Knowledgeable Automated Organization Design for Multi-Agent Systems

Mark Sims, Daniel Corkill, and Victor Lesser
University of Massachusetts
Multi-Agent Systems Laboratory
{msims,corkill,lesser}@cs.umass.edu

UMass Computer Science Technical Report 07-43

August 16, 2007

Abstract

The ability to create effective multi-agent organizations is key to the development of larger, more diverse multi-agent systems. In this article we present KB-ORG: a fully automated, knowledge-based organization designer for multi-agent systems. Organization design is the process that accepts organizational goals, environmental expectations, performance requirements, role characterizations, and agent descriptions and assigns responsibilities and roles to each agent. These long-term responsibilities and roles serve as organizational-control guidelines that are used by each agent in making minute-by-minute operational control decisions.

An important aspect of KB-ORG is its efficient, knowledge-informed search process for designing multi-agent organizations. KB-ORG uses both application-level and coordination-level organization design knowledge to explore the combinatorial search space of candidate organizations selectively. KB-ORG also delays making coordination-level organizational decisions until it has explored and elaborated candidate application-level agent roles and responsibilities. This approach significantly reduces the exploration effort required to produce effective designs as compared to modeling and evaluation-based approaches that do not incorporate design expertise.

In this article, we use examples from the distributed sensor network (DSN) domain to show how KB-ORG uses situational parameters as well as application-level and coordination-level knowledge to generate organization designs. We also show that KB-ORG designs effective, yet substantially different, organizations when given different organizational requirements and environmental expectations.

1 Introduction

Without an organizational structure, large-scale multi-agent systems suffer from inefficiencies that arise from repeated planning, task allocation, and coordination [3].

The ability to design effective multi-agent organizations, therefore, is key to the development of larger, more diverse multi-agent systems. In this article, we describe KB-ORG, a fully automated, knowledge-based organization design framework. We discuss the situational parameters and knowledge necessary to provide to KB-ORG to design organizations in an application domain. We also discuss the search process that enables KB-ORG to select appropriate organization coordination mechanisms given those inputs. While KB-ORG's design process is general, we ground the discussion with examples from the distributed sensor network (DSN) domain and show how situational parameters as well as application-level and coordination-level knowledge are provided to KB-ORG to generate organization designs.

In keeping with the emphasis pioneered by March and Simon, we make a sharp distinction between organizational and operational control. Organizational control is characterized by long-term guidelines while operational control pertains to the specific moment-to-moment coordination and control activities of agents as they continuously elaborate their organizational guidelines [28]. Specifically, organizational control provides long-term organizational goals, roles, and responsibilities as guidelines for each agent.[1] By restricting an agent's set of choices, these guidelines reduce the complexity of each agent's operational decision making, lower the cost of distributed resource allocation and agent coordination, help limit inappropriate agent behavior, and reduce communication requirements [4]. In a DSN, for example, organizational guidelines specify which agents are responsible for scanning for new vehicles, tracking vehicles, data interpretation, etc. The guidelines also specify the other agents that each agent may need to communicate with in order to meet its responsibilities. Operational control, on the other hand, involves the detailed coordination and control activities of agents as they perform particular tasks as laid out in the organizational guidelines. Operational activities in a DSN include having multiple agents dynamically agreeing to direct their scanning devices toward a particular location at a particular time. In maintaining the distinction between organizational and operational control, KB-ORG does not attempt to pre-determine operational activities or detailed control policies. Rather than describe in detail how particular operational decisions are to be made, KB-ORG assigns roles to agents and ensures that resources and coordination mechanisms exist for agents to make efficient operational decisions during the life of the organization.

To design organizations, KB-ORG employs a knowledge-informed search process that applies both application-level and more generic organization coordination-level knowledge to organizational goals, performance requirements, and environmental expectation information. Using input such as agent descriptions and role characteristics, KB-ORG first finds groups of agents for each goal that have the combined resources to achieve that goal. Then, as necessary, it adds organizational structures that support the agents in coordinating their actions with regard to their joint goals. KB-ORG also reserves agent resources to enable the dynamic formation of teams where more long-term organizational structures are not appropriate.

The major contribution of our work is KB-ORG's efficient, knowledge-informed search process for designing multi-agent organizations that

[1]We provide detailed, technical definitions of "goal" and "role" in Section 3.1.

- utilizes both application-level and coordination-level knowledge to bind agents to roles and
- delays coordination-level decisions until it has sufficient information about application-level bindings.

Through knowledgeable exploration, KB-ORG reduces the amount of search required in designing effective, yet substantially different, organizational forms when given different organizational requirements and environmental expectations. With its use of both application-level and coordination-level organization design knowledge, KB-ORG is able to explore the combinatorial search space of candidate organizations selectively, requiring much less search than is performed in the modeling and evaluation-based work of Horling [17]. Indeed, Horling showed that the problem of finding a suitable organization is NEXP-complete [17], and his results indicate the significant reduction in exploration effort that can be obtained by using a knowledgeable organization-design approach.

Knowledgeable Pruning The key to KB-ORG's approach to organization design is its use of knowledgeable pruning as opposed to blind search and evaluation. We categorize the knowledge KB-ORG uses into two types: application-level and coordination-level. The former includes knowledge of:

- how to decompose organizational goals
- how to select application-level roles for goals
- how to subdivide application-level roles among agents

The latter includes knowledge of:

- when to trigger the search for coordination strategies
- how to select coordination strategies
- how to subdivide coordination-level roles among agents after a strategy is chosen

Both types of knowledge are provided to KB-ORG in part through parameterized application-level and coordination-level role characterizations and in part through organization design functions.

Application-level roles provide alternatives for achieving organizational goals. In a DSN, for instance, the organizational goal of scanning for new vehicles could be achieved with either a general purpose scanning role or a more directed, higher resolution role. The choice depends on how the environmental parameters and performance requirements affect the roles' characterizations. The coordination-level roles provide mechanisms for coordinating the chosen application-level roles when those roles are split among a set of agents due to spatial, temporal, resource, or functional limits of individual agents. For instance, if a general purpose scanning role is assigned to a number of agents with limited range, those agents could use a peer-to-peer mechanism for coordinating their scans, a managerial structure, or another organization coordination mechanism (see work by Horling [18] for a survey of organization types).

Again the choice depends on the values of the parameters of the coordination-level role characterizations. The organization design functions provide additional knowledge to KB-ORG. For instance, in our example DSN an application-level function provides an algorithm for finding scanning agents that together cover the region to be monitored. A coordination-level function, for example, provides knowledge about when to add a level of hierarchy in an hierarchical organization.

An additional benefit to KB-ORG's categorization of organization design knowledge is that by separating application and coordination issues, our approach reduces the organizational search space further by recognizing when agents do or do not need coordination. More specifically, splitting an application-level role among a group of agents triggers the formation of a coordination goal which must be satisfied by coordination roles. If no coordination goal is triggered, the search process does not explore the use of coordination roles. Although other researchers [42, 7, 8, 43] make similar distinctions between application-level and organization coordination-level knowledge, KB-ORG is the first to utilize this distinction in an automated system to prune and direct a search process for organization candidates.

Article Structure The remainder of the article is organized as follows. Section 2 gives a high-level overview of KB-ORG's design process as well as a description of the example DSN domain. Section 3 provides all details of the KB-ORG framework except the search process which is covered in Section 4. Section 4 also compares our search process to that in Horling's work on ODML [17]. Section 5 provides examples of organization designs generated by KB-ORG for a DSN under various environmental conditions and performance requirements. In Section 6 we discuss related work. We conclude and discuss future work in Section 7.

2 Overview of KB-ORG's Design Process and the Example Domain

The input to KB-ORG consists of

- situational parameters including
  - environmental conditions and expectations
  - performance requirements
  - agent capabilities
- application-level organization goals parameterized by the environmental conditions and performance requirements
- knowledge including
  - application-level role characterizations
  - coordination-level role characterizations

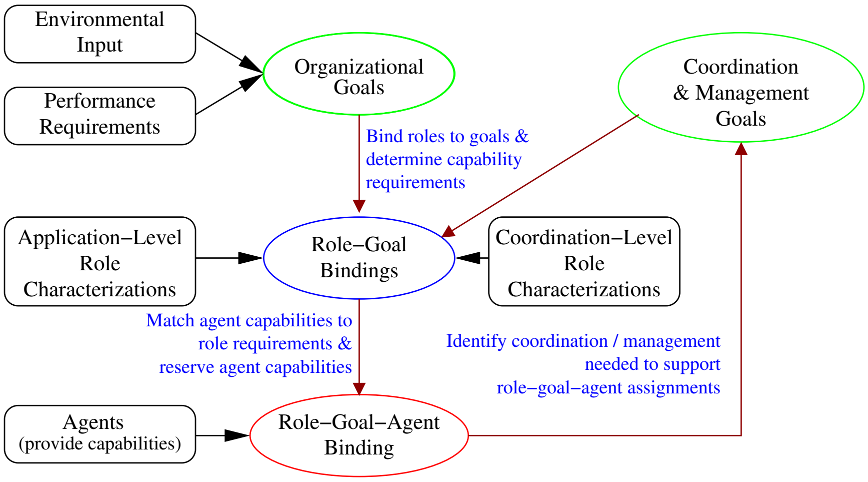


Figure 1: The KB-ORG Organizational Design Process

- application-level organization design functions
- coordination-level organization design functions

KB-ORG's output is an assignment of both application-level and coordination-level roles to each agent such that the performance requirements are satisfied and the organization operates effectively over anticipated environmental conditions.

As noted in Section 1, the KB-ORG approach to designing multi-agent organizations exploits the separation between application-level and coordination-level issues. The former, shown on the left side of Figure 1, involves binding application-level roles and agents to organizational goals. The latter, shown on the right, pertains to the coordination mechanisms that are needed when multiple agents are required to perform the application-level roles jointly (e.g., when several agents are required to scan a region cooperatively in a distributed sensor network). The result is a set of bindings for each agent to both application-level and coordination-level roles. The bindings specify not only the roles an agent is assigned to, but also the other agents and their roles that the agent sends information to and receives information from.

Consider our example DSN which is derived from the DARPA EW Challenge Problem [27]. In it, vehicles (or targets) take arbitrary paths through a region monitored by MTI Doppler radars. Each radar has a fixed location and is controlled by an agent. Each radar can scan a circular area with a 20 foot radius; however, each is only able to scan in one of three 120° regions at a time, and because each determines only amplitude and frequency information, data from multiple, coordinated agents must be fused to determine the position of a target. Lastly, the agents operate in a communication bandwidth constrained environment. These factors give rise to the need for tight coordination among the sensors to allow for proper triangulation of target position and to enable continuous measurements as vehicles move throughout the region [17].

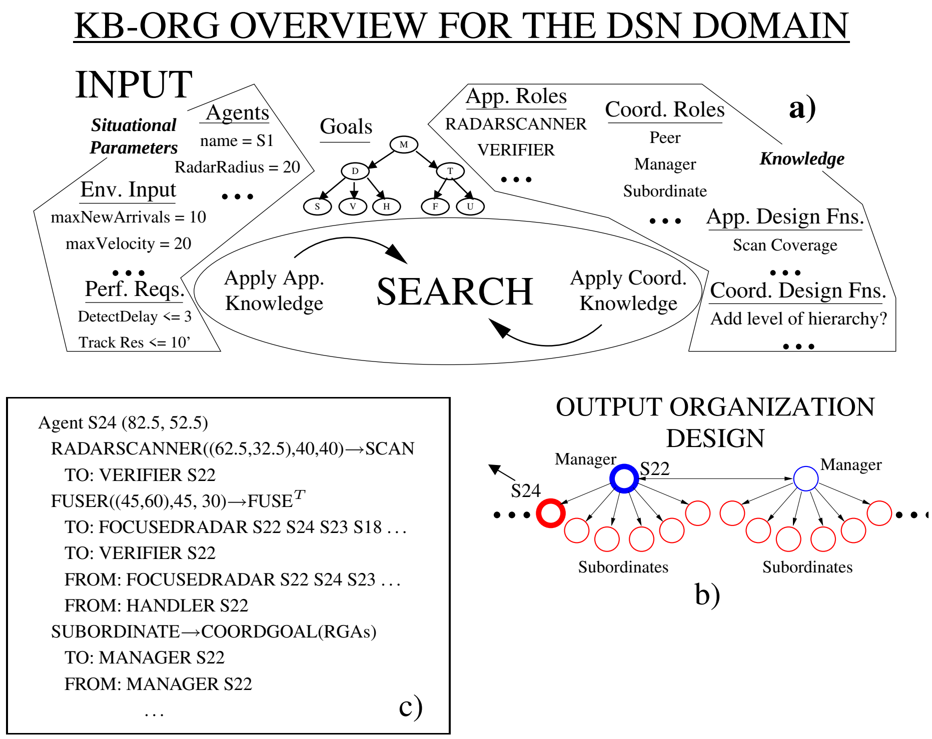


Figure 2: Overview of KB-ORG for the DSN domain. a) The input to KB-ORG consists of environmental information, performance requirements, agent descriptions, an organizational goal tree, application-level roles, coordination-level roles, application-level organization design functions, and coordination-level organization design functions. KB-ORG then engages in a search process that utilizes both application-level and coordination-level knowledge. b) The output is an organization design that includes role bindings for each agent. c) The bindings of a single agent, S24, are shown in the inset.

