Exploiting Reusable Organizations to Reduce Complexity in Multiagent System Design *

Walamitien H. Oyenan, Scott A. DeLoach, and Gurdip Singh

Multiagent and Cooperative Robotics Laboratory, Kansas State University
234 Nichols Halls, Manhattan Kansas, USA
{oyenan,sdeloach,gurdip}@k-state.edu

Abstract. Organization-based Multiagent Systems are a promising way to develop complex multiagent systems. However, it is still difficult to create large multiagent organizations from scratch. Multiagent organizations created using current AOSE methodologies tend to produce ad-hoc designs that work well for small applications but are not easily reused. In this paper, we provide a conceptual framework for designing reusable multiagent organizations. It allows us to simplify multiagent organization designs and facilitate their reuse. We formalize the concepts required to design reusable organization-based multiagent services and show how we can compose those services to create larger, more complex multiagent systems. We demonstrate the validity of our approach by designing an application from the cooperative robotics field.

Key words: Multiagent Organizations, Design Methodologies, Cooperative Robotics

1 Introduction

Multiagent Systems (MAS) have been seen as a new paradigm to cope with the increasing need for dynamic applications that adapt to unpredictable situations. Large MAS are often composed of several autonomous agents engaging in complex interactions with each other and their environment. Consequently, providing a correct and effective design for such systems is a difficult task. To reduce this complexity, Organization-based Multiagent Systems (OMAS) have been introduced and they are viewed as an effective paradigm for addressing the design challenges of large and complex MAS [8, 25]. In OMAS, the organizational perspective is the main abstraction, which provides a clear separation between agents and system, allowing a reduction in the complexity of the system. To support the design of OMAS, several methodologies have been proposed [7].

Nonetheless, one of the major problems with the wide-scale adoption of OMAS for the development of large-scale applications is that, so far, the methodologies proposed work well for small systems, but are not well suited

* This work was supported by grants from the US National Science Foundation (0347545) and the US Air Force Office of Scientific Research (FA9550-06-1-0058).

for developing large and complex applications. Designers generally handle complexity based on intuition and experience, leading to ad-hoc designs that are difficult to maintain.

Therefore, we propose employing reusable multiagent organizations designed using both component-oriented and service-oriented principles. These two well-established software engineering approaches allow us to decompose large multiagent organizations into smaller organizations that can be developed separately and composed later, thus allowing designers to design large and complex OMAS more easily. Herein we assume that organizations are populated by cooperative agents that always attempt to achieve their assigned goals.

In our approach, services are the key concept allowing us to compose OMAS components into larger, more complex systems. *OMAS components* are autonomous multiagent organizations that provide or use services as defined by generic interfaces called *connection points*. OMAS components are composed such that required services match provided services. The composition process ensures the consistency of all the bindings and results into a valid composite organization.

Our contribution is two-fold. First, we rigorously specify the key concepts used in our framework and describe the mechanisms that allow the design of reusable OMAS components. Second, we formally define the composition process through which reusable OMAS components can be composed to build larger organizations.

We present related work in Section 2 followed by an overview of our organizational model in Section 3. Section 4 defines the key concepts of composable organizations while Section 5 describes our composition process. Finally, we illustrate our approach with an example followed by conclusions and future work.

2 Related Work

Several frameworks for multiagent systems have been proposed to deal with the complexity of large software systems [8, 9, 18, 25], while decomposition has long been suggested as a mechanism to deal with complex systems [13]. While current agent-oriented methodologies suggest decomposing organizations, they fail to provide guidance or a rigorous composition process. For instance, Ferber et al. propose partitioning a multiagent system into groups [8] that interact via a gatekeeper agent participating in multiple groups. Unfortunately, they provide no prescription for actually aggregating those groups into a coherent system. Cossentino et al. also propose developing OMAS with a hierarchical structure based on holons [3]. While they divide the system into loosely coupled sub-organizations (holons), there is no guidance on how to recombine them. In general, in order to recombine the organizations, the designer must understand the internal behavior of each sub-organization. Our approach proposes the rigorous specification of interfaces required for composition, thus

supporting reusability and maintainability of complex OMAS. Organizations can be developed by different designers and combined later.

