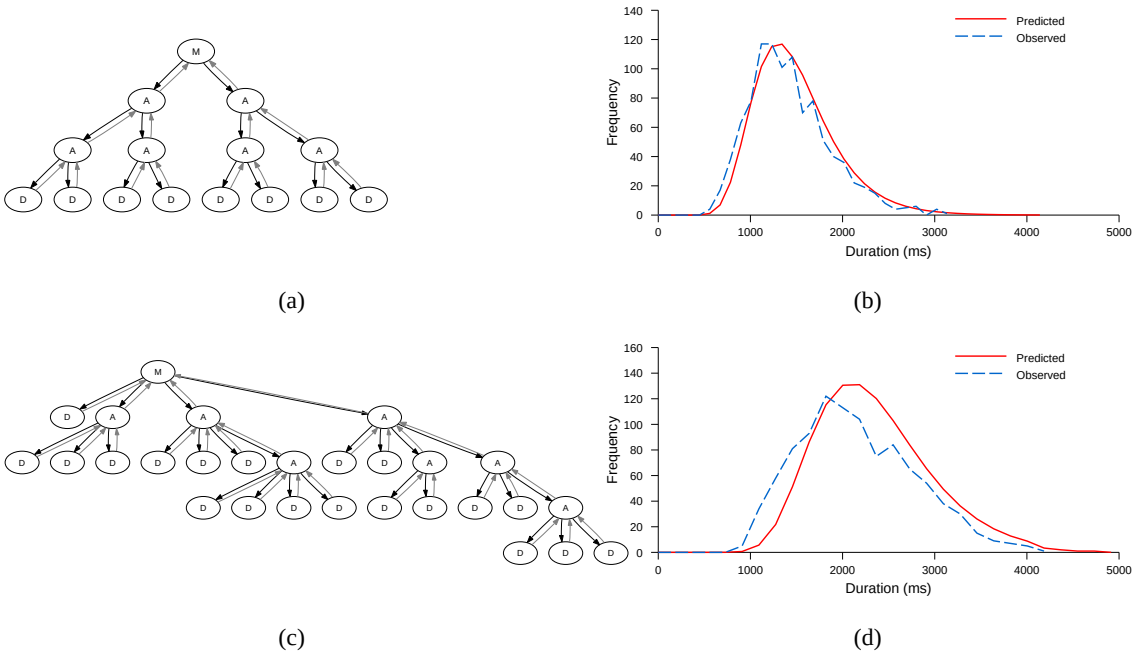


Figure 9: A comparison of the predicted and observed response time distributions in organizations with (a,b) fifteen [1M,6A,8D], and (c,d) twenty-eight [1M,7A,20D] agents. In (a,c), node M is a mediator, A are aggregators, and D are databases.

3.2.3 Coalitions

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. Unlike the previous two examples, hierarchical relationships generally do not exist within the groups. An example ODML template is seen in Figure 10a, which shows that both the number of coalitions, and the number and type of participants in each coalition can vary. A sample coalition instance produced from this template is shown in Figure 10b. Note that although the organizational instance is structured as a tree, the organization itself is not hierarchical. As with the previous two examples, the *organization* node is used as a convenient place to encode global information about the organization. The individual *coalition* groups act as independent peers.

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 could 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 (33). This can be modeled by setting *max_coalitions* to some value greater than one.

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

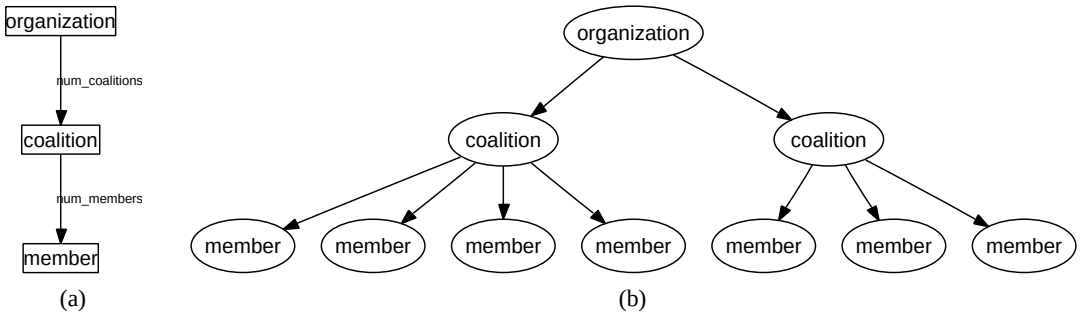


Figure 10: An ODML coalition a) template and b) example instance.

of this, and can be modeled with a simple summation in the *coalition* node:

$$strength = forallsum(m, members, m.mass)$$

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:

$$\begin{aligned} skill\_a &= forallsum(m, members, m.skill\_a) \\ skill\_b &= forallsum(m, members, m.skill\_b) \\ strength\_a &= 1/(1+e^{\wedge} - ((skill\_a - 0.5)/0.1) \\ strength\_b &= 1/(1+e^{\wedge} - ((skill\_b - 8)/0.5) \\ strength &= strength\_a * strength\_b \end{aligned}$$

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.

