An Investigation of Reorganization Algorithms

Christopher Zhong Scott A. DeLoach
czhong@k-state.edu sdeloach@k-state.edu
Kansas State University
Manhattan, KS 66506

Abstract

Organization-based multiagent systems are designed to be highly adaptable systems. These systems are able to adapt themselves to changes that occur in the environment. One of the ways to achieve adaptability is through reorganization. Currently, there are a number of organization models that achieve adaptability with reorganization. This paper looks at how reorganization might occur for the Organization Model for Adaptive Computational Systems. This paper highlights the shortcomings of the current model with respect to general-purpose reorganization algorithms and provides some suggestions that could improve efficiency of these reorganization algorithms.

1 Introduction

As technology presses forward, more complex tasks are being delegated to computational systems. Generally, these systems are distributed and expected to adapt to changes in their environment. While distributed systems offer increased reliability and access to distributed resources, adaptive systems continue to perform effectively while reacting to their dynamically changing environments.

One approach to building adaptive, distributed systems is that of multiagent systems. However, early multiagent systems were typically designed with a set of predefined goals and emphasized individual agents and their interactions. This resulted in adaptivity at the agent-level with system-level adaptation being a byproduct of the agent-level adaptation.

To achieve system-level adaptation, a system-level mechanism is required. Such a mechanism is the focus of a number of ongoing research efforts based on an organizational metaphor. Various research groups are looking to provide mechanisms that guide a group of agents by specifying high-level objectives within a predefined organization structure. [1, 2, 3]

This paper investigates the feasibility of reorganization algorithms for the *Organization Model for Adaptive Computational Systems* (*OMACS*), provides some suggestions to improving OMACS with respect to these algorithms, and highlights some characteristics that can be used for designing "good" and "efficient" OMACS models.

2 Organization Model for Adaptive Computational Systems

Organization-based approaches typically follow the social definition of an organization. Socially, an organization is defined as a group of people who coordinate together to achieve shared goals. Figure 1 shows the OMACS model [1]. Only a subset of the model is described here because parts of the model are not relevant to this paper.

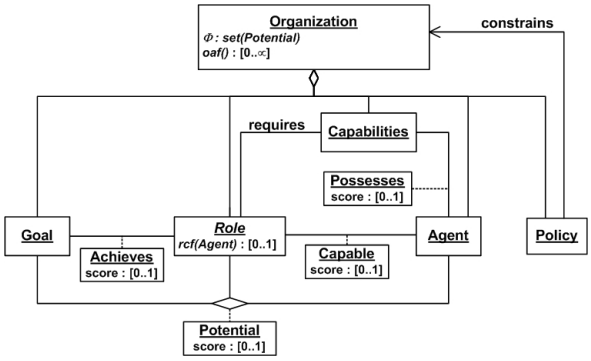

Figure 1: Organization Model

OMACS defines the standard entities of an organization: goals ($\boldsymbol{G}$), roles ($\boldsymbol{R}$), and agents ($\boldsymbol{A}$). Furthermore, three additional entities are defined: capabilities ($\boldsymbol{C}$), an assignment set ($\boldsymbol{\Phi}$), and policies ($\boldsymbol{P}$).

All organizations, even artificial organizations, have an overall goal that the organization is attempting to achieve. In OMACS, an organization contains a set of goals that the organization is try-

ing to achieve. This set of goals is defined by the goal model. In our research, we use the *Goal Model for Dynamic Systems* (*GMoDS*), which follows a classic AND/OR goal decomposition. A detailed description of GMoDS is given in [1].

Capabilities are essential in determining what roles agents are capable of playing in OMACS. Capabilities can represent a wide variety of abilities, both soft and hard. Some examples of soft abilities are having access to resources, communications, and executing computational algorithms. Hard abilities typically model the abilities of robots such as sensors and effectors [1].

In OMACS, the goals of an organization are achieved by a set of roles. Generally, a role is capable of achieving multiple goals and a goal can also be achieved by multiple roles. OMACS defines the `achieves` function, which takes as input a goal and a role, and returns a real value (ranging from $0 \ldots 1$) reflecting how well the given role achieves the given goal. A value of 0 means the given role is not able achieve a particular goal.