Figure 2 provides an overview of the KB-ORG design process for the example DSN domain. As Figure 2a shows, the situational parameters consist of environmental information, performance requirements, and agent capabilities. The input also includes an organizational goal tree and knowledge in the form of application-level and coordination-level role characterizations as well as application-level and coordination-level organization design functions. For instance, the environmental input gives the expected traffic volume, spatial density, vehicle velocity, and available communication bandwidth. A performance requirement is to detect each new vehicle with a delay of no more than 3 seconds. The agent descriptions provide details on each agent's capabilities, such as the radius of the viewable area of its radar. The organizational leaf goal labeled "S" in the goal tree pertains to scanning for new vehicles. The application-level roles provide roles which are available to choose from to satisfy each leaf goal and include radar-based scanning, detection verification, data interpretation, and other roles.

Coordination-level roles include peer-to-peer coordinators as well as manager and subordinate roles. The organization design functions include, for instance, an algorithm for providing coverage of an area to be monitored by sensor agents and a function for determining if another level of hierarchy should be added to an hierarchical organization. We discuss all input to KB-ORG in detail in Section 3 except for the organization design functions which we cover in Section 4.

As Figure 2a shows, with the input in place, KB-ORG engages in a search process that utilizes both application-level and coordination-level knowledge. In the search KB-ORG uses the application-level knowledge parameterized by the situational parameters to bind agents to roles that satisfy each leaf goal in the goal tree. KB-ORG then uses the coordination-level knowledge to find coordination mechanisms for the agents playing the application-level roles. The best mechanism depends on a number of factors. If only a few agents are necessary to cover the monitored area, a peer-to-peer coordination mechanism may be best. If many agents are required, vehicles arrive frequently, and scanning resources are scarce, a multi-level hierarchical coordination mechanism may be appropriate. The key is that in the search KB-ORG makes the decision based on knowledge in the coordination-level role characteristics and design functions, rather than via a brute force search.

Continuing with Figure 2, the output of the search process is an organization design that includes role bindings for each agent. Figure 2b shows that a one-level hierarchy is chosen for the particular input of the figure. Figure 2c shows the details of the bindings of one agent, S24, in that design. The bindings for S24 include two application-level roles, RADARSCANNER and FUSER, with their associated parameters and context. The bindings also include a coordination-level role, SUBORDINATE, which refers to its relationship to an agent performing a MANAGER role. In addition the bindings include the other agents and their roles that S24 sends information to and receives information from in the performance of its own roles. For instance, in its SUBORDINATE role, S24, sends information to and receives information from agent S22 in its capacity as a MANAGER. S24 sends information to and receives information from S22 for a number of other reasons as well. In its roles as RADARSCANNER and FUSER, S24 sends information to S22 in its role as VERIFIER. S24 receives information from S22 in their roles as FUSER and HANDLER respectively.

Note that the roles for the DSN we discuss in this article are at a finer granularity than those used in the implementation by Horling, et al. [20] for the EW Challenge Problem. In that work agents can take on one of only three roles: sector manager, track manager, and sensor manager [17]. Those three roles were designed for the particular environmental situation of the EW Challenge problem. Each is the combination of several functions. For the work discussed in this article we have divided the roles into more discrete responsibilities to allow KB-ORG to explore different combinations of responsibilities within agents. We discuss all the finer grain roles in Section 3.1 and in Section 5 show that under environmental conditions that differ from those in the EW Challenge Problem, the KB-ORG design process assigns combinations of responsibilities that differ from those in the Challenge Problem implementation.

Environmental Inputs	
maxNewArrivals	10
maxTracks	10
maxVelocity	$20mph$
vehicleWidth	$3'$
(x,y)	(0,0)
length	$90'$
width	$90'$
bandwidth	1000 msgs

Performance Requirements	
Detect Delay	$\leq 3sec$
Track Resolution	$\leq 10'$

Figure 3: Environmental inputs and performance requirements for the DSN example.

3 The KB-ORG Design Process in Detail

In this section we describe the majority of KB-ORG's components in detail. Figures 4, 7, 10, and 13 provide the formal definitions of each component referred to throughout this section. We defer discussion of organization design functions until Section 4.

3.1 Application-Level Inputs

Environmental Inputs Referring to the left side of Figure 1, the set of environmental inputs, E, (Def. 1, Fig. 4) is a set of feature and expression pairs that gives the expected environmental characteristics over time.[2] In our example DSN, features describe expected vehicle characteristics, spatial information about the area to be monitored, and the communication bandwidth information given by the number of messages able to be sent at a given time. The corresponding mathematical expressions supply values for those features. For example, a set of environmental inputs for the DSN is

$$\begin{aligned} E = &\{\langle maxNewArrivals, 10\rangle, \langle maxTracks, 10\rangle, \langle maxVelocity, 20mph\rangle, \\ &\langle vehicleWidth, 3'\rangle, \langle (x, y), (0, 0)\rangle, \langle length, 90'\rangle, \langle width, 90'\rangle, \\ &\langle bandwidth, 1000msgs\rangle\} \end{aligned}$$

which is summarized in Figure 3. In a more detailed example, an expression for a feature could specify a distribution such as a distribution over vehicle arrivals. We utilize dot notation to indicate the value of features of the environmental inputs. For instance, $E.maxNewArrivals = 10$. We use dot notation in the same manner for the other KB-ORG elements as well.

Performance Requirements The set of performance requirements, Q, (Def. 2, Fig. 4) specifies the requirements that the organization must meet to satisfy the organizational goals. Similar to the environmental inputs, Q is a set of features and arbitrary mathematical expressions, which may be functions of elements of E, that determine values for the features.

[2]The developer specifies the environmental and all other inputs in a space-delimited text file that KB-ORG parses. We are exploring alternate formats for the configuration file such as XML to provide more standardized input in future versions of the code.

Definition 1 (Environmental Inputs, E).

$$E \quad = \quad \{\langle e_1, expr_1\rangle, \ldots, \langle e_n, expr_n\rangle\}$$

for $1 \leq i \leq n$, where e_i is an environmental feature, and $expr_i$ is a mathematical expression.

Definition 2 (Performance Requirements, Q).

$$Q \quad = \quad \{\langle q_1, expr_1\rangle \ \ldots, \langle q_m, expr_m\rangle\}$$

for $1 \leq j \leq m$, where q_j is a requirement, and $expr_j$ is a mathematical expression. $expr_j$ may be a function of the e_i in E.

Definition 3 (Organizational Goal). *An organizational goal g contains the following fields:*

$E_g \subseteq E$ (see Definition 1); if g is a root goal, $E_g = E$.

$Q_g \subseteq Q$ (see Definition 2); if g is a root goal, $Q_g = Q$.

TAL: the to-be-assigned list for g (see Definition 5); $TAL = \emptyset$ if g is not a leaf goal.

$parents$: the list of goals that are parents of g in the goal tree

$children$: the list of goals that are the children of g.

Definition 4 (Goal Tree, T). *A goal tree $T = (G, E)$ is a directed graph where G is a set of organizational goals and E is the set of edges between them. An edge exists between $g_1, g_2 \in G$ iff $g_2 \in g_1.children$.*

Definition 5 (To-Be-Assigned-List, TAL). *The to-be-assigned-list, TAL, of an organizational goal g is a list of responsibilities that must be satisfied in order for g to be satisfied.*

Definition 6 (Leaf Goals, L). *Let $L = \{g \in G | g.children = \emptyset\}$ be the set of leaf goals in goal tree $T = (G, E)$.*

Definition 7 (Goal Satisfaction). *A leaf goal g is satisfied if agents bound to it perform the responsibilities in $g.TAL$ within the performance requirements. In the case of conjunctive goals, a non-leaf goal is satisfied if all of its children are satisfied. In the case of disjunctive goals, a non-leaf goal is satisfied if at least one of its children is satisfied.*

Figure 4: Definitions of environmental inputs and performance requirements as well as components pertaining to organizational goals

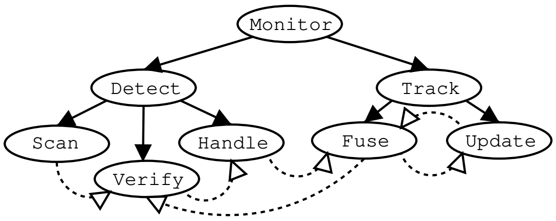


Figure 5: Example DSN goal tree and associated communication graph (dotted edges)

Most commonly, the expressions represent constraints on some aspect of the organization. For instance, in a DSN scenario one requirement might be that the time delay between a vehicle's arrival in the environment and its detection be at most 3 seconds. Another might be that the track resolution be at most 10 feet. Under those assumptions, Q for the DSN is,

$$Q = \{\langle DetectDelay, \leq 3sec\rangle, \langle TrackResolution, \leq 10'\rangle\}$$

which is also shown in Figure 3.

Note that in this work we do not explicitly specify a domain ontology as in other organization design research [42, 11, 12, 10]. As will be shown below, the connections between elements of the environmental input and the performance requirements are made in the quantitative equations that define the characteristics of roles. Furthermore, we assume that the semantic meanings of the components of the environmental input and performance requirements are contained within the implementation of the roles rather than in an explicit ontological representation. All that we need is sufficient information about roles and goals to understand what roles apply to a specific goal, how a role can be split among agents to satisfy a goal, and how effective roles are in satisfying specific goals.

Organizational Goals Returning to Figure 1, an organizational goal, g, (Def. 3, Fig. 4) is a high-level, long-term objective. We represent organizational goal decomposition as a goal tree, T, (Def. 4, Fig. 4) whose nodes are goals and edges represent subgoal relations. The developer is responsible for providing the decomposition based on his or her knowledge of the domain. Figure 5 shows a goal tree[3] for our example DSN. The root MONITOR decomposes into subgoals for detecting and tracking vehicles. Similarly, DETECT and TRACK can be further decomposed. Under DETECT, the child SCAN pertains to scanning the monitored area for new vehicles, VERIFY to determining if a detection from the SCAN goal is actually a new vehicle, and HANDLE to setting up activities associated with a new vehicle detection, in this case tracking it. The UPDATE goal pertains to sending position information about tracked vehicles while the FUSE goal pertains to fusing data from track updates.

The root goal of the tree is parameterized by the environmental inputs and performance requirements. Subgoals inherit their parents' parameters unless the developer

[3]The goal tree depicted is conjunctive. However, there is nothing that precludes the use of AND/OR goal trees.

Goal	*TAL*
SCAN	Scanning
VERIFY	Verifying
HANDLE	Handling
FUSE	Fusing
UPDATE	Updating

Figure 6: To-be-assigned lists of the leaf goals in Figure 5.

specifies otherwise.[4] If a goal, g, is leaf goal, it also has a non-empty to-be-assigned list, TAL, (Def. 5, Fig. 4) of responsibilities. The elements of the TAL are activities that the developer specifies as being necessary to perform over the life of the organization in order for the associated leaf goal to be satisfied. Therefore, a leaf goal is *satisfied* (Def. 7, Fig. 4) if agents bound to it perform the responsibilities in its TAL within the performance requirements on it. Figure 6 shows the to-be-assigned lists of the leaf goals of Figure 5. For example $SCAN.TAL = \{scanning\}$. Although the goals in our example have single responsibilities in their TALs, in general a goal will entail multiple responsibilities.

The above information is sufficient to specify an organizational goal formally for the KB-ORG process. For instance, we define the leaf goal SCAN by the following:

$$\begin{aligned}
SCAN.E_g &= \{E.(x,y), E.length, E.width, E.maxNewArrivals,\\
&\quad E.maxVelocity, E.vehicleWidth\}\\
SCAN.Q_g &= \{Q.detectDelay\}\\
SCAN.TAL &= \{scanning\}\\
SCAN.parents &= \{DETECT\}\\
SCAN.children &= \emptyset
\end{aligned}$$

Note that SCAN inherits all parameters but $E.maxTracks$ and $E.TrackResolution$.