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.

In a self-interested situation, it is not only the strength 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:

$$reward = c.reward/c.num_members$$

In this case, *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*):

$$reward = c.reward * strength/c.strength$$

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

3.3 Organizational Utility

The features contained in these models provide some insight into the space of possible characteristics that can

be relevant when deciding upon an appropriate organizational design. A search process for the best organizational design must be able to reason about the relative advantage or disadvantage each of these characteristics has relative to the environment and larger objectives. This is typically accomplished by summarizing those features into a single numeric utility value, which can then be used to compare and rank competing designs along a common scale.

The utility of a design can be considered to be an organizational characteristic like any other. As such, it can be embedded in the ODML model as a constant field called *utility*, and defined using whatever expression is most appropriate. This allows it to be based on many different, potentially interacting and domain-specific characteristics, which is consistent with the complexity that exists in determining the utility of a design in the real world.

In the simple coalition model, for example, *utility* might be defined as the sum of the strengths of the individual coalitions minus the sum of the costs, or in terms of the number of tasks that can be completed with the skill sets that are represented with a similar cost penalty. In the self-interested case, the global utility might also be a function of the individuals' utilities, or hard constraints could be added to each agent to ensure that all participants have some minimal utility.

As another example, Figure 1 shows that *utility* in the DSN domain is defined as negative *average_rms*, thus utility increases as the expected RMS decreases. This metric is appropriate for this domain because it captures what is arguably the most important measure of quality of a tracking system, which is the error of its track estimates. As described earlier, *average_rms* is defined directly or indirectly in terms of nearly all other characteristics of this organization. By association, the utility then depends on how large the sectors are, how well the tracking load is distributed, how roles are assigned, and a host of other features.

Sometimes the features that performance is dependent on are not all precisely known at the time the model is created. This is generally occurs when features are beyond the control of the designer, because they are part of the environment. It also can occur when the model is being used to predict the performance of an open organization, and the participants are not known a priori. ODML supports the use of probabilistic distributions in these cases, which allow the designer to encode a range of possibilities, along with the relative chance that each possibility has of occurring. For example, the *num_targets* constant in DSN model in Figure 1 is defined with a distribution, because we do not know *a priori* how many targets will exist in the environment. ODML provides a Monte-Carlo evaluation technique when the utility value depends on such uncertain information and that information cannot be mathematically evaluated directly. In this case, utility is determined through a series of repeated sampling and evaluation trials, the results of which are combined to produce a suitably representative value.

Utility in the IR domain is defined in terms of the design's expected response recall and response time. 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:

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

The normalization terms cause this formulation to generally favor quality over speed, and instances with equal recall will be differentiated by their response time.

As with the DSN formulation, the IR utility also depends (recursively) on many different characteristics of the design as well as the environment the organization is expected to operate in. For example, one would expect the performance of a particular IR design to depend on the rate at which queries arrive in the system. For example, a design that is sufficiently robust to handle high query rates may be less efficient when the rate is low.

Figure 11 demonstrates that this behavior is manifested in the IR ODML model, by showing the predicted utility for several different designs with a range of query rates. This figure shows all eighteen possible organizations that are possible in a six database environment with a maximum height of three and a minimum of two subordinates per node. Optimal utilities for each rate are shown in bold. Organizations have zero utility at a given query rate when the query arrival rate exceeds the organization's service rate, resulting in an infinite length queue.

In this case, 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.

As the query rate increases, first organization 8, then number 9 and finally number 11 become optimal, as the highly-connected mediator in organization 1 becomes an increasing bottleneck. The benefit the multi-mediator designs offer is increased ability to handle high work loads.

Organization	Query Rate 0.1	0.25	0.5	1	2	3	4	5	6	7
1.	**725**	**723**	**719**	**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	**695**	**680**	**657**	612	482	0
9.	705	704	702	699	690	677	**657**	**619**	**517**	0
10.	710	709	707	702	689	665	604	0	0	0
11.	379	379	379	378	378	378	377	376	375	**374**
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 11: 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.

Where no single-mediator organization can handle more than six queries per second, all eight multi-mediator designs can obtain utility with at least seven queries per second. This is because the smaller search size reduces the query rate any individual mediator sees. 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 (albeit with lower recall).

These results show the spectrum of tradeoffs that can be made in this particular design. In a broader sense they also demonstrate why explicit organizational design is useful, by providing a concrete example of how design impacts performance. From the perspective of this paper, they show that the ODML model was able to capture the underlying interactions between the various elements of the organizational design and the environment.

The ability to distill a set of organizational decisions down to a single utility value is a key component in any automated organizational design process, because it permits one to directly compare competing designs. Section 4 will show how this capability is leveraged as part of a larger search process for the most appropriate organizational design.

