Using ODML to Model Multi-Agent Organizations *

Bryan Horling and Victor Lesser
University of Massachusetts
Amherst, MA 01003-9264
{bhorling,lesser}@cs.umass.edu

Abstract

In this paper, we introduce a domain-independent organizational design representation able to model and predict the quantitative performance characteristics of agent organizations. This representation, capable of capturing a wide range of multi-agent behaviors, can support the selection of an appropriate design given a particular operational context. We demonstrate the capabilities and efficacy of this language by comparing a range of predicted metrics to empirical results from a real-world system.

1. Introduction

Any real-world system must be tailored to the environment in which it exists if it is to make effective use of the resources and flexibility available to it. In this paper, we explore the possibility of such tailoring through the system's *organizational design*. The notion of an organization is used in many different fields, and generally refers to how members of a society act and relate with one another. In multi-agent systems the organizational design can specify the types of agents, what roles they take on, and how they act and interact. This additional structure becomes increasingly important as the system scales. Imagine how difficult it would be for a large group of humans to function if individuals lacked job descriptions and long-term 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 tracking system. The system would consist of an array of agent-controlled sensors deployed to track mobile targets in an environment. Assume that the sensor nodes must collaborate in some way to be successful. Given this, a designer must determine a way to structure the agents' behaviors so that tracking may be accomplished. One strategy would 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 specification constitutes a rudimentary organizational design. It indicates 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 has an omniscient view of the entire network that can be used to find the best assignment of sensing tasks. However, under real world conditions, where resources are 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 different agents, to more evenly balance the communication and computational loads. We might also create an information dissemination hierarchy among the agents that summarizes and propagates measurement data, to use the available bandwidth more efficiently. However, distributing the role can lead to conflicts and lower utility, because no single agent necessarily has the appropriate local context to make the right decision. Similarly, a hierarchical summarization process might introduce unwanted latency and imprecision.

Implicit in this example is the idea that different organizations will affect performance in different ways, either helping or hindering depending on the situation. Intuitively, changing the manner in which agents interact or the pattern that interactions take on can change behavior at both global and local levels. 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

* 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.

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.

It is our belief that understanding the fundamental causes of these characteristics and developing accurate models of their effects are both critical to selecting an appropriate design. To address this need, we will present a new, domain-independent language designed to capture organizational information in a single predictive structure. This Organizational Design Modeling Language (ODML) incorporates quantitative information in the form of mathematical expressions that are used to predict organizational characteristics. Succinct representations of the range of organizational possibilities can be defined with ODML, using local knowledge to automatically predict the behavior of the global system. This representation addresses the needs outlined above by creating a computational model that is able to uncover the subtle effects of interactions.

Section 2 will define the ODML language. Section 3 will discuss how ODML models are created, while Section 4 will validate the model's accuracy. Our ultimate objective, to use such models to automatically select the appropriate organizational design for a particular operational context, is outlined in Section 5.

2. Representing Organizations

An organizational model, as we envision it, serves in several different capacities. At design time, it should enable the evaluation of not just a single organizational instance, but an entire family of organizational possibilities. At runtime, it should accurately describe the current organization. In both cases, the model must be sufficiently concrete such that one can predict the organization's effectiveness, and rank alternatives according to specified criteria. Below, we enumerate the capabilities and characteristics the modeling language should possess to satisfy these requirements:

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

Many different organizational representation schemes have been developed by researchers [1, 9, 2, 3, 8, 6, 10]. Nearly all these representations can satisfy the first two points, but none are able to incorporate quantitative knowledge in such as way that concrete predictions can be made directly from the model itself. In this section we describe a new formalism called ODML that explores how such information can be modeled and used.

Most existing representations fall into one of two categories: either they represent a wide range of organizational characteristics abstractly, or they can capture a smaller set of characteristics concretely. The former are usually good at representing what exists 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.

For example, OMNI [2] and $\mathcal{M}$OISE$^+$[6] can each capture a greater variety of organizational concepts than ODML, but do so in a largely qualitative way. Conversely, both SADDE [9] and MIT's Process Handbook [8] can incorporate arbitrary quantitative information, but neither couples this information with the organizational structure in a way that enables one to deduce how the design characteristics affect one another. The representation created by Sims [10] does incorporate quantitative information, but lacks the innate ability to compare alternatives based solely on this information. Ultimately, each representation has its strengths, and ODML's goal is not to supplant these works – but to demonstrate another approach that makes different tradeoffs.