We show how goal satisfaction is determined in Section 3.2 when we discuss role-goal-agent bindings.

Application-Level Roles As in traditional planning, where goal decomposition continues until leaf goals can be achieved by primitive actions, organizational goal decomposition continues until the TALs of leaf goals can be fulfilled by assigning application-level roles (Def. 11, Fig. 7). Unlike planning actions, roles are atemporal “job descriptions” lasting throughout the organization’s lifetime. In this work a single role may be suitable for achieving more than one leaf goal and multiple roles may be suitable for achieving a single goal. A secretary role, for instance, may be suitable for a number of

[4]Equivalently, each leaf goal could be parameterized by the complete set of environmental inputs and performance requirements. Later in the design process those parameters which are not required to determine requirements could be ignored. We chose to let the developer omit parameters as a way to express knowledge that certain parameters do not pertain to certain leaf goals.

Definition 8 (Assignable-list, AL). *An assignable-list AL of a role is a list of responsibilities that the role when assigned to an agent is able to perform.*

Definition 9 (Quality Function, qf). *Let $qf : L \to \mathbb{R}$ be a quality function of a role. L is the set of leaf goals in the goal tree T. qf indicates how well the role achieves a leaf goal $g \in L$.*

Definition 10 (Role Constraints, K). *Let K be a set of constraints for role r. Each constraint is a pair $\langle k, R \rangle$ where k is a tuple of variables and R is a relation. Note that the elements of k may themselves be functions of the fields of the goal g that the role is bound to.*

Definition 11 (Application-Level Role). *An application-level role is an atemporal activity description defined by the following fields:*

- *an assignable-list AL*
- *a quality function qf*
- *a set of requirement functions K*

Definition 12 (Application-Level Roles). $R = \{r_1, r_2, \ldots r_{n_r}\}$ *is the set of application-level roles supplied by the developer where n_r is the number of application-level roles supplied.*

Definition 13 (Communication Graph). *Let $C = (L, V)$ be the communication graph. L is the set of leaf goals in goal tree T, and V is the set of edges. An edge (u, v) exists if information must flow from goal u to goal v.*

Definition 14 (Agent). *An agent a is defined by the following fields:*

- *a set of pairs $\phi = \{\langle f_1, v_1 \rangle, \ldots \langle f_n, v_n \rangle\}$ of features and their values. f_i, $1 \leq i \leq n$, is the feature and v_i is its value. An example is the x-coordinate of the agent's location along with its value.*
- *a set $CAP = \{\langle c_1, v_1 \rangle \ldots, \langle c_m, v_m \rangle\}$ of capabilities. c_j $1 \leq j \leq m$ is a capability and $v_j \in \mathbf{R}$ is its value.*
- *a set of triples $\rho = \{\langle r_1, d_1, m_1 \rangle, \ldots, \langle r_n, d_n, m_n \rangle\}$. r_i, $1 \leq i \leq n$, is a role that the agent is able to play. d_i is an abstract view of r_i's demand on the agent's resources called the percent drain caused by r_i. m_i is the number of messages per time the agent sends during its operational performance of r_i (m_i may be a function).*

Figure 7: Definitions pertaining to application-level roles, the communication graph, and agent descriptions.

Role	*AL*
RADARSCANNER	Scanning, Updating
FOCUSEDRADAR	Updating, Scanning
VERIFIER	Verifying
HANDLER	Handling
FUSER	Fusing

Figure 8: Application-Level Roles for the DSN example showing each role's assignable list.

subgoals in a human organization. Similarly, in a large organization the right role for answering phones could be a secretary role while in a small organization the CEO may answer every call. Also, since roles are associated only with leaf goals, we do not have a notion of role composition or role hierarchies as in other work [41, 42, 38].

To enable the automated selection of roles for goals, each role r has an assignable-list, AL, (Def. 8, Fig. 7) of responsibilities that it can perform. The role also has a real-valued quality function, qf, (Def. 9, Fig. 7) indicating how well the role achieves a goal and a set K of role constraints (Def. 10, Fig. 7) that must be met by agents performing the role when it is bound to a goal. In the current implementation, the developer supplies this information to KB-ORG as Java classes. For each application-level role, the developer builds a class that extends a base role class. The implementation of the subclass contains the assignable-list, quality function, and role constraints.

Figure 8 shows the roles and their assignable lists available in the DSN example. RADARSCANNER is a general purpose scanning role whose primary purpose is to perform sweeps of an area for new vehicle detections. As such, its AL contains the TAL *scanning* from the leaf goal SCAN. Also, because RADARSCANNER could be used in some applications to track vehicles rather than just detect new ones, its AL contains the TAL *updating* from the goal UPDATE as well. Thus, for the RADARSCANNER role $AL = \{Scanning, Updating\}$. FOCUSEDRADAR is a directed scanning role specifically for sending vehicle track updates; however, since its information could be used for new vehicle detections, its AL also contains both *updating* and *scanning*. Each remaining role has only a single element in its AL. VERIFIER is a role responsible for verifying that vehicle detections are not of vehicles that are currently being tracked. HANDLER is a role that performs the necessary actions to start the tracking process, and the FUSE role is a role for fusing data from agents playing the FOCUSEDRADAR role.

For each role, the quality function qf is dependent on the goal the role is bound to. The definition of the quality function is quite wide open. It can be any real-valued function as long as two roles that can satisfy the same goal have quality functions that can be compared and ordered with respect to one another. More specifically, any role whose assignable list contains a goal's TAL may be bound to that goal. However, since this may be true for multiple roles, the quality functions of the roles help to determine which role is the most suitable for a goal given that goal's parameters. The quality functions provide a local heuristic for ordering which roles should be explored first for satisfying goals. For instance, since RADARSCANNER and FOCUSEDRADAR contain the same elements in their ALs (see Figure 8), by comparing their quality functions, the KB-ORG process can determine which is more likely to be useful in a given environment before considering agent bindings. In contrast to Horling's work [17] which does not have an analog, this serves to prune the search space early in the design process by focusing the search on finding agents able to perform the role most likely to satisfy the goal effectively. The quality function is used to guide the automated search process using expert knowledge that the designer may possess as to the relative suitability of one role over another for particular goals, environmental conditions, and performance requirements. Thus, for RADARSCANNER in the DSN example,

$$qf_{RS}(SCAN) = 1.0 \quad (1)$$
$$qf_{RS}(UPDATE) = 0.5 \quad (2)$$

for all E and Q. Similarly, for FOCUSEDRADAR,

$$qf_{FR}(SCAN) = 0.5 \quad (3)$$
$$qf_{FR}(UPDATE) = 1.0 \quad (4)$$

for all E and Q. These provide knowledge that RADARSCANNER is better for the SCAN goal in all domains. As will be seen in Section 4, although the quality functions may make one role seem better than another, in some circumstances during the search for an organization exploring another role with lower quality may be necessary.

As with the quality function, the role constraints, K, of a role are dependent on the goal, g, the role is bound to and its fields, in particular $g.E_g$ and $g.Q_g$. While the quality functions are useful in selecting roles that satisfy goals before agents are bound to them, the role constraints in the set K of a role specify requirements that must be met by agents performing that role when bound to a goal. For instance, a role constraint for RADARSCANNER is

$$\langle\langle ScanFrequency(SCAN), \frac{1}{SCAN.Q_g.DetectDelay}\rangle, \geq\rangle$$

which we write more succinctly as

$$ScanFrequency_{RS}(SCAN) \geq \frac{1}{Q.DetectDelay} \quad (5)$$

As shown in Figure 11, when RADARSCANNER is bound to SCAN, the role constraint for RADARSCANNER given the value of $Q.DetectDelay$ determines how often the region must be scanned to guarantee vehicle detections within the acceptable track delay. This is important because it specifies not only what individual agents must do, but also what the combined behavior of a group of agents must be if no single agent has the capabilities to perform the role individually. In other words, any coordination mechanism used to coordinate the agents performing the role jointly should ensure that the group behavior meets the requirements specified by the requirement functions.

A role constraint for FOCUSEDRADAR is as follows:

$$numRequired_{FS}(UPDATE) \quad \geq \quad 3 \tag{6}$$

This specifies that FOCUSEDRADAR for the update goal requires at least three agents working in conjunction to perform the role. The reason is that three sensor readings are necessary to triangulate the position of a vehicle. Note that this does not specify the number of agents that will ultimately be assigned to the FOCUSEDRADAR role. The assignment of agents to roles is dependent on the capabilities of the agents in the agent population. Equation 6 specifies that for the region covered by an agent playing the FOCUSEDRADAR role, at least two other agents should be assigned to that role in the same area.

In other domains, it may be the case that only a specific number of agents is allowed to be bound to a role. A requirement function could be used to specify this as well. As noted above, we do not specify an ontology that distinguishes the types of requirement functions. Rather, in the implementation of the Java class for a particular role, the developer provides code that determines if the role constraints are satisfied.

Finally, we note that KB-ORG makes a sharp distinction between roles and goals since it may be possible that multiple roles are suitable for achieving the same goal. In a distributed sensor network, for instance, both low resolution and high resolution scanning roles may be available to satisfy the goal of scanning for new vehicles. However, depending on the capabilities of the agents in the environment, the relative cost of performing each role, the drain on the resources of the agents performing the roles (which affects the agents' abilities to perform other roles), etc., one role may be more appropriate than another in a given situation. We, therefore, maintain the distinction between roles and goals since which role is bound to a goal may change from one scenario to the next.

Communication Graph In addition to specifying a goal tree and roles to satisfy the leaves, it is necessary to specify how information is to flow among goals since certain goals require information from others. We represent such relationships as a directed communication graph (Def. 13, Fig. 7).

The dotted edges between goals in Figure 5 show the edges in the communication graph for our DSN. An edge between two leaf goals indicates that when agents are bound to those goals information must be exchanged between the agents. However, the edge does not indicate whether all agents bound to one role will exchange information with all agents bound to the other or if only subsets of the agents will communicate with each other. This determination is made when the design process binds agents to the goals (see Section 3.2 for details on how agents are bound to goals).

For example, an edge exists in the communication graph in Figure 5 from SCAN to VERIFY. Assume that the RADARSCANNER and VERIFIER roles are bound to these goals respectively. Also assume that the agents in set A_1 are each bound to the RADARSCANNER role and that those in A_2 are bound to VERIFIER. Since the SCAN goal is spatial in character, not every agent in A_1 necessarily needs to send information to every agent in A_2. To represent this, the parameters of each spatially defined goal specify the area for which the goal is responsible. Initially, in the case of

SCAN, this is the complete region to be monitored. As we will see below, after the responsibilities of a role bound to a spatial goal are distributed among a set of agents, each agent bound to the role becomes responsible for a subregion of the whole. This is represented in the parameterization of the role the agent is bound to after that role is split. An agent bound to such a split role will then send information to another agent bound to a different role if an edge exists between the goals in the communication graph and the subregions specified for each agent's role overlap.

We see an example of responsibility for split roles in Figure 2 where Agent S24 is responsible for scanning in the subregion with the top-left corner at coordinates (62.5, 32.5) and a length and width of forty feet each while it is responsible for fusing data pertaining to the area with top-left corner at (45,60) and length 45 feet and width 30 feet. As part of its RADARSCANNER role, S24 sends information to Agent S22 which acts as the VERIFIER responsible for the area encompassing S24's scanning area.

One could alternatively view information as needing to flow between roles rather than goals as in work by van den Broek [38]. However, since in the KB-ORG framework multiple roles may be used to satisfy the same goal we find it more general to specify how information is to flow between goals and to let the information flow between roles be derived accordingly. Still, it is important to recognize that different roles bound to the same goal may not require the same amount of communication when they are enacted by agents. The amount of communication required by a role is specified in the agent's characterization as seen below.

Agents The characterization of an agent (Def. 14, Fig. 7) contains agent-specific features such as name and location, capabilities such as the radius of a radar's scanning area,[5] the roles the agent is able to play along with the drain on the agent's resources caused by that role and the number of messages per time the agent sends during its operational performance of the role.[6] For example, the definition of an agent for the DSN is shown below. For clarity, we do not include all roles and capabilities.

[5]In this work, we assume each agent has a finite set of capabilities and is able to acquire the resources needed to perform a role if it has the capabilities to perform that role.

[6]This work assumes a fairly coarse-grained communication model. When a role is split among a group of agents, the number of tasks performed and messages sent by each agent is known. Finer-grained models are also possible. For example, one could divide the messages sent by an agent enacting a particular split role according to the other roles the agent sends information to. Furthermore, in this work, we assume that the communication required by an agent enacting one role is independent of the other roles the agent is playing. Again, this need not be the case and other communication models could be implemented.