3.4 Model Fidelity Versus Complexity

Based on our experiences, first designing and then evaluating the DSN and IR models against their real counterparts, we believe that ODML does a good job of satisfying our initial objective of providing a flexible-but-quantitative toolkit for organizational design. The DSN, IR and coalition systems are substantially different, using different organizational styles with different objectives and different sets of relevant characteristics. As demonstrated by the results presented in this section, ODML is effective at capturing these quantitative characteristics, allowing one to quickly determine the utility of each design. This is in contrast to the majority of the related systems described in Section 2, where it is not possible to explicitly tie organizational decisions to metrics such as tracking error, response recall or a coalition's contribution to utility.

A significant drawback to the approach we propose is the complexity of the modeling process itself. The DSN and IR models were created and incrementally refined, adding new features as the mechanics of the corresponding systems were better understood. In some domains, sophisticated techniques may be needed to accurately capture relevant phenomena. We believe, however, that if such sophistication is needed, it is needed regardless of one's choice of modeling language. The complexity is a characteristic of the domain in question, and tools that lack the ability to express this complexity can only approximate the behavior of the system. This can ultimately affect the quality of the decisions that are made based on those tools.

One way to cope with design complexity is to recognize common elements that 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 taking advantage 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 12. 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 3a) 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.

Another facet of the same complexity problem is knowing when a model is complex enough that additional refinements are not necessary. Ideally, one would avoid modeling minutia that may not needed to produce a satisfactory answer to the organizational design problem. If a domain is well-understood and has recognizably dominant interactions, then capturing those features will generally be sufficient. The level of needed complexity also depends on the context in which the system will be deployed. If computational resources are abundant, for example, it may not be necessary to model agent computational load.

Consider Figure 13, which shows a range of modeling strategies, from more abstract to more detailed. The DSN model presented in Section 3.2.1 exists at the far left of this figure. Additional modifications presented in (13) add notions of space and time, representing points further to the right. In model from Section 3.2.1, we chose to ignore the fact that, for example, sensors on the edge of the area will likely be used less than their counterparts in the interior. Instead, usage is captured with an approximation that is uniform over all sensors. The additional detail needed to capture geographic differences might be warranted if one of the organizational decisions that needed to be made included where to deploy "robust" versus "normal" sensors.

When interactions are more complex or bounds are tight, one can determine if additional details are needed by comparing predictions produced by organizational instances against known phenomena, simulation results, or even intuition. Like any mathematical model or simulation, ODML templates should be vetted through analysis or empirical comparison to determine their accuracy. As mentioned above, however, we believe that model complexity is ultimately driven by the application domain. The ability to choose the level of detail that is most appropriate sets ODML apart from most existing organizational design frameworks. We will return to this concept in Section 4.2.4.

4 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 tem-

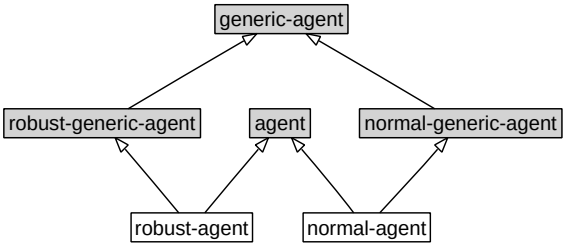


Figure 12: Reusing common agent models in new domains.

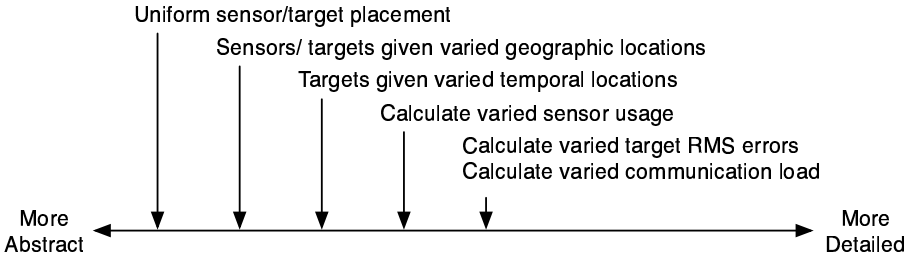


Figure 13: A range of modeling possibilities, each with different levels of abstraction.

plate and selecting an appropriate instance. This organization space is defined by decision points that exist in the template. These decision points are 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 *num_sensors* variable exists in the *sector* node. This variable can take on one of several discrete numeric values, as shown in Figure 1. This controls the size of the has-a relationship *sector* has with *sensor*. As shown earlier, each of these choices results in an organization with different characteristics. In the IR domain, a similar has-a size variable called *topic_mediators* exists that determines how many mediators will exist in the organization. It can be assigned different values, which will also 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*, and assume that both *normal_agent* and *robust_agent* have an *is-a* relationship with *agent*. 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. In that model, both *aggregator* and *database* are possible instances of *source*. The *mediator-source* has-a relationship therefore represents a decision point, because two different types of source may be used to satisfy the relationship. 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 by dividing scarce resources, as well as the performance of any other entities 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.