A role cannot be played by just any agent. Before an agent is allowed to play a given role, that agent must first meet the requirements of that role. First, a role requires a set of capabilities that agents must possess and second, the `rcf` function, also known as the role capability function, must return a positive value. The `rcf` takes as input a role and an agent and returns a real value ranging from $0 \ldots 1$. Possessing the required capabilities may not be a sufficient requirement for playing a role. Thus, the `rcf` allows roles to indicate the importance of certain capabilities.

In OMACS, every organization has a set of heterogeneous agents, which are defined as "computational system instances that inhabit a complex dynamic environment, sense and act autonomously in this environment, and by doing so realize a set of goals"[1]. The `possesses` function captures the quality of an agent's capabilities. The `possesses` function takes as input an agent and a capability, and returns a real value ranging from $0 \ldots 1$, where 0 indicates that an agent does not possess the given capability.

In OMACS, agents are assigned roles to achieve goals. How well an agent can achieve a particular goal is captured by the `potential` function. The `potential` function takes as input a goal, a role, and an agent, and returns a real value ranging from $0 \ldots 1$ to indicate how well the agent can play the role to achieve the goal.

In OMACS, every organization has a set of policies that govern it. OMACS provides three types of policies: assignment policies ($\boldsymbol{P_\Phi}$), behavioral policies ($\boldsymbol{P_{beh}}$), and reorganization policies ($\boldsymbol{P_{reorg}}$). Assignment policies provide restrictions on the assignment set such as "an agent can only play one role". Behavioral policies specify how the organization should behave when some event occurs. Reorganization policies provide heuristics that guide the organization when reorganizing.

OMACS contains an `oaf` function, also known as the organization assignment function, that determines the effectiveness of the current assignment set. The `oaf` returns an organization score of a real value ranging from $0 \ldots \infty$, where the higher the organization score the better the organization performs. Typically, the `oaf` is application specific. However, a default `oaf` is given in [1] as the sum of the potentials from the assignment set. An assignment is a tuple of agent, role, and goal that is placed in $\boldsymbol{\Phi}$. For example, $\langle a, r, g \rangle$ means that agent a has been assigned to play role r to achieve goal g.

$$oaf = \sum_{\forall \langle a,r,g \rangle \in \Phi} potential(a, r, g) \qquad (1)$$

3 Reorganization Algorithm

In our research, a general purpose reorganization algorithm that produces an optimal solution was developed for use with OMACS. Figure 2 shows the pseudo code for the reoganization algorithm. An optimal organization score is the organization with the highest organization score returned by the `oaf`. Therefore, finding the optimal organization score requires going through every possible combination of the assignment set and computing the organization score for each combination.

Lines 1–5 find and create all the possible role-goal mappings between every goal from $\boldsymbol{\omega_G}$ and the roles that are able to achieve that goal. Line 6 creates a powerset from the mappings (which is a set of sets of mappings) and then uses the assignment policies to remove invalid sets of mappings. Lines 7–13 find and create all the possible assignments between the agents from $\boldsymbol{\omega_A}$ and the elements from the reduced powerset. If an agent is capable of playing a given set of role-goal mapping(s), then an assignment is created. Line 14 removes invalid assignments from the based on the assignment policies, and then computes the combinations of the assignments. A combination is an assignment set. Lines 15–24 go through each combination and keeps track of the best combination so far. Line 25 returns the best combination (as-

function reorganize(`oaf`, $\boldsymbol{\omega_G}$, $\boldsymbol{\omega_A}$)