The formal definition of an ODML *template* specification $\mathcal{O}$ is given below. Section 3 will give examples of how these features are used in practice.

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

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 managers, relationships, agents and the environment, among other things. Each node N_i contains a number of elements, defined below:

t The node's $type$. This label must be unique within the set of nodes that make up the organization.
$N.t = \langle symbol \rangle$
$\forall N, M \in \mathcal{N},\ N.t = M.t \Leftrightarrow N = M$

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

I The set of node types that this node has an *is-a* relation with using conventional object-oriented inheritance semantics. If we assume that a node's $I = \{a, b\}$, an instance of the node will also be an instance of a and b, possessing the characteristics of all three node types. Is-a relationships cannot be cyclic, i.e., N cannot have itself as a decedent.
$N.I = \{\langle type \rangle, \dots\}$
$\forall i \in N.I, N \neq i \wedge N \notin i.I \wedge ...$

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 instances of both a and b. It is through this type of relationship that the primary organizational decomposition is formed. The magnitude specifies the number of instances connected by the relationship.
$N.H = \{\langle symbol, type, magnitude \rangle, \dots]$
$magnitude = \langle symbol \rangle$

C A set of $constants$ that represent quantified characteristics associated with the node. Constants may be defined with numeric constants (e.g., 42), or mathematical expressions (e.g., $x + y$).
$N.C = \{\langle symbol, expression \rangle, \dots\}$

K A set of $constraints$. Also defined with expressions, an organization is considered valid if all of its constraints are satisfied.
$N.K = \{\langle symbol, op, expression \rangle, \dots\}$
$op \in \{<, >, \leq, \geq, =, \neq\}$

M A set of $modifiers$ that can affect (e.g., mathematically change) a value contained by a node. Multiple modifiers may affect the same value. Modifiers model flows and interactions by allowing the characteristics of one node to affect those of another.
$N.M = \{\langle symbol, op, expression \rangle, \dots\}$
$op \in \{+, -, \times, \div\}$

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

$symbol$ refers to a user-defined string, similar to a conventional variable name. These typically describe or refer to a particular characteristic. $type$ is the type name of some defined node, so $\exists N \in \mathcal{N}$ such that $N.t = type$. $expression$ is an arbitrary algebraic expression, possibly referencing constants, symbols and function calls. Expressions support floating point values, lists of floating point values, and discrete probabilistic distributions. Collectively, we refer to C, K, M, V as a node's *fields*, and the quantitative state of a field as its *value*.

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

ODML *instances* are quite similar to ODML templates. The key difference is that where a template is a description of what *could be*, an instance is a description of what *is*. Where a template might specify that a *manager* role can be assigned to a single *agent* or distributed across multiple *agent* nodes, an instance would indicate that *manager_1* is distributed across *agent_5* and *agent_7*, and so on.

The formal definition of an instance is nearly identical to that given in Equation 1, so we will not repeat it here. The differences principally relate to the replacement of node types in the template with instances of those nodes in the organizational instance. Thus, the set $\mathcal{N}$ is the set of node instances, whose individual types no longer need be unique. 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.

It is the ability to use an ODML model to deduce quantitative values for node characteristics that sets it apart from other representations. The manner in which these values are determined for an instance node's characteristics is defined by the pseudocode in Figure 1. This shows how various sources of information, non-local data and interrelationships interact to create the features of a particular node. It is

get_value(*symbol s*)
  $r \leftarrow$ *null*
  if (*s* is of the form $s_1.s_2$)
    $n \leftarrow$ **get_value**(s_1)
    $r \leftarrow n$.**get_value**(s_2)
  else if ($\exists\ c \in C \mid c.symbol = s$)
    $r \leftarrow$ **evaluate**(*c.expression*)
  else if ($\exists\ h \in H \mid h.symbol = s$) $r \leftarrow h$
  else if ($\exists\ v \in V \mid v.symbol = s$)
    $r \leftarrow$ **evaluate**(*v.expression*)
  else if ($\exists\ p \in \bar{p} \mid p.symbol = s$) $r \leftarrow p$
  else forall $i \in I$
    $r \leftarrow i$.**get_value**(*s*)
    if ($r \neq$ *null*) **break**
  forall $m \in M$
    if ($m.symbol = s$)
      $r \leftarrow r\ m.op$ **evaluate**(*m.expression*)
  forall $n \in \mathcal{N}$
    forall $m \in n.M$
      if (*m.symbol* is of the form $s_1.s_2$)
        $\wedge\ (s_1 = N) \wedge (s_2 = s)$
        $r \leftarrow r\ m.op\ n$.**evaluate**(*m.expression*)
  return *r*