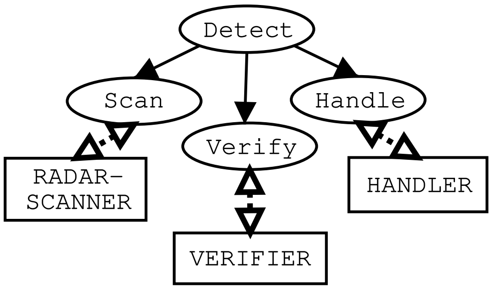


Figure 9: Subtree of the goal tree in Figure 5 with roles bound to each leaf goal.

$$\begin{aligned}
\phi &= \{\langle name, S1\rangle, \langle xloc, 7.5\rangle, \langle yloc, 7.5\rangle\} \\
CAP &= \{\langle radarRadius, 20\rangle, \langle radarScanRate, 0.25\rangle, \\
&\quad \langle focusedscanrate, 0.75\rangle, \langle comRange, 600\rangle, \ldots\} \\
\rho &= \{\langle RADARSCANNER, 0.2, \\
&\quad \frac{\pi \times CAP.radarRadius^2}{E.length \times E.width} E.maxNewArrivals\rangle, \\
&\quad \langle FOCUSEDRADAR, 0.3, \\
&\quad \frac{\pi \times CAP.radarRadius^2}{E.length \times E.width} \frac{E.maxTracks}{Q.trackUpdateInterval}\rangle, \\
&\quad \ldots\}
\end{aligned}$$

This definition shows that the agent's name is $S1$ and has an (x,y) location of (7.5, 7.5) in the rectangular region to be monitored by the DSN. It shows a subset of its capabilities and the triples for two of its roles, RADARSCANNER and FOCUSEDRADAR which require 20% and 30% of the agent's resources respectively. The number of messages per time that the agent sends in its role as a RADARSCANNER is computed by multiplying the fraction of the total area that the agent is able to "see" by the maximum number of new vehicle arrivals per time. The number of messages sent per time by the agent in its FOCUSEDRADAR role is computed by multiplying the same fraction by the maximum number of tracks in the environment and dividing it by the required time interval between track updates.

3.2 Application-Role Binding

Section 3.1 described the input that the developer provides in the configuration files of the automated designer. We next describe how the KB-ORG design process first uses that information to perform application-level bindings of roles to goals and then agents to the roles that have been bound to goals.

Definition 15 (Role-Goal Binding). *A role-goal binding, rgb, is defined by the following fields:*

- *a role $r \in R$*
- *a leaf goal $g \in L$ such that $g.TAL \subseteq r.AL$*
- *a set K' of constraints that must be satisfied. K' consists of all constraints in $r.K$ with specific values provided for all variables.*

Definition 16 (Role-Goal-Binding Set). $RGB = \{rgb_1, rgb_2, \ldots, rgb_{n_{rgb}}\}$ *is the set of application-level role-goal-bindings determined by the KB-ORG design process where* $n_{rgb} = |RGB]$.

Definition 17 (Role-Goal-Agent Binding). *A role-goal-agent binding is defined by the following fields:*

- *the role-goal binding, rgb, the agent is bound to where the fields of $rgb.g.E_g$ are modified to reflect that agent's area of responsibility.*
- *a set $toRoles = \{tr_1, \ldots, tr_n\}$ of role types that the agent sends information to in the fulfillment of its role-goal binding. $tr_i \in R$ for $1 \leq i \leq n$ where $n = |toRoles|$.*
- *a set $toAgents = \{\langle tr_1, \{a_1, \ldots a_{n_{tr_1}}\}\rangle, \ldots, \langle tr_n, \{a_1, \ldots a_{n_{tr_n}}\}\rangle\}$. $tr_i \in toRoles$, $1 \leq i \leq n$, where $n = |toRoles|$ and n_{tr_i} is the number of agents playing tr_i to which the agent sends information.*
- *a set $fromRoles = \{fr_1, \ldots, fr_n\}$ of role types that the agent sends information to in the fulfillment of its role-goal binding. $fr_i \in R$ for $1 \leq i \leq n$ where $n = |fromRoles|$.*
- *a set $fromAgents = \{\langle fr_1, \{a_1, \ldots a_{n_{fr_1}}\}\rangle, \ldots, \langle fr_n, \{a_1, \ldots a_{n_{fr_n}}\}\rangle\}$. $fr_i \in fromRoles$, $1 \leq i \leq n$, where $n = |fromRoles|$ and n_{fr_i} is the number of agents playing fr_i from which the agent receives information*
- *a Boolean flag T. If true, T indicates that the binding is a teaming assignment. Otherwise, the binding is not. Teams are described in Section 3.4.*

Figure 10: Definitions pertaining to application-level role bindings

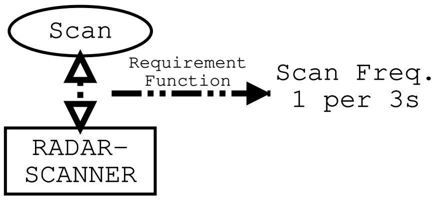


Figure 11: RADARSCANNER's requirement function generates the scan frequency requirement for the RADARSCANNER→SCAN binding.

Binding Application Roles to Goals With the input described in Section 3.1, the KB-ORG design process attempts to assign application-level roles to organizational leaf goals to form role-goal bindings (Def. 15, Fig. 10). As discussed in Section 3.1, any role whose AL contains a goal's TAL may be bound to that goal while the quality functions of roles help order and limit the search for which roles to select for goals. Figure 9 shows a subtree of the organizational goal tree in our DSN example with one role bound to each leaf goal. In this case RADARSCANNER was chosen over FOCUSEDRADAR to be bound to the goal SCAN because, as Equations 1 and 3 indicate, RADARSCANNER's quality function is higher for SCAN.

Binding a role to a goal produces constraints on the performance of the role as specified by the role's set of constraints, K (recall Definitions 10 and 11). As Equation 5 and Figure 11 show, if the RADARSCANNER→SCAN binding is instantiated, RADARSCANNER's application-level role constraint equation generates the scan frequency necessary to meet the performance requirement on new vehicle detections. For example, when RADARSCANNER is bound to SCAN under the performance requirements shown in Figure 3, the resulting role-goal binding, rg is defined by

$$rg.r = RADARSCANNER \tag{7}$$

$$rg.g = SCAN \tag{8}$$

$$rg.K' = \{\langle ScanFrequency_{rs}(SCAN) \leq \frac{1}{3}sec\rangle\} \tag{9}$$

where K' is given by Equation 5 with $Q.detectDelay = 3$ and gives the scan frequency that must be maintained.

Splitting Application-Level Roles among Agents Next the KB-ORG design process binds agents to each role-goal binding. By matching agents' capabilities to the requirements of a role-goal binding, the design process identifies agents that together satisfy the requirements of the role-goal binding, thus causing the goal to be satisfied according to Definition 7. These agents form a set of role-goal-agent bindings (Def. 17, Fig. 10) for the role-goal binding. In the DSN domain, for instance, no single agent has the capability to scan the entire region to be monitored by the DSN as specified by the RADARSCANNER→SCAN role-goal-binding since the scanning radius of each agent is only 20 feet. Therefore, KB-ORG finds agents whose combined capabilities in terms of their locations, scanning ranges, and scanning frequencies meet the requirements

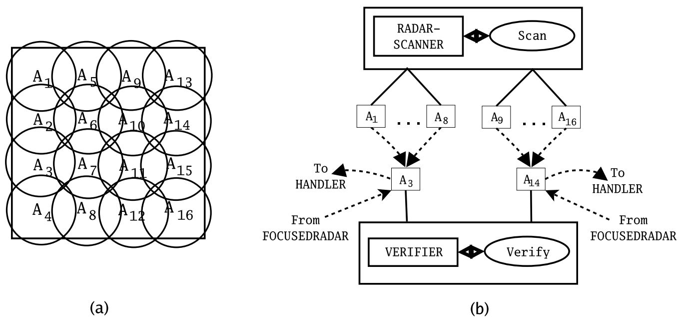


Figure 12: Role-goal-agent bindings for RADARSCANNER→SCAN and VERIFIER→VERIFY. The dotted arrows show how information flows between the bindings.

on scanning. These agents when bound to the RADARSCANNER→SCAN role-goal binding make up the set of role-goal-agent bindings for that role-goal binding.

In more detail, KB-ORG uses knowledge in the form of application-level organization design functions specified by the developer to find role-goal-agent bindings. To find role-goal-agent bindings for the RADARSCANNER→SCAN role-goal-binding in the DSN, for instance, KB-ORG uses a coverage function that finds a set of agents such that the following conditions hold if possible:

- Every point in the region to be monitored can be seen by at least 3 agents able to play the RADARSCANNER role.
- Each agent in the set above is able to scan the area it can view within the scan frequency determined by Equation 5.

When the design process identifies agents that meet the requirements of a role-goal binding, there may be multiple, alternative sets of agents able to be bound to it since many agents may have similar capabilities. As will be discussed in Section 4, the search process for an organization uses knowledge to limit the number of alternatives explored.

When KB-ORG makes a role-goal-agent binding for a particular agent, the binding specifies the role-goal binding the agent is bound to, the parameterization of the role-goal binding to indicate the agent's area of responsibility due to splitting the role, and the sets of agents it receives information from and sends information to.

Figure 2 gives an example of a single agent's role-goal-agent bindings in the DSN example. Also, consider Figure 12 which illustrates role-goal-agent bindings for the RADARSCANNER→SCAN and VERIFIER→VERIFY role-goal bindings for a set of sixteen, homogeneous sensor agents. In the distributed sensor network agents have fixed locations and can scan circular areas with fixed radii (the overlap of the sensors' viewable areas is greater than that depicted). Figure 12a shows that all sixteen agents are bound to RADARSCANNER→SCAN. If this were not the case, gaps would exist in the coverage.

Figure 12b shows that Agents 3 and 14 are also bound to VERIFIER→VERIFY in addition to RADARSCANNER→SCAN. The reason that a single agent is not bound to the VERIFIER role is that no agent among the available set has the processing capabilities both to scan for new vehicle detections and keep track of all existing tracks in order to verify new ones. This decision is made by using the agents' capabilities. As described in Section 3.1, the agent specification includes the number of messages per time an agent sends and the drain on its resources when it is bound to a particular role. In a process similar to that of binding agents to the RADARSCANNER→SCAN role-goal described above, the design process is able to use knowledge to determine the computation and communication load on agents given possible role assignments and the environmental inputs. In this case binding a single agent to the VERIFIER role would have depleted that agent's computational resources.

The reason more than two agents are not bound to the VERIFY role is that to do so would put extra load on the agents and require more communication since the agents acting as VERIFIERs would need to share information with each other about existing tracks in order to verify new detections accurately. As we will see in Sections 4 and 5, the decision to split a role in a particular way may require organization design knowledge in the form of design function in addition to an analysis of agents' capabilities. In this case, the decision to limit the number of VERIFIERs to two is an attempt to make a tradeoff between agent loading and inter-agent communication. The tradeoff decision is made assuming a simple organizational utility function that is a weighted sum of one minus the ratio of the required bandwidth to the available bandwidth and one minus the average fraction of the resource usage of each agent:

$$u \quad = \quad w_1(1 - \frac{b_r}{b_a}) + w_2(1 - load_{avg}) \tag{10}$$

where w_1 and w_2 are user-defined weights, b_r is the bandwidth required by the organization, b_a is the available bandwidth, and $load_{avg}$ is the average loading on the agents' resources. We subtract the two ratios from one to indicate that smaller amounts of communication and load are preferable within an organization. In the search process, this utility function is evaluated whenever a tradeoff decision needs to be made or partial or complete organizational candidates with role-goal-agent bindings must be compared.

The dotted arrows in Figure 12b show the application-level flow of information between the sets of bindings as determined by the communication graph in Figure 5. Each scanning agent must send detection information to a verifying agent and since scanning is a spatial activity, different VERIFIERs are responsible for the different groups of scanners. Also, after each VERIFIER determines that a detection indicates a new vehicle rather than part of an existing track, the VERIFIER must forward the detection information to an agent bound to the goal of handling new detections as specified by the edge between VERIFY and HANDLE. Also note that in order to maintain up-to-date track information, VERIFIERs receive information from agents enacting roles bound to the FUSE goal (the figure assumes FOCUSEDRADAR is bound to FUSE). Note that the dotted arrows do not specify the flow of information pertaining to the organizational coordination of the agents. While the application-level bindings do provide some level of coordination among agents, organizational coordination structures are still necessary. For instance, the application-level bindings do not give agents act-

Definition 18 (Coordination Goal). *A coordination goal cg is a goal not in the original goal tree that is generated when an application-level role is split among a set A of agents such that $|A| > 1$. cg is defined by the fields:*

- *$RGAs$ which is the set of all role-goal-agent bindings associated with the role that has been split.*
- *TAL which is as defined in Definition 3.*

Definition 19 (Coordination Role). *A coordination role is defined identically to an application-level role (see Definition 11).*

Figure 13: Definitions pertaining to coordination-level role bindings

ing as RADARSCANNERs the ability to schedule their scans in a coordinated fashion. Without such coordination, gaps in coverage could exist. Coordination bindings are made immediately after role-goal-agent bindings are made in the design process and are discussed in the next subsection.