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 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.

In either case, the search progresses by determining the validity of each considered design, and using the calculated *utility* value from Section 3.3 to compare and rank it against competing designs. The valid design with the greatest utility is considered optimal. 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 necessity for such a systematic approach to design is derived from at least three parts. The first, as shown in Section 3.3, is that the particular organization that is employed can have a significant effect on a range of important runtime characteristics. Section 3.2 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 11 came up with 16 alternatives. It was only after the more methodical and computational search techniques 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.

4.1 Design Complexity

Although superficially straightforward, the combinatorics of the search space conspire to complicate the search process. For example, 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*, making the search progressively more difficult as the structure is incrementally produced. In fact, the problem of finding even a valid organization, not necessarily the more desirable optimal or most appropriate one, is shown below to be NEXP-Complete. This is consistent with complexity results presented in related work by Nair and Tambe (24), who analyzed the complexity of the role assignment and execution problem. We 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.

In these proofs, we 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 recur-

sion, adding in placeholder nodes as needed to represent the individual recursion levels.

Lemma 4.1. *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, $\{N_1, \ldots, N_n\}$. 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. Because of this, there will 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 $\{N_1 \ldots N_k\}$ 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}:

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

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

Lemma 4.2. *ODML-SAT is in NEXP.*

Proof. Assume we have an arbitrary ODML structure 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 the largest organization derivable from a non-recursive ODML structure 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.

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

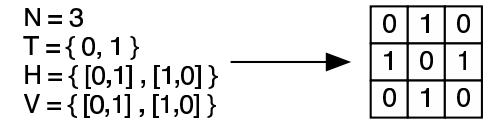


Figure 14: A sample TILING problem and consistent solution.

be nondeterministically discovered and validated in exponential time, it is in the NEXP complexity class. □

To demonstrate that ODML-SAT is NEXP-Hard, we will reduce from the TILING problem, as defined in (25; 1). A TILING problem consists of a set of tile types $T = \{t_0, \ldots, t_k\}$, 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 $f : \{0, \ldots, N-1\} \times \{0, \ldots, N-1\} \rightarrow T$. 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 14.

Lemma 4.3. *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 15. 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 row*s of *N tile*s 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 the *N row* nodes of *N tile*s 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 4.4. *ODML-SAT is NEXP-Complete*

Proof. By Lemmas 4.2 and 4.3, and because TILING is itself NEXP-Complete (25), ODML-SAT is NEXP-Complete. □

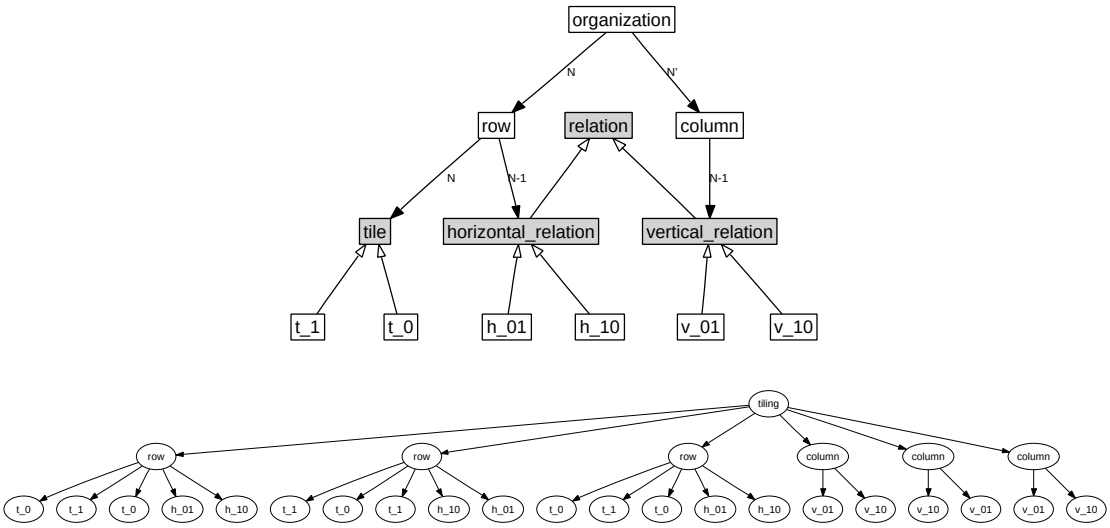


Figure 15: (top) An example ODML template used to reduce a TILING problem. (bottom) A valid organizational instance created from that template.

4.2 Searching for Organizations

These complexity results suggest that an optimal but combinatorially feasible search technique that works for all search spaces is unlikely to exist. 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. We 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 techniques will eliminate, avoid or more efficiently find 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 remain after applying the algorithmic technique. Additional search techniques not described here can be found in (13), including how to avoid redundant search, the use of general mathematical solvers, the use of cached values and the inclusion of homogeneity in the model to shrink the search space.