Others have proposed compositional approaches for building MAS. DESIRE [1] supports the compositional design of MAS in which components represent agents that can be composed of subcomponents. However, like other component-based frameworks, DESIRE is agent-centric and does not support OMAS.

Few agent-oriented approaches explicitly use service-oriented principles. Most of the unifying work between services and agents concern agent-based service-oriented systems, where agents simply wrap services [11, 12]. However, Cao et. al. propose a methodology called Organization and Service Oriented Analysis and Design (OSOAD) [2], which combines organizational modeling, agent-based design and service-oriented computing to build complex OMAS. Their approach is similar to ours in the sense that complex OMAS are built using service concepts. However, OSOAD services are offered only at the agent level. Our approach permits entire organizations to act as service providers (while still allowing agent-level services), thus allowing us to develop *cooperative services* that cannot be provided by individual agents. In addition, we specify standard interfaces that allow OMAS components to be composed without requiring internal design knowledge.

3 Organizational Model

Our work is based on the Organization Model for Computational Adaptive Systems (OMACS) [6]. OMACS is a formal framework for describing OMAS and is supported by the O-MaSE process framework [9]. OMACS defines an organization as a set of goals (G) that the organization is attempting to accomplish, a set of roles (R) that must be played to achieve those goals, a set of capabilities (C) required to play those roles and a set of agents (A) who are assigned to roles in order to achieve organizational goals. While an agent may play an assigned role in any way it wishes, OMACS does assume the agent will adhere to some minimal expected behavior (e.g. working toward its assigned goal). (There are more entities defined in OMACS that are not relevant for this paper and the reader is referred to [6] for the complete model). In this paper, we use a generalization of the OMACS model and only consider the goals, roles and the relationship that exists between them. Those entities represent the persistent part of the organization, the organization structure [8], which can be populated later with heterogonous agents to produce a concrete organization. There are many other organizational models for OMAS [7] and the approach proposed in this paper could well be adapted to any of them.

In a typical multiagent organization, organizational goals and roles are organized in a goal tree [10, 22, 24] and a role model [14, 24, 25] respectively. We choose to organize our goals using a Goal Model for Dynamic Systems (GMoDS) [16]. In a GMoDS goal model, goals are organized in a goal tree such that each goal is decomposed into a set of subgoals using an OR-decomposition

or an AND-decomposition. In addition, the GMoDS goal model contains two time-based relationships between goals: the *precedes* and *triggers* functions. We say *goal g1 precedes goal g2* if g1 must be satisfied before g2 can be pursued by the organization. Moreover, during the pursuit of specific goals, events may occur that cause the instantiation of new goals. Instantiated goals may be parameterized to capture a context sensitive meaning. If an event e can occur during the pursuit of goal g1 that instantiates goal g2, we say *g1 triggers g2 based on e*. GMoDS defines a goal model GM as a tuple $\langle G, Ev, parent, precedes, triggers, root\rangle$ where:

- *G: set of organizational goals (where the set* G_L *represent the leaf goals).*
- *Ev: set of events.*
- *parent:* $G \rightarrow G$ *; defines the parent goal of a given goal.*
- *precedes:* $G \rightarrow 2^G$ *; indicates all the goals preceded by a given goal.*
- *triggers:* $Ev \rightarrow 2^{G \times G}$*;* $\langle g1, g2\rangle \in triggers(e)$ *iff g1 triggers g2 based on e.*
- $root \in G$*; the root of the goal model.*