3.3 Coordination-Role Bindings

The preceding has shown how application-level information can be used to bind roles and agents to organizational leaf goals as in Figure 1. Since, in general, a role will require multiple agents to fulfill the performance requirements of an organizational leaf goal, those agents must be coordinated in their activities. In other words, the process of determining the role-goal-agent bindings specifies the agents that will be performing roles in the organization. Only after that information is in place is it possible to determine the type of coordination mechanism to be used.

For example, all of the agents bound to the application-level role-goal binding RADARSCANNER→SCAN have the necessary capabilities to satisfy the requirements of that role-goal binding, but unless their scanning is synchronized, holes may exist in the coverage since in the DSN sensor agents have limited range and can scan in only one of three directions at a time. Similarly, the agents bound to the role-goal binding, FOCUSEDRADAR→UPDATE, have the necessary capabilities to track vehicles, but since at least three are needed to triangulate the position of any vehicle, their activities must be coordinated as well. The right coordination mechanism for these agents is dependent on many factors including the number of agents needing to be coordinated in a particular activity, the particular agents' capabilities, the performance requirements, etc.

Coordination Goal Generation Consider Figure 14 which illustrates coordination goal generation and the assignment of coordination roles for the application-level bindings in Figure 12. Because the RADARSCANNER role is split among a group of agents, as shown in Figure 14a, the agents must be coordinated in their fulfillment of that role. Note that when a role is split, the coordination required may be trivial if the agents' activities do not interact; similarly, it may be complex if their activities are

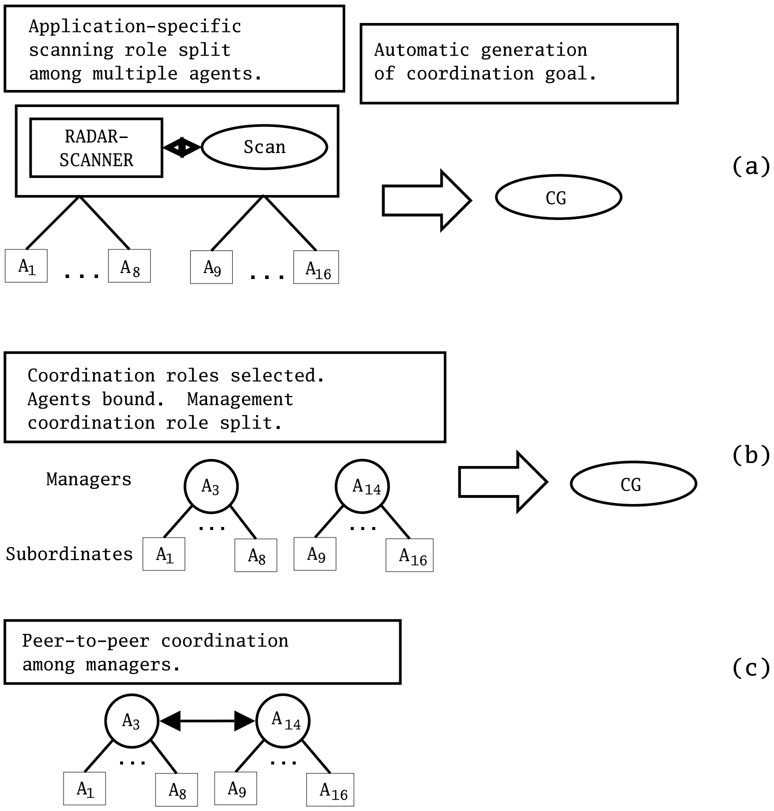


Figure 14: Illustration of coordination goal generation and the assignment of coordination roles. (a) The RADARSCANNER role is split among a group of agents which automatically causes the system to generate a new coordination goal. (b) In this example the system chooses a one-level hierarchy to satisfy the new goal, but since the management role must also be split among a group of agents, it generates yet another coordination goal which (c) in this example is satisfied by a peer-to-peer structure.

highly interdependent. Regardless, the splitting of a role among a set of agents automatically triggers the system to generate a new *coordination goal* (Def. 18, Fig. 13) that was not part of the original goal decomposition.

Coordination Roles After a coordination goal is generated, it must be fulfilled by coordination roles (Def. 19, Fig. 13), as shown on the right of Figure 1. For the DSN work described in this article, we have developed four coordination roles. These include:

- a peer-to-peer role for negotiation of scan schedules,
- a low-level manager role in which an agent develops the scan schedule for a group of scanners,
- a subordinate role to be managed by low-level managers, and
- a higher-level manager role for coordinating the actions of lower-level managers. If higher-level managers are unnecessary, the lower-level managers coordinate in a peer-to-peer fashion.

Note that the process of finding coordination domain bindings is not specific to these four roles. The process generalizes to other coordination roles such as federations and holarchies and others described by Horling [18, 17]. In the case of federations, for example, one would need to develop a federation member role along with one or more intermediary roles.

KB-ORG utilizes the same mechanism for binding coordination roles to coordination goals as it does for application-level roles and goals. When a coordination goal is formed, the system compares the quality functions of each coordination role when bound to that goal. In our current implementation, the quality function of each coordination role we have developed depends on the size of the role-goal-agents field in the associated coordination goal. The reason is that the primary distinctions between the coordination roles arise from the differing amounts of communication between agents and the computational resources required within agents when bound to the coordination roles. For instance, the characterization of the Manager role assumes that the number of messages necessary to coordinate the activities of Subordinates is $O(n)$, whereas the characterization of the peer-to-peer coordination role assumes $O(n^2)$ messages. Thus, under most scenarios the quality function of peer-to-peer roles is lower than that of managerial roles.

Splitting Coordination Role-Goal Bindings A coordination role-goal-binding can, itself, require a set of agents to satisfy it causing the creation of another higher-level coordination goal. This is illustrated in Figure 14 in which the RADARSCANNER application-level role is first split among a set of agents. This automatically causes the creation of a coordination goal. In Figure 14b, a one-level hierarchy is chosen to satisfy the coordination goal, but since since the management role is also split among a set of agents, another coordination goal is generated. In Figure 14c, a peer-to-peer mechanism is chosen to satisfy the new coordination goal because with only two managers present this represents the choice with highest quality. In other situations, another level of hierarchy may have been chosen resulting in a multi-level hierarchy of sensing, middle-manager, and overall manager roles as seen in Figure 19.

In deciding which agents should act as managers (or other coordination roles), the design process takes the same steps that it uses to bind agents to application-level roles. The process considers the other roles each agent plays and the relative utility of assigning managerial responsibilities to them as given by Equation 10. In Figure 14b, Agents 3 and 14, which are also VERIFIER agents are chosen to be the managers. This choice was made to minimize the amount of communication by multiplexing verifying and managing responsibilities within the same agent. As will be seen in Section 5, if balancing agent load is more important than communication usage, it may be better to bind other agents to the managerial role.

3.4 Designing Organizations to Facilitate Teaming

The role-goal-agent bindings and their parameters specify the long-term structure of the designed organization. Although such bindings are appropriate for long-term organizational goals, more transient goals are better satisfied by teams [13, 37, 33, 35, 25, 2]. Teams, coalitions, and congregations are temporary structures that form to satisfy particular tasks that enter the environment and disband when the tasks are completed. In the DSN, tracking a vehicle might be done by a team whose membership changes as the vehicle moves. Teams are not strictly part of the organizational structure; the assignment of agents to roles associated with the team will be shorter lived than the assignment of agents to roles to satisfy organizational goals. However, teams are not purely operational either, as sufficient resources must be set aside organizationally to allow for generating and participating in teams. Furthermore, when an agent within an organization is participating in a team, its team activities will impact its performance in its other roles. Therefore, the organizational structure must be prepared for team activity by its members.

KB-ORG does not generate teams since team generation occurs operationally in response to dynamic events. Rather, it ensures that appropriate organizational structures and resources are reserved to form teams as needed. Consider the goal TRACK in Figure 5. Satisfying this goal requires that each new vehicle that enters the monitored area be tracked. Due to the uncertainty in vehicle movements, which agents will be involved in tracking any particular vehicle or when agents will be called upon to track cannot be known at design time. Therefore, under most circumstances TRACK is best satisfied by teams. For the DSN example, this means finding role-goal-agent bindings for the leaf goals of TRACK and setting the team flag T to true to indicate that agents participate in the role only as needed.

It is possible that for a given domain, teaming may be the right choice for a goal in some circumstances while longer-term organization coordination mechanisms may be right in others. This could happen in the DSN example, for instance, if the frequency of new vehicle arrivals was high enough that agents' performance of tracking roles was predictable and regular over the lifetime of the organization. In that case organization coordination mechanisms instead of teaming mechanisms could be put in place. This might include a manager or managers responsible for coordinating the activity of tracking agents to handle sensor resource contention that arises when individual agents are requested to assist in tracking multiple vehicles.

A team role resembles an organizational role in that the agent with a team-role will have an expected number and frequency of messages to send and amount of work to do for that role. The agents bound to these roles, however, will only be expected to perform those activities if and when they are called upon to join a team. We must also specify appropriate coordination roles in order to enable teams to form. In this work, we define a coordination role, TEAMINITIATOR, that is responsible for generating teams operationally.

As mentioned in Section 3.2, Figure 2 shows a set of bindings for an agent in the DSN. Each binding specifies which organizational subgoal the agent is bound to and the agents to which it sends information and those from which it receives information. If the role is a teaming assignment such as FUSER→FUSE, it is annotated with a superscript T.

4 Knowledge-Based Search and Design Suitability

The application-level and coordination-level sides of the KB-ORG process interact through a search process since the choice of agents for organizational goals and the organizational structures to coordinate their actions depend on one another. Consider an agent in a DSN that was originally chosen to scan for vehicles. If that agent is later chosen to manage other scanners, it may no longer have the resources necessary to scan for vehicles itself. Also, in general multiple roles can satisfy the same organizational subgoal, many agents can be bound to a role-goal binding, and each agent can play multiple roles, making it computationally infeasible to generate every binding.

KB-ORG employs a best-first search process that uses organization-design knowledge and the utility function of Equation 10 to generate efficiently a reasonable set of bindings. For the application-level portions of the design process, KB-ORG uses knowledge provided in the quality and requirement functions of roles, the capabilities of the agents, and the organization design functions. For example, to evaluate and compare sets of role-goal bindings the system finds the average value of the roles' quality functions given the goals they are bound to. Consider Figure 9 once more. It shows a subtree of the organizational goal tree with roles bound to each leaf goal. In an alternate set of role-goal bindings, FOCUSEDRADAR, not RADARSCANNER, could be bound to SCAN because FOCUSEDRADAR also contains *scanning* in its assignable list (see Figure 8). The original set of bindings, however, will have a higher average quality than the other, since RADARSCANNER's quality function will have higher value than FOCUSEDRADAR's when bound to scan. This causes the search process to explore organizational candidates with the role-goal bindings shown in Figure 9 before considering candidates that use the other bindings.

Algorithm 1 provides pseudocode for KB-ORG's search algorithm. The search explores the space of organizational candidates. To understand an organizational candidate, consider the formal definition of an organization (Def. 20, Fig. 15) shown in Figure 15. An organizational candidate is an organization $\mathcal{O}$ which contains $\mathcal{O}.gt$,

Definition 20 (Organization). *An organization $\mathcal{O}$ within the KB-ORG framework is defined by the following fields:*

- *a goal tree, gt, with leaf goals L,*
- *environmental inputs, E,*
- *a set, Q, of performance requirements,*
- *a set, R, of application-level roles available to the organization,*
- *a communication graph, COM,*
- *a set, A, of agent descriptions,*
- *a set, RGB, of both application-level and coordination-level role-goal bindings,*
- *a set, $RGAB$, of both application-level and coordination-level role-goal-agent bindings,*
- *a set, CR, of coordination roles available to the organization*

Figure 15: Formal definition of organization

$\mathcal{O}.E$, $\mathcal{O}.Q$, $\mathcal{O}.R$, $\mathcal{O}.COM$, $\mathcal{O}.A$, and $\mathcal{O}.CR$ as they are provided by the developer, but in which at least some of the other fields are empty. In other words a candidate organization is an organization in which all required bindings are not yet made. A complete organization, on the other hand, is one in which all required bindings have been made.