4.2.1 Exploiting Hard Constraints

One way of avoiding unnecessary branches in the search tree is to exploit the existence of hard constraints in the model. This is accomplished by determining the numeric trend of a constrained value, and bounding 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.

For example, in the DSN 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 bound to a *sensor*. The one sensor per agent restriction is modeled by a hard constraint in the *agent* node, which specifies that $sensors\_controlled \leq 1$, as shown in Figure 1. Because *sensors_controlled* is only affected by the *sensor* modifier, it can only increase monotonically. Therefore, if the constraint is ever violated it will remain unsatisfied regardless of what subsequent decisions are made.

Because a valid organizational instance cannot contain unsatisfied constraints, if a constraint has become unsatisfied during the course of an organizational search it may 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. A strategy that blindly backtracks when an unsatisfied constraint is observed is incorrect, since valid organizations may potentially be missed in either case.

Because a constrained value may change as a result of decisions made elsewhere in the model, for example when a new modifier is instantiated or a choice is made for a variable the value is dependent on, one cannot determine the trend of a characteristic myopically. However, it is possible to make this determination through a wider inspection of the relationships defined in the organizational template. 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. Because this technique relies on features that can be derived directly from the underlying model, it is domain-independent.

The algorithm used to determine satisfiability is outlined in Figure 16. The 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 first 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 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

```
is_satisfiable(constraint c)
    if (c.satisfied) return true
    t_v ← find_trend(c.LHS)
    t_e ← find_trend(c.RHS)
    if (t_v = unknown ∨ t_e = unknown) return true
    if (t_v = constant ∧ t_e = constant) return false
    if (t_v = increasing ∧
        (t_e = constant ∨ t_e = decreasing)) return false
    if (t_v = decreasing ∧
        (t_e = constant ∨ t_e = increasing)) return false
    return true

find_trend(expression e)
    D ← e.dependencies
    M ← e.modifiers
    t ← constant
    for (f ∈ D ∪ M)
        t_f ← find_trend(f)
        d_f ← ∂e/∂f
        if (d_f.dependencies ⊆ { f } ∧ d_f is linear)
            t_f ← ∇ d_f (i.e., wrt f and t_f, increasing,
                  decreasing or constant)
        else t_f ← unknown
        if (t_f = increasing ∨ t_f = decreasing)
            if (t = constant) t ← t_f
            else if (t_f ≠ t) t ← unknown
        else if (t_f ≠ constant) t ← t_f
    return t
```

Figure 16: Pseudocode for the constraint satisfiability and trend estimation procedures.

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.

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

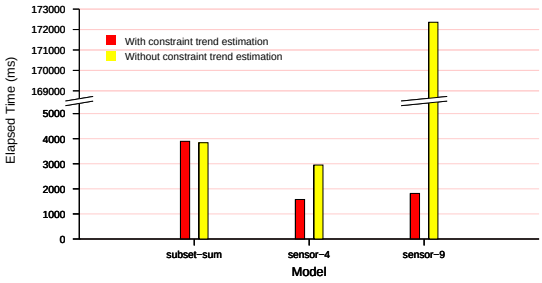


Figure 17: A comparison of the search time differences with and without the constraint estimation algorithm across three different models.

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. If *is_satisfiable* 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 impossible to determine satisfiability using this particular analysis technique.

The potential benefit of using this technique has been demonstrated through a series of experiments, the results of which are shown in Figure 17. Three different models were contrasted in a pair of trials, one that bound the search using the constraint trend estimate, and one that did not. The two DSN models contain a constrained monotonic trend in *sensors_controlled*, as described above. The subset-sum model was included as an example because it contains a constrained non-monotonic trend. 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 its constrained value was non-monotonic. 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 technique we will discuss exploits the idea of equivalence classes to reduce the search space. To do this, we must formalize the various ways that a has-a relation may be satisfied. Recall from Section 3 that $\mathcal{N}$ is the set of node templates present in an ODML model, $h.t$ 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:

$$\mathcal{N}_t = \bigcup_{N \in \mathcal{N}} N | (N.t = t) \vee (\exists_{i \in N.I} i \in \mathcal{N}_t) \quad (12)$$

Let A be the set of nodes 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:

$$D_h = \bigcup_{N \in \mathcal{N}_{h.t}} \{a'_{N.t}\} \cup \{a \in A | a.t = N.t\} \quad (13)$$

$a'_{N.t}$ 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.

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 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.

The potential benefits of using this technique are shown in Figure 18, which compares the number of organizational alternatives (both valid and invalid) that are considered with and without equivalence classes across four DSN 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

Figure 18: 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.