evaluate(*expression e*)
  forall $s \in \{$ non−function symbols referenced by *e* $\}$
    $v_s \leftarrow$ **get_value**(*s*)
    substitute all occurrences of $s \in e$ with v_s
  $r \leftarrow$ mathematical result of *e*
  return *r*

Figure 1. The get_value function, used to quantify the characteristics of a node.

through the execution of this function on a particular symbol that predictions are made of the design's performance. For example, $agent.get\_value(total\_load)$ would return a prediction of $agent$'s $total\_load$.

The validity of candidate organizational instance $\mathcal{O}$, derived from the satisfaction of the constraints within it, is defined as:

$$\begin{array}{ll} \mathcal{O} & \text{is valid iff} \quad \forall N \in \mathcal{O}.\mathcal{N}, N \text{ is valid} \\ N & \text{is valid iff} \quad \forall k \in N.K, \\ & (N.get\_value(k.symbol)\ k.op\ k.expression) = true \end{array} \tag{2}$$

3. Modeling Organizations

A working system that was developed for a distributed sensor network (DSN) domain will be used to motivate and demonstrate the capabilities of ODML. The DSN framework 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. The architecture, comprising roughly 40,000 lines of Java code and described in detail in [7], has been demonstrated in both simulation and real-world experiments.

We will proceed with an overview of how an ODML model was produced for this system. For clarity, the names of nodes and fields will be represented in italics. A graphical depiction of some aspects of the ODML DSN template can be seen in Figure 2a. Vertices in the graph, such as *sensor* and *track_manager*, correspond to nodes in the ODML model. Nodes can represent both tangible (e.g. *agent*) and intangible (e.g. *sector*) entities. Space precludes showing the complete specification, which is roughly 300 lines long.

Edges in the graph show relations between nodes. Those with a solid arrow represent has-a relations, and the corresponding label indicates the relation's magnitude. For example, consider the *track_manager* node, which corresponds to the DSN's track manager role. It has a number of *agents* defined by *num_agents*, modeled by the has-a relationship between the two nodes in Figure 2a, that represent the agents the role is bound to. In the DSN system, a track manager role may migrate among different agents over time. The magnitude *num_agents* is used to represent this behavior, and modifiers distribute the characteristics accordingly. It is worth noting that has-a relationships in the template may also be self-referential. This facilitates the modeling of hierarchies, by making it possible to represent organizations with varying numbers of levels.

A hollow-arrow edge represents an is-a relation, so *normal_agent* is a type of *agent*. Shaded nodes, such as *agent*, are abstract and cannot be directly instantiated. Thus, any node with a has-a relation with *agent* can instead substitute *normal_agent*. This indirection can be used to model alternatives with different capabilities. For example, suppose there were two types of agents available: a normal agent, and a "robust" agent that had more processing power but a higher cost. To model this, a *robust_agent* node is depicted that also has an is-a relation with *agent*, and can be substituted for *agent* in the same way.

Figure 2b shows a particular instance created from the template in Figure 2a. Vertices in the graph represent nodes,

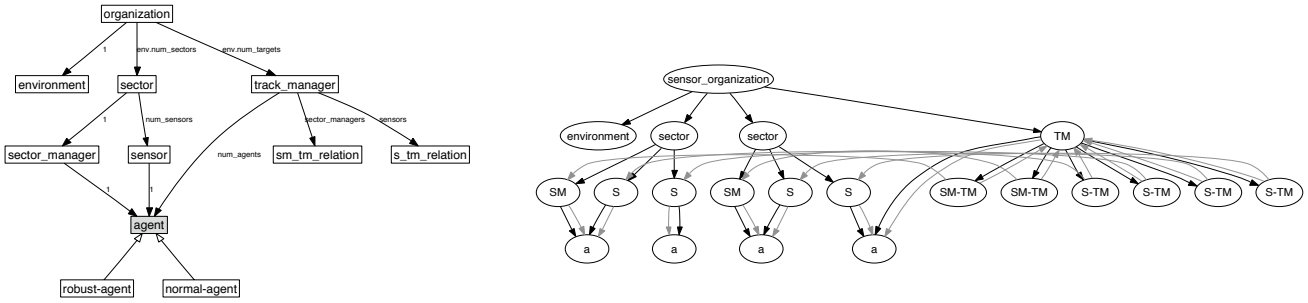


Figure 2. Example ODML (a) template and (b) instance structures for the DSN organization.