Algorithm 1 takes an ordered list $\mathcal{C}$ of organization candidates as input. At the start of the search, $\mathcal{C}$, contains no elements. In response, the algorithm creates an empty candidate organization c and pushes it onto $\mathcal{C}$. Algorithm 1 then recursively calls itself on $\mathcal{C}$.

When $\mathcal{C}$ contains at least one element, a depth first search for an organization with a complete set of bindings begins. The loop beginning on line 8 of Algorithm 1 starts with the most highly rated candidate in $\mathcal{C}$ and calls $expandCandidate$ shown in Algorithm 2 to generate a set of successor candidate organizations. We discuss $expandCandidate$ in detail shortly. If the list $\mathcal{E}$ of successors that $expandCandidate$ returns is not empty, $\mathcal{E}$ is sorted on line 11.

In the sort, successors are compared according to the utility function of Algorithm 4 which returns a value in one of three ways depending on the type of candidate being evaluated. If the candidate being evaluated has role-goal bindings but no role-goal-agent bindings (type 1), Algorithm 4 returns the average value of the quality functions of all bound roles in the candidate. If there are role-goal-agent bindings for all role-goal bindings (type 2), Algorithm 4 returns a value given by Equation 10. If there is at least one role-goal binding that does not have role-goal-agent bindings associated with it (type 3) which happens after role-goal bindings for a newly generated coordination goal are made, Algorithm 4 returns the sum of the two values. Note that Algorithm 1

Algorithm 1 organizationSearch

```
INPUT: An ordered list 𝒞 of candidate partial organizations
RETURNS: An organization that meets the requirements or nil if one cannot be found.
if 𝒞 = ∅ then
   c ← an empty candidate organization.
   push c onto 𝒞
   return organizationSearch(𝒞)
else
   for i = 0 to C.length − 1 do
      ℰ ← expandCandidate(𝒞[i])
      if ℰ ≠ ∅ then
         sort ℰ according to utility function in Algorithm 4
         if ℰ[0] is a complete organization then
            return ℰ[0]
         else
            return organizationSearch(ℰ)
         end if
      end if
   end for
   return nil
end if
```

is structured such that only the utility of candidates of the same type will be compared. Returning to Algorithm 1, after the sort is complete, if the most highly rated candidate, $\mathcal{E}[0]$, has all requisite bindings, that organization is returned. Otherwise Algorithm 1 is again called recursively on $\mathcal{E}$.

In short, Algorithm 1 first expands the initial empty candidate to generate an ordered list of candidates with application-level role-goal bindings. It then expands the most highly rated of these to generate another ordered list of candidates with application-level role-goal-agent bindings for every application-level role-goal binding. Next, it expands the most highly rated of these to generate an ordered list that has coordination role-goal bindings for every set of role-goal-agent bindings that consists of more than one agent. The process continues to generate a list of candidates with coordination-role-goal-agent bindings until it finds a candidate with all requisite bindings. If at any point along the way, the search cannot expand a candidate, either the next highest rated sibling candidate is expanded or the next candidate in the previous level of recursion is expanded. This continues either until one or more complete organization candidates are found or until the algorithm determines that no candidates that meet the requirements can be found. This provides a level of backtracking to account for the fact that initial binding choices may lead to a candidate organization in which no agent given its current set of roles and capabilities can satisfy the remaining responsibilities. For instance, although the quality functions indicate that RADARSCANNER is the better role to bind to the goal SCAN, in some instances this may not be the right choice. Although one role may initially appear best when binding roles to goals, other factors may ultimately cause another role to be chosen. The agents that are able to perform the highest quality role may have their resources devoted to performing other, more important roles. Still, even with this possibility, our intuition is that the use of quality functions results in searches that are quicker than exhaustive searching since as Horling showed searching for an organization is NEXP-complete [17].

Algorithm 2 expandCandidate

1: **INPUT:** c a partial organization candidate
2: **RETURNS:** a list, $\mathcal{L}$, of partial or complete successor organization candidates
3: $\mathcal{L} \leftarrow \emptyset$
4: **if** c does not have role-goal-bindings for all leaf goals in its goal tree **then**
5:     {Generate a successor candidate for every possible combination of role-goal-bindings for the leaf goals without role-goal bindings}
6:     $\mathcal{L} \leftarrow generateAllRoleGoalBindings(c)$
7: **else if** c does not have role-goal-agent-bindings for all role-goal-bindings **then**
8:     {Generate a limited number of candidates with role-goal-agent bindings.}
9:     $\mathcal{L} \leftarrow generateRoleGoalAgentBindings(c)$
10:     {$generateRoleGoalAgentBindings(c)$ generates coordination goals for each role-goal-binding split among more than one agent.}
11: **end if**
12: return $\mathcal{L}$

Algorithm 3 generateRoleGoalAgentBindings

INPUT: c a partial organization candidate with role-goal bindings for every leaf goal in the goal tree
RETURNS: a list $\mathcal{L}$, of successor candidates with role-goal-agent bindings for each role-goal binding.
for all $rg \in c.rg$ **do**
    Utilize knowledge (see Table 1) to populate $\mathcal{L}$ with successors that contain role-goal-agent bindings.
    For each role-goal-agent binding generated, if it consists of more than one agent, add a coordination goal associated with it to the goal tree.
end for

We now return to the details of $expandcandidate$ in Algorithm 2. This function behaves like a finite state machine. If the candidate provided as input does not have role-goal bindings for all leaf goals in its goal tree (line 4), $expandCandidate$ generates a successor candidate for every possible combination of role-goal bindings for the unbound leaf goals and adds these successor candidates to the list $\mathcal{L}$. The pseudocode for the function that does this, $generateAllRoleGoalBindings$, is not shown because the process is obvious.

If the candidate organization provided to $expandCandidate$ does have role-goal bindings for every leaf goal, but does not have role-goal-agent bindings for every role-goal binding (line 7), $expandCandidate$ calls Algorithm 3 to set $\mathcal{L}$ to a list of candidate organizations that have role-goal-agent bindings for every role-goal-binding. Note

Algorithm 4 utility

INPUT: a candidate organization c such that $c.rg \neq \emptyset$
if there are role-goal bindings but no role-goal-agent bindings **then**
  return
$$\frac{\sum_{rg \in c.RBG} rg.qf}{|c.RGB|}$$
else if there is a role-goal-agent binding for every role-goal binding **then**
  return $c.util$ where $c.util$ is given by Equation 10
else
  return
$$\frac{\sum_{rg \in c.rg} rg.qf}{|c.rg|} + c.util$$
  where $c.util$ is given by Equation 10
end if

that to generate a successor for every possible combination of role-goal-agent bindings would be computationally infeasible. To combat this Algorithm 3 utilizes knowledge in the organization design functions to limit the number of successor candidates when it populates the list $\mathcal{L}$ in line 4 of Algorithm 3. As discussed in Section 3.2, for example, when $generateRoleGoalAgentBindings$ encounters the RADARSCANNER→SCAN role-goal binding, it utilizes the coverage function to find a single set of agents such that the following conditions hold:

- Every point in the region to be monitored can be seen by at least 3 agents able to play the RADARSCANNER role.
- Each agent in the set above is able to scan the area it can view within the scan frequency determined by Equation 5.

Algorithm 3 uses similar knowledge for the other application-level roles. In addition, it uses knowledge based on communication load and spatial proximity as well as knowledge of how to multiplex roles. We discuss these below after discussing how Algorithm 3 generates coordination goals.

When $generateRoleGoalAgentBindings$ generates a successor with role-goal-agent bindings for each role-goal binding, it also generates a coordination goal for each role that is split among more than one agent. In other words, if the set of role-goal-agent bindings for a particular role-goal binding contains more than one agent, $generateRoleGoalAgentBindings$ adds a new coordination goal for that set of bindings to the goal tree. Therefore, when $expandCandidate$ is called on a candidate organization with role-goal-agent bindings, but no coordination bindings, the condition on line 4 of Algorithm 2 is true which causes coordination role-goal bindings to be generated. Similarly, on the next call to $expandCandidate$ the condition in line 7 will hold and role-goal-agent bindings for coordination roles will be generated by $generateRoleGoalAgentBindings$. This process continues until no more coordination goals are generated.

Knowledge Source	Purpose
Quality Functions	Reducing number of role-goal bindings explored
Utility Function	Tradeoff decisions and org. candidate evaluation
Role-Specific ODFs	Limit the number of role-goal-agent bindings.
Message Multiplexing ODF	Reducing perceived communication load of an agent
Spatial proximity ODF	Breaking ties between role-goal-agent bindings
Time to Perform ODF	Making decisions about levels of hierarchy

Table 1: Summary of the knowledge used in this work. ODF stands for organization design function.

With the exception of the quality functions which are used by the sort function, $generateRoleGoalAgentBindings$ uses the knowledge summarized in Table 1 most of which are organization design functions (ODFs).[7] The utility function, given by Equation 10 is a weighted sum of one minus the ratio of the required bandwidth to the available bandwidth and one minus the average fraction of the resource usage of each agent. To evaluate the first term of Equation 10, the search process determines the ratio of the average bandwidth required by the agents to perform their roles as specified in their capabilities to the available bandwidth specified in the environmental inputs. To evaluate the second term the algorithm computes the average load on the agents based on the resource drain of each role on each agent as specified in their capabilities.

To see how the search uses knowledge of how to multiplex messages, consider the DSN example once again. As we saw in Section 3.3, it is possible to assign the Manager role to an agent acting as both VERIFIER and RADARSCANNER or to an agent acting solely as a RADARSCANNER. If the Manager role is multiplexed within the same agent as the VERIFIER role and the Manager and VERIFIER are responsible for the same agents, the knowledge assumes that the agent is able to combine verifying and managing messages to reduce the total bandwidth it requires. If the Manager role were assigned to an agent acting solely as a RADARSCANNER, the search process assumes that the agent would have to send management messages in addition to its scanning messages and the verifying messages of the VERIFIER. Thus, the former would have a smaller combined communication load than the latter.[8]

Also related to messaging is the knowledge in KB-ORG that it is better for agents who must communicate to be spatially near one another. This is helpful in deciding among alternative role-goal-agent binding sets for a particular role-goal binding.

Another type of knowledge used in making coordination decisions has to do with the amount of time available to perform a task as specified in the performance requirements. If the time is small, the system is more likely to add a level of hierarchy to an

[7] Although more principled and more domain-independent knowledge than that in Table 1 would be ideal, we note that mechanisms similar to those in Table 1 for structuring organizations are common in the organization design literature [16].

[8] Deciding which message types can and cannot be piggybacked is complicated. It involves an analysis of when each message type will be sent and to whom. The coarse-grained communication model in our current work does not support such a detailed analysis. Instead, we use a simple model that if two roles within an agent need to send messages to the same set of agents the messages from those two roles can be combined to reduce the total communication requirements.

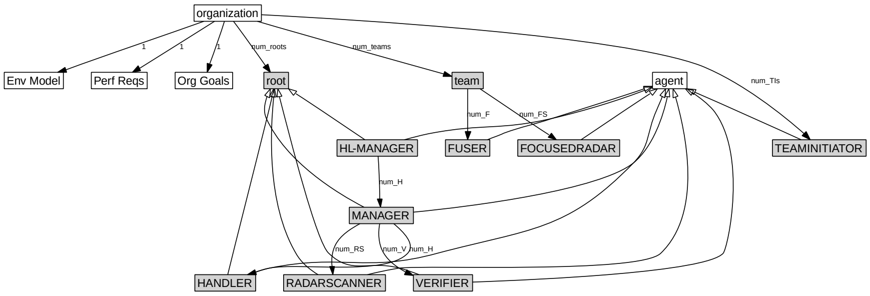


Figure 16: An ODML template [17] for the environmental inputs, requirements, goals, and roles presented in this work.

existing hierarchy. The assumption is that because high-level managers have greater context, they can notice and correct for inefficiencies not noticed by lower-level managers and may be worth the overhead of a more complicated structure. This will be seen in Section 5 for the DSN when the acceptable delay on new vehicle detections is small.