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. In particular, if the space of candidate decisions is small or highly varied, then the number of equivalence classes may not be much smaller, and there may not be much observed benefit. Conversely, larger, more regular organizations like the example shown here can see significant reductions in search time. This is similar to the hard constraint exploitation technique from the previous section, which also relied on the existence of particular features in the model, and is a drawback to both approaches. The parallel search technique in the following section does not suffer from this problem, but as will be seen it also does not achieve the same level of benefit.

4.2.3 Parallel Search

A different approach to the search problem is to decompose the space and distribute it among a group of processors. This distribution is facilitated by the naturally decomposable organizational space. Recall that 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 significantly reduced by performing multiple searches in parallel.

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, efficiently creating such a characterization for an ODML template in the general case is not a simple process. 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, we have employed an alternative division technique that partitions the space dynamically as the search progresses, similar to that presented in (27). 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 an 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

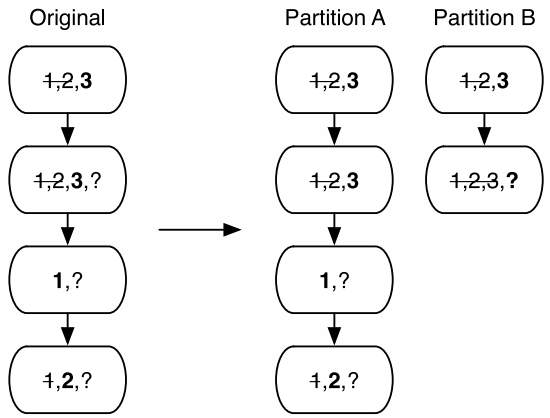


Figure 19: The partitioning of a local search tree. Strikeouts indicate visited choices, bolded are the current choice, while a question mark indicates additional unknown choices remain.

requester then stops its querying and begins searching the new space. This continues until the search is completed.

The performance of the distributed search using different numbers of processing agents is shown in Figure 20. These were produced from a series of optimization searches on a template with an organizational space containing approximately 12 million alternative designs.

Figure 20a 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 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 20b, which indicates a roughly linear increase in messaging as the number of processing agents increases.

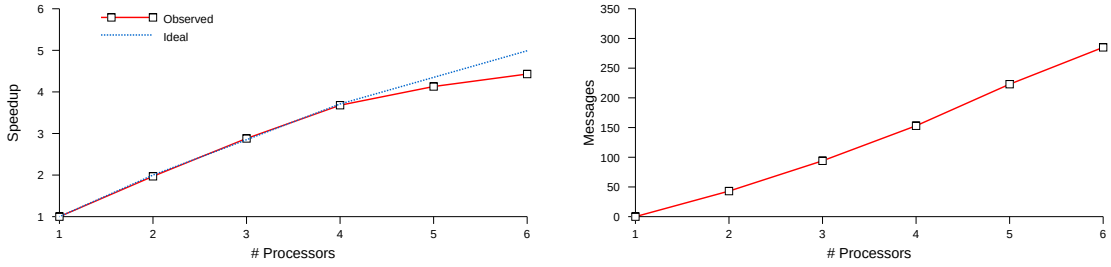
The benefit that this approach offers over those in Sections 4.2.1 and 4.2.2 is that it works on all models and requires no modifications to a model to be used. The drawback is that it clearly requires additional physical resources, and the amount of achievable speedup is lower. For example, a roughly linear improvement was observed with the distributed approach, while the search techniques described earlier produced results that were orders of magnitude better in some cases. These techniques are not mutually exclusive, and in practice the earlier search techniques are used without modification by the individual participants in the distributed search to further improve performance.

4.2.4 Model Abstraction

Improving the techniques used by the search process is not the only way that the efficiency of automated design can be improved. A different approach is to use abstraction to reduce elements of the structure to the simplest form that still produces the desired level of accuracy. 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, resulting in a smaller organizational space and a more tractable search problem. 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 from most other existing representations, which typically require a complete structure down to the agent level. Because the model itself is changed with this technique, it can potentially lead to an undesirable loss of expressivity, 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 is to truncate the model at some point higher than the level actually used by the running system. For example, one could choose to model only the most pertinent aspects of an agent's decision making process. A more concrete example of this technique is to not model down to the level of assigning roles to individual entities or agents, as shown in Figure 21. Organizations derived from a truncated template will specify what roles exist, and where they are located in the orga-



(a) (b)

Figure 20: Characteristics of the distributed search using different numbers of processors. The speedup factor is shown in (a), and the number of messages required in (b).

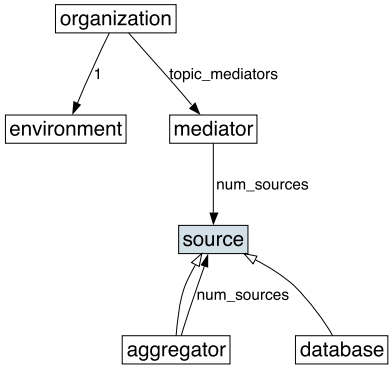


Figure 21: An information retrieval template derived from Figure 7a that incorporates abstraction by eliminating the assignment of roles to distinct agents.

nizational 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 (30; 24; 3).