and a gray edge indicates the existence of a non-local modifier from the source to a field in the target. Black directed edges again represent has-a relationships, where template has-a relationships are replaced with a set of distinct edges of size *magnitude*. So, where *sector* → *agent* might have the magnitude *num_sensors* in the template, a discrete value of two has been chosen in this particular instance. Because of this, each sector in the instance has a has-a relationship with two distinct sensors (S). Normal agents (a), sector managers (SM), track managers (TM), and two kinds of track manager relations (SM-TM and S-TM), are also present.

The heart of any ODML model exists in the expressions encoded within nodes' fields. Each expression consists of a sequence of standard mathematical operations (e.g., $+, \div, x^y$, etc.) and a limited number of predefined functions (e.g., *min, max, sqrt, round, forallavg*, etc.). These expressions allow one to represent how different characteristics of the node may be computed, by combining local and nonlocal information to calculate new local values as defined by the process in Figure 1.

A selection of these fields, contained by the *track_manager* and *s_tm_relation* nodes, are shown in Figure 3. The former defines the track manager role, while the latter represents the relationship that role has with sensors in the environment. This fragment shows a simple set of expressions used to calculate the track manager's logical footprint (area) of a target as it moves through the environment. This area will depend on the amount of uncertainty the manager has in the target's location. In our model, this area will be a circle; line 10 shows how the *target_area* of a track manager is derived from the target's *uncertainty_radius*. The number of sensors presumed capable of sensing the target is the average number of sensor which lie within the target area. Therefore, although the number of *desired_sensors* is independent of the environment, the *actual_sensors_available* to the manager will depend indirectly on the *target_area* and *sensor_density*, as shown in line 15. The *requested_sensors* will be the minimum of the desired and available.

The number of measurements provided to the track manager is modeled in a similar way. The *actual_measurement_rate* in the sensor-track manager relationship is derived from the locally calculated *requested_measurement_rate* and *actual_measurement_ratio* computed by the sensor node. This value is then used in a pair of modifiers defined in lines 29 and 30 that specify for the track manager and sensor the actual number of measurements that will be taken.

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.

This description touches on just a portion of the model that was created for the DSN system. A range of characteristics not described here have also been incorporated, including the physical and task environment, agent interactions, multiple role assignments, dynamic role assignment, heterogeneity, geographic coalitions, 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.

4. Evaluating the Representation

Our previous work analyzed the effects that the DSN organization had on performance across a range of metrics [5]. In those tests, the number of agents in each sector was varied to demonstrate how changing the organization can have far-reaching consequences. To gauge the representa-

```
<node type="track_manager">
  <param>organization:org,environment:env,[sector]:sectors</param>
  <is-a>entity</is-a>
  <has-a name="agent" size="num_agents">agent(env)</has-a>
  <has-a name="sm_relations">forall(sm, sector_managers):sm_tm_relation(org, this, sm)</has-a>
  <has-a name="s_relations">forall(s, sensors):s_tm_relation(org, this, s)</has-a>

  <!-- Determine target bounds -->
  <constant name="uncertainty_radius">5</constant>
  <constant name="target_area">3.14 * uncertainty_radius^2</constant>

  <!-- Calculate requested measurement rate -->
  <constant name="desired_sensors">3</constant>
  <constant name="sensor_density">forallavg(sectors.sensor_density)</constant>
  <constant name="actual_sensors_available">target_area * sensor_density</constant>
  <constant name="requested_sensors">min(desired_sensors, actual_sensors_available)</constant>
</node>

<node type="s_tm_relation">
  <param>organization:org,track_manager:tm,sensor:s</param>

  <!-- Calculate actual measurement rate -->
  <constant name="requested_sensor_rate">tm.requested_sensors / org.total_sensors</constant>
  <constant name="requested_measurement_rate">tm.requested_measurement_rate * requested_sensor_rate</constant>
  <modifier name="s.requested_measurement_rate" op="+">requested_measurement_rate</modifier>

  <!-- Assign measurement communication load -->
  <constant name="actual_measurement_rate">requested_measurement_rate * s.actual_measurement_ratio</constant>
  <modifier name="tm.actual_measurement_rate" op="+">actual_measurement_rate</modifier>
  <modifier name="s.message_rr" op="+">actual_measurement_rate</modifier>
</node>
```

Figure 3. A portion of the raw ODML specification for the *track_manager* and *s_tm_relation* nodes.