We organize our roles using a role model that connects the various roles by protocols. There are two types of roles: internal roles and external roles. Internal roles are the typical roles defined inside the organization. External roles are placeholders for roles from an external organization; they represent unknown roles with which the organization must interface. Eventually external roles will be replaced by concrete roles (internal roles) from other organizations. We define our role model RM as a tuple $\langle R, P, participants\rangle$ where:

- *R: set of internal and external roles*
- *P: set of protocols*
- *participants:*$P \rightarrow 2^{R \times R}$*; indicates the set of role pairs connected by a protocol*

Finally, we define a multiagent organization *org* as a tuple $\langle GM, RM, achieves, INCP, OUTCP\rangle$ where:

- *GM: Goal Model*
- *RM: Role Model*
- *achieves:* $R \rightarrow 2^{G_L}$ *; indicates the set of leaf goals achieved by given a role.*
- *INCP: the set of entry connection points exposed by the organization (see Section 4.3).*
- *OUTCP: the set of exit connection points exposed by the organization (see Section 4.3)*

4 Service Model

In our framework, services are common functionalities encapsulated in OMAS components. Once designed, OMAS components can be used by other organizations to build larger systems. Fig. 1 shows our metamodel, comprising the service and organizational entities along with their relationships. The central concept is that of *Service*. Services offer one or more *operations*. Each operation

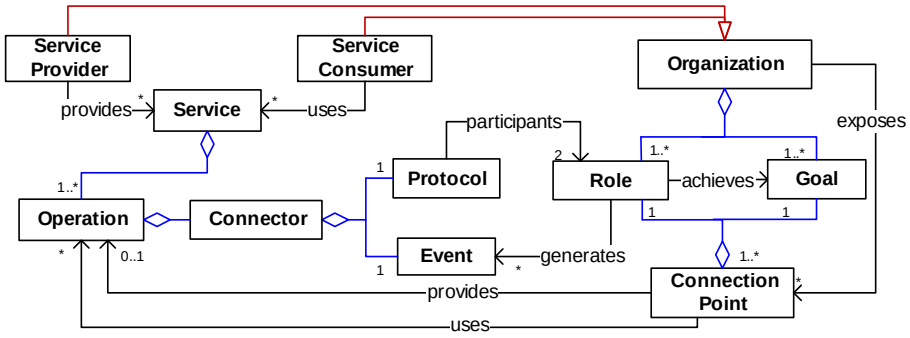

Fig. 1. Organizational Service Metamodel

possesses a *connector* that is used to connect connection points exposed by organizations. *Connection points* are goals and roles that can be bound together by *events* and *protocols* from connectors. *Service providers* and *service consumers* are both autonomous organizations who respectively provide and use operations from services. These entities are discussed in detail in the following subsections.

4.1 Services

A service is a logical entity that represents a coarse-grained multiagent functionality. This coarse-grained functionality is made of a set of fine-grained functionalities, called operations. Each service possesses an *XML-based specification* that contains a description of what the service proposes and provides a specification of each operation provided. To be functional, a service must be implemented by at least one provider. Services facilitate reuse in that they allow consumers to request operations based solely on the service specification.

4.2 Operations and Connectors

An operation represents an implementation of a functionality declared in a service. From an organizational standpoint, we view an *operation* as a set of application-specific organizational goals that an organization needs to achieve in order to reach a desired state. Operations can result in computations (e.g. computing an optimal path for a swarm of UAVs) or actions (e.g. neutralizing an enemy target).

Each operation has a set of *preconditions* and *postconditions*, an *interaction protocol*, and a *request event*. The request event is used to invoke the operation and includes the parameters passed to the operation at initialization. Once the operation is instantiated, the interaction occurs via the interaction protocol, which specifies the legal interactions between consumers and providers. The interaction protocol and the request event form a *connector*, which provides the "glue" that binds consumers and providers together.