This technique is analogous to those presented by Durfee in (10), which reduced complexity by using team-level abstraction to leave specific agent assignments unbound during coordination. 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 auxiliary 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 (37). 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 has 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.

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

Template	Decisions	Valid Organizations	Utility
Baseline	6,210,780,885	12341	692.86
Abstract	59,940	12	692.86

Table 2: Search results from two small-scale IR templates. Utility is given for the optimal found organization.

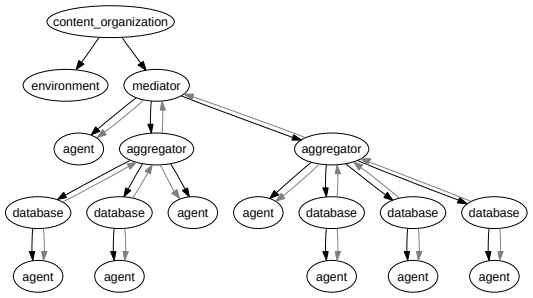


Figure 22: The optimal organization found by the baseline template for the small-scale scenario.

out. Some approaches will clearly be better than others in terms of space complexity and achievable utility, and these must be weighed against the value of information lost through the modeling changes.

To demonstrate the effectiveness of this technique, two small-scale variants of the IR template were evaluated. The design allowed up to five databases, up to two mediators, and the aggregators could have two, three or four sources. The number of agents was unbounded. The source node types are the same hierarchies discussed in Section 3.2.2, 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). The baseline template is shown in Figure 7a. The second template employs abstraction, by not assigning a particular agent to each role as shown in figure 21. Alternatively, one could also view this model as creating a new agent for each role.

The test results are given in Table 2, which shows the number of decisions made during the entire search, the number of valid organizations that were found, and the utility of the optimal found structure. These show a dramatically smaller search space when abstraction is used, which reduced the number of valid organizations that had to be evaluated by several orders of magnitude. The reduction in decisions that were made was even greater. This demonstrates that 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 22. In this case the optimal organization found in both searches was the same, because there was no limit on the number of agents and no cost associated with each agent. In general one cannot assume that the optimal organization will be retained in the modified model, and care must be taken to ensure that the space of possibilities is not adversely affected.

5 Conclusions

As was shown in Section 2, many different organizational representation languages have been created by researchers in the past. The ODML language defined in Section 3 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. We believe this approach leads to an increased diversity of representable situations as well as an increased level of predictive detail. 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.

Section 3 showed that ODML can be used to capture many different organizational features across different domains. These include a complete model of a real-world distributed sensor network architecture, 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 distributed information retrieval network, 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. A

simpler model of a coalition was also discussed, showing how organizations lacking any semblance of a hierarchy may be represented, and providing some further insights into rewards, costs and self-interested agents.

The result of these modeling efforts 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, they show 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.

In addition to their predictive qualities, the models templates also describe the range of possible organizational instances, using a pair of relatively simple decision types. This allows an ODML model to characterize the space of design alternatives, which can then be systematically searched to enact the automated organizational design process. Section 4 proved the complexity of this process, which lead to the exploration of several approaches that have been employed to cope with that complexity. Techniques that exploited hard constraints and notions of equivalence used particular model features to achieve significant reductions in search time. A distributed approach to the search that works with all models exhibited less impressive, but still worthwhile improvements. These are currently unified in a single implementation that attempts to use each technique where applicable, and is conservative where they are not. Finally, we also showed that through changes to the model itself one can vary the level of abstraction and detail, directly affecting the size of the organizational space. 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 these techniques remove the fundamental complexity issues that arise as the general problem scales (notice that, for example, the trends in Figures 18 that use the equivalence technique 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 not an exhaustive study of such general approaches, and we believe that in many cases a suitably crafted domain-specific approach can yield additional benefits.

A notable drawback to the detailed approach taken by ODML is the level of effort needed to build the models themselves, and the complexity of the resulting artifact. 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. We contend that the majority of this complexity is a product of the domains themselves; ODML just provides a means to express it. Existing organizational frameworks generally lack the ability to represent such concepts, or they are specialized to capture a subset of domain-independent characteristics.

Searching through the organizational space created by these models is also difficult, particularly as the number of decisions increases. This underscores the need to develop good heuristics that can navigate the search space of common-case organizational designs. A benefit of our approach is that it enables an efficient inner loop for this search, by using the embedded web of equations to quickly evaluate candidate designs. Evaluating a particular ODML instance is orders of magnitude faster than running the trials needed to analyze a working system in simulation or with a prototype. This ability means that more designs can be searched in less time, which can correspondingly increase the quality of the final result. If sufficiently accurate modeling techniques are not known, it is also possible to exploit the strengths of both approaches, by using simulation trials to analyze a set of solutions previously culled by a model-based approach.