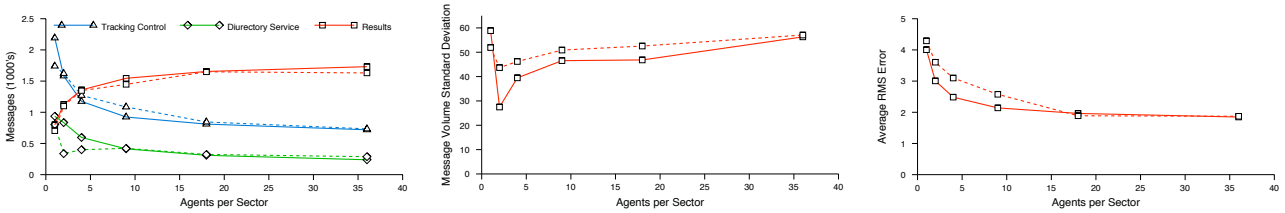
tional efficacy of ODML, we have used the model described in the previous section to create organizational instances that match those prior test runs. Characteristics defined in the ODML model make predictions for the same metrics that were originally tested, allowing us to calculate values that can be compared against the empirical results. These predictions were obtained using the get_value function described in Figure 1. 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. In this context, we are exploring only the accuracy of the ODML model's predictions. Additional details of the original empirical results can be found in [5]. 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 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. Our model does not capture these geographic differences, and will therefore generally have a lower estimated deviation.

To evaluate how our 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.

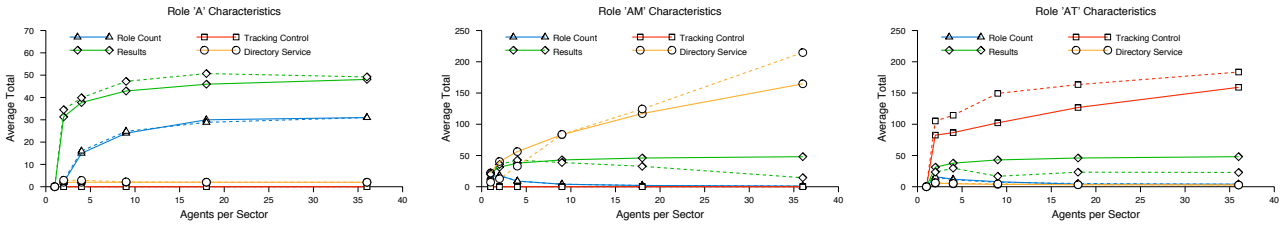
5. Using ODML to Design Organizations

We have thus far argued that organizations can have a tangible effect on performance, and it is therefore useful to be able to understand those effects when designing an agent system. The previous section demonstrated ODML's ability to correctly model and predict the global and local characteristics of particular organizational designs used in a



(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.

previously constructed, real-world application. Our ultimate goal, however, is to use this capability to support an automated organizational design process. It is this objective that motivates the need for the detailed information that makes up an ODML model. In this section we will outline how this has been accomplished, by using ODML as the foundation for a new design methodology. Space permits only a superficial description of this work, a more detailed description will be given in a later publication.

Recall that ODML representations are divided into two distinct classes: *templates* that encompasses the range of all possible organizations, and *instances* that are each particular organizations derived from a template. The range of possible instances in a template, defined by how its variables and has-a relations can be decided, constitutes an organizational space that can be searched.

For example, one of the *environment* node's variables expresses the range of possible *sensors_per_sector*, which controls the shape of part of the organization. Other uses of variables might be to decide the relative priority of an agent's tasks, or the amount of time it is willing to wait for a response. Decisions made for the *agent* has-a relationships in the three roles will determine the specific role-agent bindings that will be used. Sequences of similar decisions could also decide if the manager role will be distributed, or how tall a data processing hierarchy should be. A range of options are possible, and different choices will result in different organizations. Constraints defined within nodes have the opposite effect, by limiting the set of valid organizations to some subset of those that are possible.