4.3 Connection Points

A *connection point* is a logical construct associated with an operation. It is composed of a goal-role pair. There are two types of connection points: entry and exit connection points. An *entry connection point* guarantees a proper execution of the operation it provides. Its goal and role components are called the entry goal and entry role respectively. The set of entry connection points of an organization is denoted by INCP. We say that an entry connection point *provides an operation* if its entry goal is instantiated based on the occurrence of the request event of that operation and its entry role engages with an external role in the interaction protocol defined for that operation. An *exit connection point* guarantees a proper request of the operation it uses. Its goal and role components are called the exit goal and exit role respectively. The set of exit connection points of an organization is denoted by OUTCP. We say that an exit connection point *uses an operation* if its exit goal generates a goal trigger based on the request event of that operation and if its exit role engages with an external role in the interaction protocol defined for that operation (Fig. 2). Hence, a connection point role is an organizational role that would need to be played by an agent in order to achieve the corresponding connection point goal.

4.4 Service Providers

A *service provider* is an organization (OMAS component) that provides all the operations of a particular service. We say that an organization provides a service if, for all operations of the service, it exposes a unique entry connection point providing that operation. In addition, a service provider needs to be designed such that whenever an operation is requested and its preconditions are met, it pursues a set of its goals whose achievement satisfies the postconditions of that operation. As for any goal failures in OMACS, operation failures result in an autonomous reorganization in an attempt to overcome the failure [17].

4.5 Service Consumer

A *service consumer* is an organization (OMAS component) that uses one or more operations from various services. To use an operation, an organization needs to exposes at least one exit connection point using that operation. For each operation used, service consumers can expose multiple connection points. Designers of service consumers choose the operations they need based on the service specification that describes what each of its operation is suppose to do.

5 Composition of Services

Composition is a design-time process that binds a consumer organization with a provider in order to create a single composite organization. This process is illustrated in Fig. 2. Basically, given an operation, the composition process

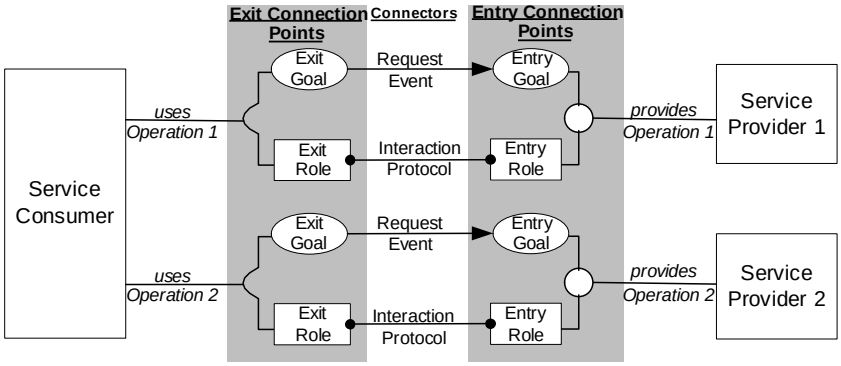

Fig. 2. Composition of Organizations through connection points and connectors

connects the exit connection point of a consumer to the entry connection point of a provider using the operation's connector. This interconnection ensures that the consumer organization can invoke the operation via the request event and that both organizations can interact via the interaction protocol. Formally, the *composition of organizations* org_1 *with* org_2 *over a connection point* cp_1 *requiring an operation op* is defined whenever cp_1 is an exit connection point from org_1 using op and org_2 exposes a connection point cp_2 providing op. This composition is denoted $org_1 \vdash^{cp_1,op} org_2$.

Sometimes, designers may want to compose all exit connection points using the same operation from only one provider. Thus, we define the composition of two organizations over an operation as their successive compositions over all the exit connection points requiring that operation. Hence, for all connection points cp_i from org_1 using an operation op, we have:

$$org_1 \vdash^{op} org_2 = (...((org_1 \vdash^{cp_1,op} org_2) \vdash^{cp_2,op} org_2) \vdash^{\cdots} ... \vdash^{cp_n,op} org_2).$$

The composition process is iterative and continues until the resulting composite organization requires no more operations. The result is a standalone application that uses no external services. Having a single organization simplifies reorganization tasks by allowing us to reuse existing work concerning reorganization of single organizations [20, 21, 26].