1: **for each** goal g from $\boldsymbol{\omega_G}$ **do**
2: **for each** role r that achieves g **do**
3: $m \cup \langle r, g \rangle$
4: **end for**
5: **end for**
6: $ps \leftarrow \boldsymbol{P_\Phi}(\text{powerset}(m))$
7: **for each** agent a from $\boldsymbol{\omega_A}$ **do**
8: **for each** set s from ps **do**
9: **if** a is capable of playing s **then**
10: $pa \cup \langle a, s \rangle$
11: **end if**
12: **end for**
13: **end for**
14: $c \leftarrow$ combinations($\boldsymbol{P_\Phi}(pa)$)
15: **for each** combination i from c **do**
16: **for each** assignment x from pa **do**
17: $\boldsymbol{\Phi} \cup \langle x.a, x.s_i \rangle$
18: **end for**
19: **if** $\boldsymbol{P_\Phi}(\boldsymbol{\Phi})$ is valid **then**
20: **if** $(score \leftarrow \text{oaf}(\boldsymbol{\Phi})) > best.score$ **then**
21: $best \leftarrow \langle score, \boldsymbol{\Phi} \rangle$
22: **end if**
23: **end if**
24: **end for**
25: **return** $best.\boldsymbol{\Phi}$

Figure 2: Pseudo Code

signment set).

The best case time complexity of the algorithm is $\Theta(2^{g \times r_{avg} g})$ and $\Theta(2^{g \times r_{avg} g \times a})$ is the worst case, where r_{avg} is the average number of roles that can achieve each goal (see [4] for proof details). The exponential time complexity is the result of the "blackbox" functionality provided by the `oaf` and `rcf`. This exponential time complexity prevents any practical use of a general purpose reorganization algorithm during runtime.

4 Reducing Time Complexity

There are several approaches to reducing the time complexity of the algorithm.

Because OMACS provides a "blackbox" functionality to reorganization algorithms, the lack of information on the internal workings of the `oaf` and `rcf` restrict the use of heuristics in general purpose reorganization algorithms. Efficient reorganization algorithms have to be application specific or make assumptions about the organization.

Exposing the internal workings of the `oaf` and `rcf` requires a modification to OMACS. A better interface to the `oaf` and `rcf` would allow more efficient algorithms to be designed for general use. For instance, for the `oaf`, knowing that when two agents are playing the same role, their combined score would be a factor instead of the simple additive score would help tremendously in reducing the search space. Again, for the `rcf`, knowing how much a capability contributes to the `rcf` score would also help. One approach to exposing the internal workings of the `oaf` and `rcf` would be to encode the internal workings of the `oaf` and `rcf` into some standardized data structure. Algorithms would then be able to extract information from the data structure about the `oaf` and `rcf`. Further research into this area may reveal how the `oaf` and `rcf` can be redesigned for more effective uses.

An overview of the communication costs for distributed reorganization algorithms is given in [4], which shows that a simple distributed algorithm improved the time complexity by a factor. The best case is $O((g \times r_{avg} g) + \lceil \frac{2^{g \times r_{avg} g}}{a} \rceil + \epsilon)$, where ϵ is the communication cost, and the worst case is $O(2^{g \times r_{avg} g \times a} + \epsilon)$. By adopting a purely distributed approach to designing reorganization algorithms, there could be significant gains in the time complexity as the number of agents increases. However, the additional communication costs could be significant as well. Further research into this area may unveil the advantages and disadvantages of distributed reorganization algorithms.

Assignment policies have varying effects on the time complexity of reorganization algorithms. For instance, if there is a policy where "agents can only play one role at a time", the search space for each agent is reduced significantly. For example, assume an agent is able to play five roles and each role achieves three goals. Without the policy, this agent has $2^{5 \times 3 = 15} = 32,768$ possible assignments. However, with the policy, this agent only has $5 \times 3 = 15$ possible assignments. If there are four of such similar agents, then there are $32,768^4 = 1,152,921,504,606,846,976$ combinations without the policy but only $15^4 = 50,625$ combinations with the policy.