As with other characteristics, an ODML model can be used to capture the expected *utility* of an organization by relating organizational characteristics with mathematical expressions. For example, in the DSN model, *utility* is based on *average_rms*, which is derived from the *rms_error* values of the *tracker* roles, which are themselves dependent on many other factors. By defining a field with the semantics of "utility", one can quantitatively evaluate and rank candidate organizations by comparing these fields' values. This provides a clear way to discriminate and decide among the alternatives that exist in the template's organizational space.

An automated organizational design process can be built upon these two concepts, by leveraging the predicted characteristics to navigate the organizational space. Each ODML model includes a description of the expected operating context, available resources, etc., in addition to the possible organizational structures. Given such a model, the most appropriate design (i.e. the one with maximal *utility*) can be found with an appropriate search of the organizational space defined by the model.

The number of possible organizations grows exponentially with the number of decisions that must be made in its construction. When organizations are being designed for a large number of agents, or when the template is particularly flexible, the space of possibilities can easily become intractable. Because of this, it is important to develop techniques able to cope with such situations.

One technique that we have implemented bounds the search by exploiting hard constraints that exist in the system. The expression-based knowledge in the ODML model makes this possible. Recall that all hard constraints must be satisfied in an organization for it to be considered valid (see Equation 2). If a constraint has become unsatisfied during the course of an organizational search, it may be reasonable to halt the search before the organization is fully formed, and backtrack from that point. Two issues complicate this process. The first is that constraints may be initially unsatisfied in a partially instantiated structure, and only become satisfied later in the decision making process. Additionally, because values may change nonmonotonically, a constraint can change its state repeatedly during the instantiation process. Both cases may cause a backtrack decision to be incorrect by missing valid organizations.

By analyzing the relationships that exist between fields, it is sometimes possible to determine when choices may be safely ignored. For example, in the DSN domain a single agent cannot control more than one sensor. Each agent has a *sensors_controlled* value, that is initially zero and later incremented using a modifier when it is assigned to a sensor. This restriction is modeled by the hard constraint *sensors_controlled* ≤ 1. For a particular organization with n sensors and a available agents, there are a^n possible assignments of agents to sensors. However, only $\binom{a}{n}$ of them are valid according to the *sensors_controlled* constraint. If it were possible to detect when an invalid assignment had been made before all organizational decisions have been completed, one could bound the search at that point and backtrack to where the constraint was satisfied.

Because role assignments are permanent, there is no decision that could be made to reduce the number of sensors controlled by an agent. Therefore, it is reasonable and correct to bound the search and backtrack if an agent is ever found to control more than one sensor, because that constraint will never be satisfied. While a human expert might intuitively know to avoid such impossible configurations, a generic search process is too myopic to perceive this fact. However, the relationship that this knowledge is based upon is represented in the organizational template, in that *sensors_controlled* is affected by only that one type of modifier from the *sensor* node, that increments the value by a positive constant. It is possible to use this information to deduce the monotonically increasing trend of *sensors_controlled*, and backtrack when appropriate.

Our technique uses partial derivatives on a field's dependent variables as a general way of determining the trend of that field's value. The trends for a constrained value and the expression it is constrained by are determined by first recursively finding the trends of all the fields they depend on. By taking the partial derivative with respect to each field referenced by an expression, along with the trend of each field, the trend of the original expression may be determined. With this information one can automatically detect when constraints have become unsatisfiable, and correctly bound the search.

5.1. Applying Designs to Actual Systems

Once the search process has completed successfully, a particular design will be available that must then be applied to a running system, assuming it has not already been gradually phased in in an emergent fashion as described above. ODML's free-form nature precludes a simple, straightforward mapping from design to system for an arbitrary model. For example, parts of the model may clearly correspond to tangible artifacts that can be directed, such as roles or agents. Other parts may be more ephemeral, or be there solely to model an environmental response that needs no instruction. Yet another class of information may not correspond to any specific process, despite the fact that it contains details that must be correctly incorporated for the organization to function. The *sector* node from the sensor network model is such a case, because although it is only a logical construct it still contains vital information, such as the size and membership of the set of sensors belonging to it.