Next, we formally define the composition process through which reusable OMAS components can be composed to build larger organizations. We have a proof sketch that shows this composition will always be correct under certain conditions, but space would not permit us to put any details in the paper.

Given two organizations $org_1 = \langle GM_1, RM_1, achieves_1, INCP_1, OUTCP_1 \rangle$, $org_2 = \langle GM_2, RM_2, achieves_2, INCP_2, OUTCP_2 \rangle$, an operation *op* and two connection points cp_1 from org_1 and cp_2 from org_2 such that cp_1 *uses op* and cp_2 *provides op*. Given that $org_3 = \langle GM, RM, achieves, INCP, OUTCP \rangle$, such that $org_3 = org_1 \vdash^{cp_1,op} org_2$, we define the composite organization org_3 in the next subsections.

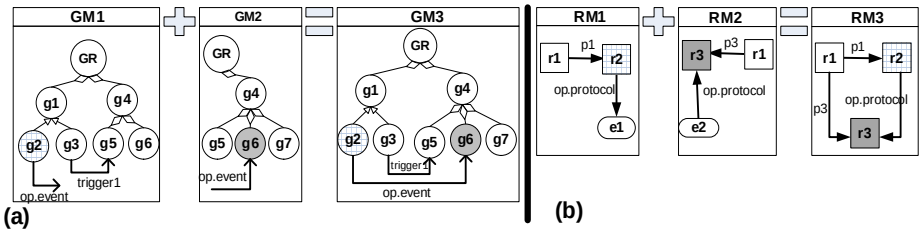

Fig. 3. Merging goal models (a) and role models (b)

5.1 Goal Model Composition

Without loss of generality, we assume that all goal models have the same root, which is an AND-decomposed goal called the generic root (GR). Moreover, we consider that two goals are equal if they are identical and their parents are identical. This definition of equality of goals ensures that the union of two goal trees is a tree instead of a graph. Given two goal models $GM_1 = \langle G_1, Ev_1, parent_1, precedes_1, triggers_1, GR\rangle$, and $GM_2 = \langle G_2, Ev_2, parent_2, precedes_2, triggers_2, GR\rangle$, we define the composite goal model $GM = \langle G, Ev, parent, precedes, triggers, root\rangle$ such that:

$\boldsymbol{root} = GR, \quad \boldsymbol{G} = G_1 \cup G_2, \quad \boldsymbol{Ev} = Ev_1 \cup Ev_2,$
parent: $\forall g \in G, parent(g) = parent_1(g) \cup parent_2(g),$
precedes: $\forall g \in G, precedes(g) = precedes_1(g) \cup precedes_2(g),$
triggers: $\forall e \in Ev$, *triggers(e)*=

$$= \begin{cases} triggers_1(e) \cup triggers_2(e) & \text{if } e \neq op.event, \\ triggers_1(e) \cup triggers_2(e) \cup \{(cp_1.goal, cp_2.goal)\} & \\ \quad -\{(cp_1.goal, \varnothing), (\varnothing, cp_2.goal)\} & \text{if } e = op.event. \end{cases}$$

Note that cp_1.goal is an exit goal in GM_1 and cp_2.goal is an entry goal in GM_2. The composition is illustrated in Fig. 3a, where g2 is an exit goal and g6 is an entry goal.