5.1 Future Directions

There are a number of areas where this work may be extended. Possibly the most ambitious would be its application to dynamic reorganization. Although the models described in Section 3.2 incorporate notions of change, the resulting design is still static in the sense that that the underlying principles that produced it are not reevaluated. In the face of a changing environment, any static organization is vulnerable to contextual changes that can render it inefficient or ineffective. We 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 (14). Appropriate search strategies must then be developed to find organizational appropriate solutions in a timely manner. We feel ODML's abil-

ity 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.

Less ambitious, but equally useful would be the development of additional techniques to search the space defined by an ODML model. The existing search techniques described in Section 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. If 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 to this search 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. As mentioned earlier, the possibility of using a domain or model-specific search process that relies on environmental or design assumptions also seems potentially fruitful.

Finally, we have also considered how ODML can be applied to fundamentally different situations than the top-down, cooperative design process described here. For example, an ODML model could be used by a particular agent as a tool to help predict the drawbacks and benefits of joining an existing competitive organization, such as a marketplace. ODML might also be used to create an appropriate model of an existing organization. The organizational space defined by the model can then be analyzed in an effort to determine how good that organization's design is relative to what is possible. Similar techniques could be used to determine vulnerabilities of organizations, by isolating weak or critical dependencies in the underlying equation graph. These and other potential benefits motivate further research into representations like ODML that are capable of capturing the quantitative details of organizations.

References

[1] 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.

[2] Daniel D. Corkill and Susan E. Lander. Diversity in Agent Organizations. *Object Magazine*, 8(4):41–47, May 1998.

[3] 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.

[4] Herbert Aron David. *Order Statistics, 2nd Ed.* Wiley, 1981.

[5] K. Decker and V. Lesser. An Approach to Analyzing the Need for Meta-Level Communication. *International Joint Conference on Artificial Intelligence*, 1, January 1993.

[6] 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.

[7] 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.

[8] Jay L. Devore. *Probability and Statistics for Engineering and the Sciences*. Wadsworth, Inc., Belmont, CA, 1995.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] Bryan Horling. *Quantitative Organizational Modeling and Design for Multi-Agent Systems*. PhD thesis, University of Massachusetts at Amherst, February 2006.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] L. Kleinrock. *Queueing Systems. Volume I: Theory*. John Wiley & Sons, New York, 1975.

[19] 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.

[20] V. Lesser, C. Ortiz, and M. Tambe, editors. *Distributed Sensor Networks: A Multiagent Perspective (Edited book)*, volume 9. Kluwer Academic Publishers, May 2003.

[21] 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.

[22] Thomas W. Malone and Stephen A. Smith. Modeling the performance of organizational structures. *Operations Research*, 36(3):421–436, 1988.

[23] 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.

[24] 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.

[25] Christos H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.

[26] 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.

[27] V. Nageshwara Rao and Vipin Kumar. Parallel depth first search, part i: Implementation. *International Journal of Parallel Programming*, 16(6):479–499, 1987.

[28] Reiss, R.D. *Approximate Distributions of Order Statistics*. Springer-Verlag, New York, NY, 1989.

[29] S. Ross. *Introduction to Probability Models*. Academic Press, Boston, MA, 5th edition, 1993.

[30] 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.

[31] 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.

[32] Sandip Sen and Edmund H. Durfee. A formal study of distributed meeting scheduling. *Group Decision and Negotiation*, 1998.

[33] Onn Shehory and Sarit Kraus. Methods for task allocation via agent coalition formation. *Artificial Intelligence*, 101(1–2):165–200, 1998.

[34] 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.

[35] 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.

[36] 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.

[37] 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 Multi-Agent Systems (AAMAS 2003)*, pages 867–874, Melbourne, AUS, July 2003. ACM Press.

[38] Young-pa So and Edmund H. Durfee. Designing tree-structured organizations for computational agents. *Computational and Mathematical Organization Theory*, 2(3):219–246, 1996.

[39] Miland Tambe. Towards flexible teamwork. *Journal of Artificial Intelligence Research*, 7:83–124, 1997.

[40] Milind Tambe, Jafar Adibi, Y. Alonaizon, 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.

[41] Thomas Wagner, John Phelps, Valerie Guralnik, and Ryan VanRiper. Coordinators: Coordination managers for first responders. In *AAMAS '04: Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 1140–1147, Washington, DC, USA, 2004. IEEE Computer Society.

[42] 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.

A Information Retrieval ODML Model

This appendix contains the complete ODML listing for the information retrieval domain model described in Section 3.2.2. The raw textual model given below contains additional structure not shown in Figure 7a. 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 earlier discussion.

As with most 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 probability distributions described in Section 3.2.2. 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 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"?>
<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="&gt;=">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>

    <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 percieved 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="&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>
<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(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>

    <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="&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="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>
```