Although utilizing knowledge above should lead to an organization that meets the performance requirements, it does not give enough information to rank every candidate organization that satisfies all the requirements. We must consider other factors to evaluate them. For that it is important to have an organizational evaluation function that is based on user-specified criteria to determine a particular candidate's utility. In on-going work, we are developing a detailed evaluation capability to evaluate fully specified organizations and to prune the search through partially complete ones. For now, we rely on simple the utility criteria of Equation 10. We also note that the search process described in this section is not optimal since it takes a greedy approach to finding an organization.

Comparison to ODML In contrast to Algorithm 1, consider the search process in Horling's work on using ODML for organizational design [17]. Figure 16 shows an ODML representation of an organizational template using the application-level and coordination roles, environmental inputs, goals, and performance requirements in our work. In the figure the arrows with hollow tips represent *is-a* relationships and the arrows with filled tips represent *has-a* relationships. The template has a set of roots at the highest level, specified by the root node and the *num_roots* label on the *has-a* relationship between organization and root. A root represents the highest-level organizational entity. Because we allow peer-to-peer coordination among application-level roles, RADARSCANNERs, VERIFIERs, and HANDLERs all have *is-a* relationships with root. However, since those roles could be managed by MANAGERs, the MANAGER role also has an *is-a* relationship with root as well as *has-a* relationships with the application-level roles.[9] This shows that the exhaustive search would have to explore

[9]We have not included the SUBORDINATE role in the ODML representation. That role is assumed in the *has-a* relationships of MANAGER.

the options of having MANAGERs with subordinates or peer-to-peer coordination. Similarly, the HL-MANAGER (high-level manager) coordination role also has an *is-a* relationship with root and a *has-a* relationship with MANAGER. Every role has an *is-a* relationship with agent.

For clarity Figure 16 does not show the various relationships between the environmental inputs, requirements, goals, and roles. It also assumes that the FUSER and FOCUSEDRADAR roles will be teaming roles. Despite this, the figure shows that the exhaustive search in Horling's work would be computationally expensive. It would explore organizational candidates with every possible role combination.[10] Our work limits this search by exploring particular application and coordination role combinations only as required. As described above, it uses the quality functions to explore the use of RADARSCANNERs rather than FOCUSEDRADARs for the scan goal, and only explores using FOCUSEDRADARs if no feasible role-goal-agent bindings exist for RADARSCANNER. Similarly, the search through the space of coordination roles is triggered only when a role is split.

5 KB-ORG Designs for Different Conditions

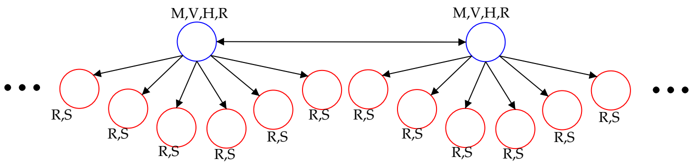
We present four example organization designs generated by KB-ORG on the goal tree and communication graph in Figure 5, the parameters in Figure 3, and the roles in Figure 8. We varied the input along several dimensions: size of the area monitored, number of agents, value of the acceptable detect delay,[11] and the relative costs of communication and agent load as specified in the weights in Equation 10. In all cases the agents we used were evenly spaced throughout the region, each with identical features, roles they can be bound to, and capabilities.[12] Figures 17-20 summarize the results. Each organization in the figures took only a few seconds to generate because the knowledge used the search process substantially pruned the search space.

The first scenario shown in Figure 17 involved 36 agents in a $90' \times 90'$ rectangular area, an acceptable detect delay of 3 seconds, and the cost of communication greater than that of agent loading. The result was a single-level hierarchy with 6 managers each managing 6 agents. The managers coordinated among themselves using a peer-to-peer mechanism. In order to minimize communication, there were 6 verifying and handling roles each multiplexed within the same agents as the managing roles. This organization corresponds closely to the hand-crafted structure used for the EW Challenge Problem [20] where communication cost was a major concern. The performance of this organizational form relative to others was recently tested experimentally [19, 21]. Also, in this scenario and the others, the FUSER and FOCUSEDRADAR roles were set as team roles with the TEAMINITIATOR role distributed among the HANDLER agents.

[10] We have not run Horling's ODML code on this template because our abstraction of agents' capabilities and our representation of the interactions among agents and roles do not correspond to how Horling represents this information. In continuing work, we are attempting to modify our representation to make it compatible with Horling's.

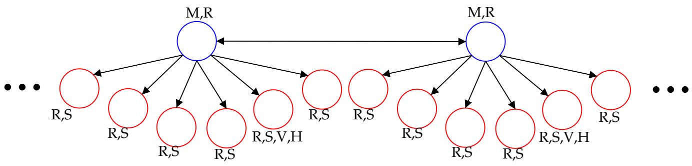
[11] Recall that the detection delay is used in the requirement function for RADARSCANNER shown in Equation 5.

[12] The homogeneity of the agents is a characteristic of the DSN problem [20] we are modeling and is not a requirement of the design process.



Agents	Area	Delay	Com. Cost	Load Cost
36	$90' \times 90'$	$3s$	0.6	0.4
Single-level hierarchy: 6 Managers. Verifier and Handler roles multiplexed within same agent as Manager. Managers coordinate peer-to-peer.				

Figure 17: Example organization with the cost of communication greater than that of agent load. The labels in the figures refer to the roles present in the organization. M stands for manager, S for subordinate, V for verifier, H for handler, and R for radar scanner.

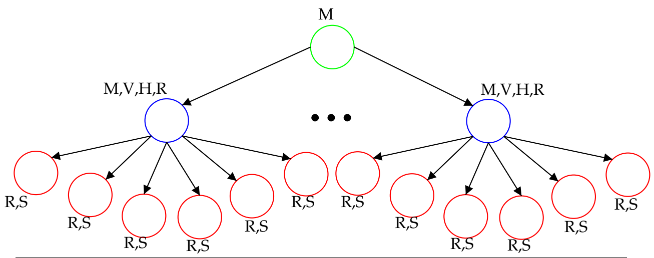


Agents	Area	Delay	Com. Cost	Load Cost
36	$90' \times 90'$	$3s$	0.4	0.6
Single-level hierarchy: 6 Managers. Verifier and Handler roles not multiplexed with Manager. Managers coordinate peer-to-peer.				

Figure 18: Example organization with the cost of communication less than that of agent load. The labels in the figures refer to the roles present in the organization. M stands for manager, S subordinate, V verifier, H handler, and R radar scanner.

Switching the relative costs of communication and load still resulted in a single-level hierarchy as shown in Figure 18, but the verifying and handling roles were no longer multiplexed within managers. They were distributed to separate agents to minimize load. In effect because communication was inexpensive, the utility given by Equation 10 of organizational candidates with higher communication usage was relatively higher than those in the previous scenario. Therefore, the organization could afford to use more communication in order to balance the computational load among the agents.

For the third scenario, we used the same costs as in the first, but reduced the acceptable track delay to 2 seconds. This time the generated organization was a two-level hierarchy with 6 mid-level managers and 1 upper-level manager to coordinate them as shown in Figure 19. At first this may seem counter-intuitive since increasing the level of hierarchy can often introduce delays. However, in this problem with a small acceptable delay on new detections, it is critical that the scanning agents have tightly



Agents	Area	Delay	Com. Cost	Load Cost
36	$90' \times 90'$	$2s$	0.6	0.4
Two-level hierarchy: 6 mid-level Managers. Verifier and handler roles multiplexed within mid-level Managers. One upper-level Manager to coordinate mid-level Managers.				

Figure 19: Example organization with reduced acceptable track delay. The labels in the figures refer to the roles present in the organization. M stands for manager, S subordinate, V verifier, H handler, and R radar scanner.

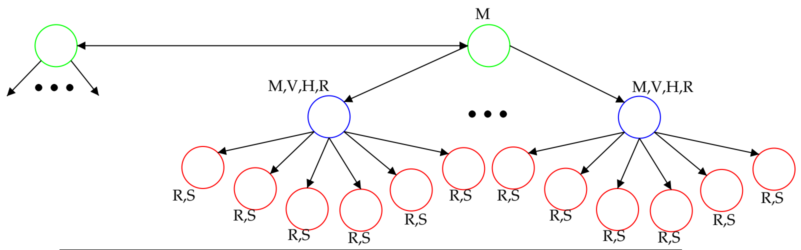
synchronized scan-schedules. Because producing a shared scan-schedule can be done in advance of detection activities, the design system added a second level of hierarchy in order to resolve scan-schedule conflicts among the managers in a centralized fashion.

In the last scenario, the parameters were also the same as in the first run except that we increased the number of agents to 100 and the size of the region to $150' \times 150'$. In this case the system generated another two-level hierarchy as seen in Figure 20 this time with nine managers and two upper-level managers which coordinate using a peer-to-peer mechanism. The extra-level was added since to coordinate the nine managers in a peer-to-peer fashion would have incurred greater communication overhead.

We are pleased that our design system produced such appropriately different organizational forms given only changes to the environmental characteristics and performance requirements. These results confirm for us the usefulness of our approach in generating organizational forms without a pre-specified organizational structure or explicit enumeration by the designer of all possible organizational forms as in Horling's work [17].

6 Relation to Other Work

To date, most explicitly designed multi-agent organizational structures have been hand-crafted, sometimes assisted by automated template expansion [32] or computed adjustments made to a pre-determined structure [31]. Other multi-agent organizational design work has focused on providing frameworks or methodologies aimed at assisting the hand-design of organizations and the adherence of organizational constraints [41, 12, 38, 9, 42, 29, 24], possibly using simulation to assist the process [14]. Still other work views the organizational design problem from a game theoretic perspective [1]. Past work in multi-agent organizational design has also been purely descriptive, such as or-



Agents	Area	Delay	Com. Cost	Load Cost
100	$150' \times 150'$	$3s$	0.6	0.4
Two-level hierarchy: 9 mid-level Managers. Verifier and handler roles multiplexed within mid-level Managers. Two upper-level Managers to coordinate mid-level Managers. Upper-level Managers coordinate peer-to-peer.				

Figure 20: Example organization increased number of agents and size of the monitored region. The labels in the figures refer to the roles present in the organization. M stands for manager, S subordinate, V verifier, H handler, and R radar scanner.

ganizational ontologies [13], has used predetermined organizational forms [32], or has focused on specifying a specific organizational design, not searching for one [11, 22].

The work of So and Durfee [31, 30] and that of Horling [17] come closest to ours. With a model based on the task environment, organizational structure, and performance metrics, So and Durfee explored how to choose an organizational structure for a given problem. However, they took a generate-and-test approach to searching for an organization rather than a knowledge-intensive approach to making organization design decisions. They were also primarily concerned with making span-of-control decisions in a hierarchical coordination. While the example organizations we provide in this article are also hierarchical, it would be possible for KB-ORG to find other organization forms if it were provided with the appropriate coordination roles. As part of continuing work, we will be generating such roles. Horling's work on using ODML for organizational design differs from ours primarily in that it does not take a knowledge-based approach to reducing the amount of search required to find an organization design. Rather Horling's work takes a generate and test approach. The techniques that Horling does use to bound the search where possible include value trend estimates and equivalence classes. They do not utilize organization design knowledge. Furthermore, because Horling's work does not distinguish between application-level and coordination-level roles, ODML's search process cannot recognize when a coordination role is necessary as KB-ORG does.

Other multi-agent work has dealt with agent coordination but emphasized operational over organizational issues. STEAM [37], for example, provides a hierarchical, role-based framework for the quick formation of agent teams and coordination between them. Similarly, GPGP [26, 6] provides a family of coordination mechanisms for dynamically organizing agents.

Several approaches to organizing or reorganizing large groups of agents utilize emergent or bottom-up techniques [40, 39, 34, 36, 23] for self-organization. While there are certainly situations in which such methods are appropriate, time constraints may not allow the self-organization processes to unfold. Also, the quality of an emergent organization may be less than that of a carefully designed one [15, 4].

Dastani's model for matching agents to roles based on the goals they can achieve [5] has some similarities to ours. However, there are a number of important differences. That work is aimed at enabling agents to enact roles as they enter open agent societies. We are interested in assigning agents to roles so that they may function together as a coherent organization. Furthermore, there are definitional differences that make the process of matching agents to roles quite different from ours. For instance, in Dastani's model, agents come to a society with goals of their own while the roles of the society also have goals. Matching an agent to a role, therefore, is done based on whether or not the agent's goals are consistent with the role's goals. In KB-ORG, matching an agent to a role is done based on the requirements of the role and the agent's capabilities.