5.2 Role Model Composition

Given $RM_1 = \langle R_1, P_1, participants_1\rangle$, $RM_2 = \langle R_2, P_2, participants_2\rangle$, let e_1 and e_2 be two external roles such that $(cp_1.role, e_1) \in participants_1(op.protocol)$ and $(e_2, cp_2.role) \in participants_2(op.protocol)$, where cp_1.role is an exit role in RM_1 and cp_2.role is an entry role in RM_2. We define $RM = \langle R, P, participants\rangle$ such that:

$\boldsymbol{R} = R_1 \cup R_2 - \{e_1, e_2\}, \quad \boldsymbol{P} = P_1 \cup P_2,$
participants: $\forall p \in P$, *participants(p)*=

$$= \begin{cases} participants_1(p) \cup participants_2(p) & \text{if } p \neq op.protocol, \\ participants_1(p) \cup participants_2(p) \cup \{(cp_1.role, cp_2.role)\} & \\ \quad -\{(cp_1.role, e_1), (e_2, cp_2.role)\} & \text{if } p = op.protocol. \end{cases}$$

The composition of role models we have just described is illustrated in Fig. 3b. In this figure, role r2 is an exit role and role r3 is an entry role.

5.3 Organization Composition

Finally, to complete org_3, we need to define the *achieves* function along with the connection points. The *achieves* function is defined as:

$$\textbf{achieves(r)} = achieves_1(r) \cup achieves_2(r), \quad \forall r \in R.$$

The sets of entry and exit connection points exposed by org_3 are:

$$\boldsymbol{INCP} = INCP_1 \cup INCP_2 - \{cp_2\}.$$
$$\boldsymbol{OUTCP} = OUTCP_1 \cup OUTCP_2 - \{cp_1\}.$$

6 Case Study

To demonstrate the validity of our framework for designing OMAS, we design an application called Cooperative Robotic for Airport Management (CRAM). In this application, a team of heterogeneous robots is in charge of handling some aspects of the airport management task. Essentially, the team needs to clean the building and perform cargos inspections. Suspicious cargos are sent to another location for further inspection.

In our framework, OMAS components can be present in a repository or come from the decomposition of the current problem. For this example, we develop one service, the *cleaning service*, and explain how it can be used to develop our CRAM application.

In the organization models presented in this example (Fig. 4, Fig. 5, and Fig. 6), *goals* are shown as ovals, *internal roles* as rectangles, *external roles* as round rectangles, *precedes* and *triggers* functions as open-head arrows, *protocols* as full-head arrows and *achieves* functions as dashed lines. *Conjunctive goals* are connected to their subgoals by diamond-shaped links and *disjunctive goals* by triangle-shaped links. *Entry goals* are identified by being the destination of a trigger that has no source. *Exit goals* are always the leaf goals achieved by exit roles. In the role models, agent capabilities [6] are identified by the keyword *'requires'*. Entry roles specify operations provided by using the keyword *'provides'* while exit roles specify operations required by the keyword *'uses'*. Due to space limits, we do not discuss aspects of the organization irrelevant to our approach.

6.1 The Cleaning Service

Cooperative cleaning is a common problem in cooperative robotics and several works have been published regarding the use of robots for cleaning [15, 19, 23]. Here, we propose a *Cleaning Service* whose main operation is to clean a given area. We design the *Cooperative Cleaning Organization*, shown in Fig. 4, which involves a team of robots coordinating their actions to clean an area. Hence, this