Not all assignment policies have such a great effect on the time complexity. Identifying the types of assignment policies that can significantly affect the time complexity could help designers produce models that are flexible as well as efficient. While reducing the number of combinations, assignment policies may add to the time complexity [4]. In the preliminary analysis, the algorithm was modifed into two versions that use the policy "agents can only play one role at a time". There are two

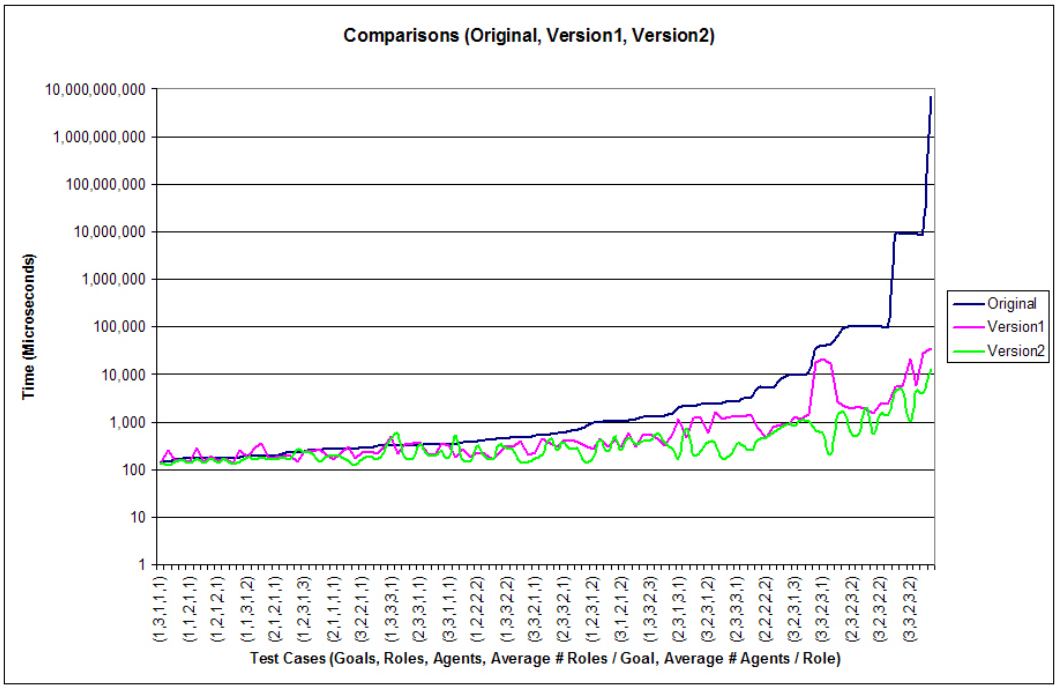

Figure 3: Comparisons of Results

locations that policy checking occurs, however, this particular policy is only applicable to the first check (line 6). The first version prunes the power set by removing invalid sets. The second version replaces the power set function with a custom function that only generates valid sets. Figure 3 shows the preliminary results comparing the original algorithm versus the two modified versions.

As expected, first version performs worse than the original version in small cases because the power set function is the greatest contributor to the time complexity and extra time is required to remove invalid sets. However, when the power set function stops being the greatest contributor to the time complexity, the first version begins to perform better than the original version because the reduced power set results in a smaller number of combinations. On the other hand, the second version always performs better than both the original and first version because the function that replaces the power set function only generates the necessary sets instead of removing unwanted sets.

5 Conclusions

This paper lays the groundwork for further research into OMACS. To enable more efficient reorganization algorithm, one suggestion is to extend OMACS by exposing the internals of `oaf` and `rcf`. Another approach is to look at how assignment policies can be integrated more deeply into the design of reorganization algorithms. A third approach is to distribute the computational work of reorganization to the agents. In conclusion, general purpose reorganization algorithms for OMACS can be made feasible for runtime use if some of the suggestions given in this paper are followed.

References

[1] Scott A. DeLoach and Walamitien H. Oyenan. An organizational model and dynamic goal model for autonomous, adaptive systems. Multiagent & Cooperative Robotics Laboratory Technical Report MACR-TR-2006-01, Kansas State University, March 2006.

[2] Virginia Dignum. *A Model for Organizational Interaction: Based on Agents, Founded in Logic*. PhD thesis, Utrecht University, 2004.

[3] Virginia Dignum, Javier Vázquez-Salceda, and Frank Dignum. Omni: Introducing social structure, norms and ontologies into agent organizations. In *PROMAS*, pages 181–198, 2004.

[4] Christopher Zhong. An investigation of reorganization algorithms. Master's thesis, Kansas State University, 2006.