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

In practice, the construction process usually falls somewhere between these two cases, where some details may be directly used and others require more effort on the part of the designer or running agent to gain access to. For example, a simulation environment has been created to evaluate a model of a hierarchical information retrieval domain (for more details, see [4]). This simulation takes an ODML instance model as input, and uses it to create the environment and the agent organization that operated within it. A bootstrap system starts the process by determining how many

agents will exist in the system. Once created, individual agents learn of the set of roles they are expected to take on by obtaining the appropriate *agent* node and finding the *role* nodes that have a has-a relationship with it. These roles can then be created and bound to the agent. Each role is responsible for inspecting its counterpart in the instance model to obtain any role-specific parameters. Finally, each role also determines the set of other roles that it should be interacting with and how those interactions should take place, which is also accomplished primarily by following has-a relationships. Other information present in the model, such as the performance metrics originally used to estimate the design's utility, can be used to evaluate runtime behaviors. This can be particularly useful when the system or environment is subject to change, by providing a set of expectations that observed behaviors can be compared against.

The end result of this inspection process is a system that can take an arbitrary information retrieval instance model as input and create a running system from it. The code required to do this, however, is specific to the particular model in question. The model does not indicate exactly what an *agent* or each *role* should do on a moment-by-moment basis. Nor does it define the communication protocols or techniques needed to perform local operations. Such details are typically much too fine grained to be practically incorporated into a model. If these elements are crucial to organizational performance then they will be appropriately modeled with an expression and the relevant characteristics used to parameterize the running code, but it would be unusual for model to provide all the details necessary to fully specify those behaviors from an implementation standpoint.

6. Conclusions

The ODML-based approach that we have presented in this paper is both general and flexible enough to model a range of common organizational characteristics. It is particularly useful when describing features that have a quantitative character. To support this claim, we have created and tested a complete model of a previously existing sensor network framework. Other work has also used ODML to model a hierarchical information retrieval service [4], as well as more abstract, theoretical problems such as SUBSET-SUM and TILINGS. These models demonstrate ODML's ability to accurately predict the small- and large-grained behaviors of an organizationally-driven agent system.

We believe the detailed, quantitative knowledge embedded in ODML models is relevant because it provides the foundation for prescriptive technologies, such as the automated design service outlined above. We have described one strategy currently used to cope with the potentially large organizational search space by inferring value trends to find unsatisfiable constraints, and also discussed how the information obtained from this search can be put to use in practice. Additional techniques are currently being developed. Armed with such techniques, the full potential of ODML can be realized by using its flexible but detailed representation of organizational possibilities to effectively design agent organizations.

References

[1] S. 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.

[2] V. Dignum, J. Vazquez-Salceda, and F. 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.

[3] M. Fox, M. Barbuceanu, M. Gruninger, and J. Lin. An Organizational Ontology for Enterprise Modeling. In M. J. Prietula, K. M. Carley, and L. Gasser, editors, *Simulating Organizations: Computational Models of Institutions and Groups*, pages 131–152. AAAI Press / MIT Press, 1998.

[4] B. Horling and V. Lesser. Quantitative Organizational Models for Large-Scale Agent Systems. In *Proceedings of the International Workshop on Massively Multi-Agent Systems*, pages 297–312, Kyoto, Japan, December 2004.

[5] B. Horling, R. Mailler, and V. Lesser. A Case Study of Organizational Effects in a Distributed Sensor Network. In *Proceedings of the International Conference on Intelligent Agent Technology (IAT 2004)*, pages 51–57, Beijing, China, September 2004.

[6] J. F. Hübner, J. S. Sichman, and O. 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.

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

[8] T. W. Malone, K. Crowston, J. Lee, B. Pentland, C. Dellarocas, G. Wyner, J. Quimby, C. S. Osborn, A. Bernstein, G. Herman, M. Klein, and E. O'Donnell. Tools for inventing organizations: Toward a handbook of organizational processes. *Management Science*, 45(3):425–443, 1999.

[9] C. Sierra, J. Sabater, J. Augusti, and P. Garcia. SADDE: Social agents design driven by equations. In F. Bergenti, M. Gleizes, and F. Zambonelli, editors, *Methodologies and software engineering for agent systems*. Kluwer Academic Publishers, 2004.

[10] M. Sims, D. Corkill, and V. 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.