Work by other researchers makes distinctions similar to ours between application-level and coordination-level knowledge. For instance, in OMNI [42] roles, relationships, and capabilities are linked to domain-specific norms and ontologies while more domain-independent architectural templates are used to provide the basis of coordinating the activities of the agents playing the domain-dependent roles. In other work, Dignum and Dignum [7, 8] extend the work of OMNI by analyzing the cost of each type of coordination template for a domain. A major difference between OMNI and the work described in this article is that in OMNI the choice of organization coordination structure is made by the human designer based on a qualitative analysis of the domain characteristics whereas in this article the choice of organization coordination mechanism is made by the automated system based on an quantitative evaluation of the domain characteristics. This highlights another important distinction between OMNI and the work in this article. OMNI is aimed at taking a human designer through the steps of designing an organization. The emphasis in our work is on an automated system for multi-agent organization design.

The work of Zambonelli et al. on the GAIA methodology [43] also contains ideas related to the separation of application-level and organizational coordination-level knowledge. For instance, in GAIA the purpose of the analysis phase is to determine how the organization is to function. This involves collecting the specifications, and requirements of the organization as well as identifying the skills and basic interactions required by the organization. The analysis phase ends with identifying the constraints that the organization will have to respect, but it does not decide which organization structure is appropriate for the domain. That occurs in the design phase. As with much of the related work, their work is aimed at assisting the hand-design of organizations.

7 Summary and Future Work

KB-ORG's knowledge-intensive organization design process has great promise. Its major contribution to the field of multi-agent organization design is its knowledgeable exploration and pruning of the organization design search space. KB-ORG achieves this by providing a system that implements a search algorithm that is able to use both application-level and coordination-level knowledge provided to it. By alternating between application-level knowledge and coordination-level knowledge in its search process, KB-ORG first finds agents suitable for organizational goals and then generates appropriate mechanisms to coordinate them. Both types of knowledge enable KB-ORG to limit search. The application-level knowledge enables it to select roles for organizational goals and agents for the chosen roles without exploring all combinations. The coordination-level knowledge allows KB-ORG to make informed coordination strategy decisions while also avoiding exploring coordination bindings where they are not needed.

We have demonstrated that KB-ORG is able to design effective organizations of different forms by varying performance requirements and environmental characteristics. This work also shows that delaying the use of coordination-level knowledge until application-level agent bindings have been made is a useful approach to multi-agent organization design. Furthermore, although the examples in this article have all involved a specific distributed sensor network, the approach is general.

We see a number of directions for extending the KB-ORG approach. We divide these into two categories: research directions and additional evaluation. Research directions include:

- Augmenting KB-ORG's internal evaluation capabilities to support additional criteria provided by the developer. In addition to the current load and communication criteria, these could include responsiveness to changes in the environment and tasks, agent failure predictions, etc.

- Providing KB-ORG with the ability to suggest what additional resources and agent capabilities, if they were provided, would have enabled a better performing organization.

- Enabling KB-ORG to design organizations with time-varying organizational requirements and environmental expectations

- Compiling a library of coordination mechanisms. It is our intuition that coordination knowledge often transcends applications. Such a KB-ORG library could include significant, generic organization coordination knowledge, requiring little or no domain-specific coordination knowledge to be specified.

- Including the possibility of task, communication, and agent failure as part of the expected organizational environment

Additional evaluation directions include:

- Designing a greater variety of coordination roles for the DSN to increase the types of organizations KB-ORG is able to consider for it

- Building an external mechanism for evaluating the organization designs produced by KB-ORG. Such a mechanism must include detailed analysis of an organization's performance and simulation results, and will be especially useful in assessing designs as KB-ORG is applied to new application domains.

Our long-term goal is to incorporate a future version of KB-ORG into a larger multi-agent organization design, instantiation, maintenance, and redesign system. The KB-ORG component will provide an organization design for a group of agents and distribute the design to them. The agents will instantiate the organization and provide feedback to KB-ORG as the environment, requirements, tasks, and agent capabilities change. Using this feedback as revised input, KB-ORG will provide either a modified design or a brand new one. Ultimately, therefore, KB-ORG will play the important ongoing role of designing and maintaining the organization throughout its lifetime.

References

[1] Guido Boella and Leendert van der Torre. Organizations in artificial social systems. In *Proceedings of AAMAS05 Workshop, From Organizations to Organization Oriented Programming in MAS*, 2005.

[2] Christopher H. Brooks and Edmund H. Durfee. Congregation formation in multiagent systems. *Journal of Autonomous Agents and Multiagent Systems*, 7:145–170, 2003.

[3] Daniel D. Corkill and Susan E. Lander. Agent organizations. *Object Magazine*, 8(4):41–47, April 1998. (An extended version of this article was published as "Diversity in Agent Organizations," technical report, Blackboard Technology, Amherst, Massachusetts, 1998.).

[4] Daniel David Corkill. *A Framework for Organizational Self-Design in Distributed Problem-Solving Networks*. PhD thesis, University of Massachusetts, Amherst, Massachusetts 01003, February 1983.

[5] 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.

[6] K. Decker and V. Lesser. Generalizing the partial global planning algorithm. *International Journal on Intelligent Cooperative Information Systems*, 1(2):319–346, June 1992.

[7] Virginia Dignum and Frank Dignum. Coordinating tasks in agent organizations. or: Can we ask you to read this paper? In *COIN@ECAI'06: Workshop on Coordination, Organization, Institutions and Norms in MAS*, Riva del Garda, Italy, August 2006.

[8] Virginia Dignum and Frank Dignum. Coordinating tasks in agent organizations. or: Can we ask you to read this paper? In *Fourth European Workshop on Multi-Agent Systems (EUMAS 2006)*, Lisbon, Portugal, December 2006.

[9] Virginia Dignum, Javier Vázquez-Salceda, and Frank Dignum. Omni: Introducing social structure, norms and ontologies into agent organizations. In *Programming Multi-Agent Systems, Second International Workshop ProMAS 2004*, pages 181–198, 2004.

[10] Marc Esteva, Julian Padget, and Carles Sierra. Formalizing a language for institutions and norms. In John-Jules Meyer and Milind Tambe, editors, *Pre-proceedings of the Eighth International Workshop on Agent Theories, Architectures, and Languages (ATAL-2001)*, pages 106–119, 2001.

[11] Jacques Ferber, Oliver Gutknecht, and Fabian Michel. From agents to organizations: An organizational view of multiagent systems. In *Proceedings of the Fourth International Workshop on Agent Oriented Software Engineering (AOSE03)*, pages 214–230. Springer Verlag, 2003.

[12] Jacques Ferber, Fabien Michel, and José-Antonio Báez-Barranco. Agre: Integrating environments with organizations. In *Environments for Multi-Agent Systems, First International Workshop, E4MAS 2004, New York, NY, USA, July 19, 2004, Revised Selected Papers*, pages 48–56, 2004.

[13] Mark S. Fox, Mihai Barbuceanu, Michael Gruninger, and Jinxin Lin. An organization ontology for enterprise modelling. In Michael Prietula, Kathleen Carley, and Les Gasser, editors, *Simulating Organizations: Computational Models of Institutions and Groups*, pages 131–152. AAAI/MIT Press, 1998.

[14] Vasco Furtado, Adriano Melo, Virginia Dignum, Frank Dignum, and Liz Sonenberg. Exploring congruence between organizational structure and task performance: A simulation approach. In *Proceedins of AAMAS05 Workshop, From Organizations to Organization Oriented Programming in MAS*, 2005.

[15] Jay R. Galbraith. *Designing Complex Organizations*. Addison-Wesley, 1973.

[16] Jay R. Galbraith. *Organization Design*. Addison-Wesley, 1977.

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

[18] Bryan Horling and Victor Lesser. A Survey of Multi-Agent Organizational Paradigms. *The Knowledge Engineering Review*, 19(4):281–316, 2005.

[19] Bryan Horling, Roger Mailler, and Victor Lesser. A Case Study of Organizational Effects in a Distributed Sensor Network. In *Proceedings of the International Conference on Intelligent Agent Technology (IAT 2004)*, Beijing, China, September 2004.

[20] 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.

[21] Bryan Horling, Roger Mailler, Mark Sims, and Victor Lesser. Using and Maintaining Organization in a Large-Scale Distributed Sensor Network. *Proceedings of the Workshop on Autonomy, Delegation, and Control (AAMAS03)*, July 2003.

[22] Jomi Fred Hübner, Jaime Simao Sichman, and Olivier Boissier. A model for the structural, functional, and deontic specification of organizations in multiagent systems. In *Proceedings of the 16th Brazilian Symposium on Artificial Intelligence*, pages 118–128. Springer-Verlag, 2002.

[23] Jomi Fred Hübner, Jaime Simão Sichman, and Olivier Boissier. Using the moise+ for a cooperative framework of mas reorganisation. In *Advances in Artificial Intelligence - SBIA 2004, 17th Brazilian Symposium on Artificial Intelligence*, pages 506–515, 2004.

[24] Jomi Fred Hübner, Jaime Simao Sichman, and Olivier Boissier. S-moise+: A middleware for developing organized multi-agent systems. In Olivier Boissier, Virginia Dignum, Eric Matson, and Jaime Simo Sichman, editors, *Proceedings of the International Workshop on Organizations in Multi-Agent Systems, from Organizations to Organization Oriented Programming in MAS (OOOP'2005) (LNCS Volume 3913*. Springer, 2006.

[25] M. Klusch and A. Gerber. Dynamic coalition formation among rational agents. *IEEE Intelligent Systems*, 17(3):42–47, May/June 2002.

[26] 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.

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

[28] James G. March and Herbert A. Simon. *Organizations*. John Wiley & Sons, 1958.

[29] Mairi McCallum, Wamberto W. Vasconcelos, and Timothy J. Norman. Verification and analysis of organizational change. In *Proceedings of AAMAS05 Workshop, From Organizations to Organization Oriented Programming in MAS*, 2005.

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

[31] Young pa So and Edmund H. Durfee. Designing organizations for computational agents. In *Simulating Organizations: Computational Models of Institutions and Groups*, pages 47–64. AAAI Press/MIT Press, 1998.

[32] H. Edward Pattison, Daniel D. Corkill, and Victor R. Lesser. Instantiating descriptions of organizational structures. In Michael N. Huhns, editor, *Distributed Artificial Intelligence*, Research Notes in Artificial Intelligence, chapter 3, pages 59–96. Pitman, 1987.

[33] Tuomas Sandholm and Victor Lesser. Coalitions among computationally bounded agents. *Artificial Intelligence, Special Issue on Economic Principles of Multi-Agent Systems*, 94(1):99–137, January 1997.

[34] David Servat and Alexis Drogoul. Combining amorphous computing and reactive agent-based systems: A paradigm for pervasive intelligence? In *Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent systems*, pages 441–448. ACM Press, 2002.

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

[36] Mark Sims, Claudia V. Goldman, and Victor Lesser. Self-organization through bottom-up coalition formation. In *AAMAS '03: Proceedings of the second international joint conference on Autonomous agents and multiagent systems*, pages 867–874, New York, NY, USA, 2003. ACM Press.

[37] M. Tambe, J. Adibi, Y. Alonaizon, A. Erdem, G. Kaminka, S. Marsella, and I. Muslea. Building agent teams using an explicit teamwork model and learning. *Artificial Intelligence*, 110:215–240, 1999.

[38] Egon L. van den Broek, Catholijn M. Jonker, Alexei Sharpanskeykh, Jan Treur, and Pinar Yolum. Formal modeling and analysis of organizations. In *Proceedings of the Seventeenth Belgium-Netherlands Conference on Artificial Intelligence, Brussels, Belgium, October 17-18, 2005*, pages 391–392, 2005.

[39] H. Van Dyke Parunak. Making swarming happen. In *Proceedings of the Conference on Swarming and Network Enabled Command, Control, Communications, Computers, Intelligence, Surveillance and Reconnaissance (C4ISR)*, January 2003.

[40] H. Van Dyke Parunak and Sven Brueckner. Entropy and self-organization in multi-agent systems. In *Proceedings of the fifth international conference on Autonomous agents*, pages 124–130. ACM Press, 2001.

[41] Javier Vázquez-Salceda and Frank Dignum. Modelling electronic organizations. In *Multi-Agent Systems and Applications III, 3rd International Central and Eastern European Conference on Multi-Agent Systems, CEEMAS 2003*, pages 584–593, 2003.

[42] Javier Vázquez-Salceda, Virginia Dignum, and Frank Dignum. Organizing multiagent systems. *Autonomous Agents and Multi-Agent Systems*, 11(3):307–360, 2005.

[43] Franco Zambonelli, Nicholas R. Jennings, and Michael Wooldridge. Organisational abstractions for the analysis and design of multi-agent systems. In *Proceedings of the First International Workshop on Agent-Oriented Software Engineering*, pages 127–141, Limerick, Ireland, 2001.