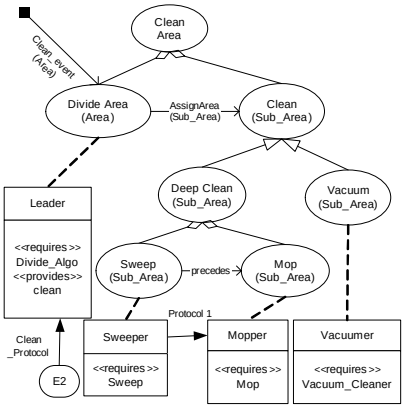

Fig. 4. Cooperative Cleaning organization model

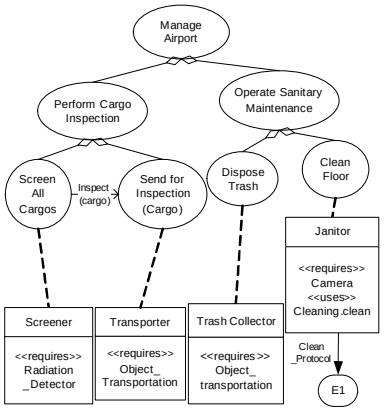

Fig. 5. CRAM organization model

OMAS component provides the *Cleaning Service*. The entry connection point providing the *clean* operation is made of the goal *Divide Area* and the role *Leader*. The *Divide Area* goal is in charge of dividing an area into smaller areas that can be handled by individual robots. Once the area to be cleaned has been divided, the *Clean* goal is triggered. The *Clean* goal is decomposed into two disjunctive goals. Hence, it offers two ways of cleaning; the organization can decide to either do a deep clean (*Deep Clean* goal) or just vacuum (*Vacuum* goal). The *Deep Clean* goal is further decomposed into two conjunctive goals: *Sweep* and *Mop*.

6.2 The Cooperative Robotic for Airport Management Organization

Next, we build the CRAM organization that uses the Cleaning Service. Its design is presented in Fig. 5. The main goal of the system, *Manage Airport*, has two conjunctive subgoals that represent the two main tasks of our system: *Perform Cargo Inspection*, *Operate Sanitary Maintenance*. Those goals are in turn further decomposed into conjunctive leaf goals. For each leaf goal in the CRAM organization, we design a role that can achieve it. Moreover, we identify that the *Janitor* role can use the *Cleaning Service* for the achievement of the *Clean Floor* goal. Thereby, the organization created contains the exit connection point (identified as goal-role pair): $\langle CleanFloor, Janitor \rangle$.

6.3 The Composition Process

In this section, we compose the CRAM application with the cleaning component in order to obtain a single composite organization. The CRAM uses the *clean* operation from the *Cleaning Service* that is provided by the *Cooperative Cleaning* organization. Let *cram* and *cleaning* be the CRAM and Cooperative Cleaning organizations respectively and let *cp_Janit* and *cp_Lead* be the connection

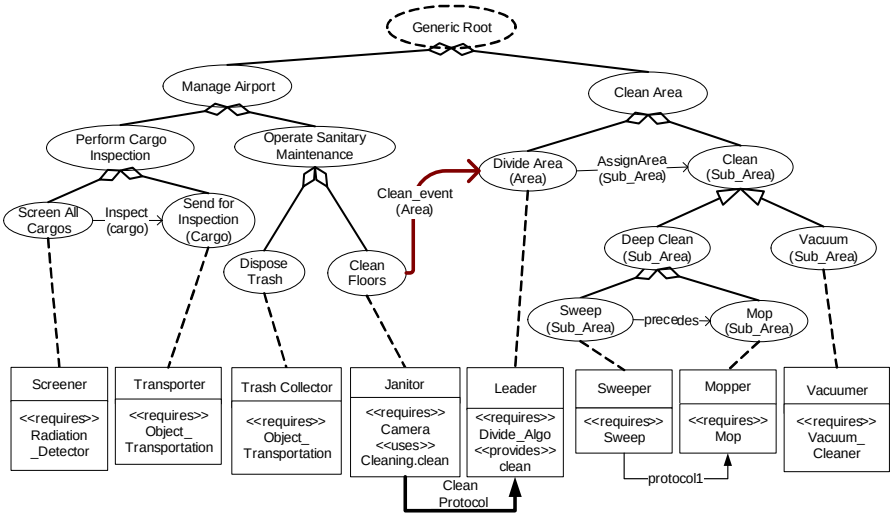

Fig. 6. Organization model obtained by composition of cram and cleaning over the clean operation.

points $\langle CleanFloor, Janitor\rangle$ from *cram* and $\langle DivideArea, Leader\rangle$ from *cleaning* respectively. We have:

$$\boldsymbol{cleaning} = \langle gm\_svc, rm\_svc, achieves\_svc, incp\_svc, outcp\_svc\rangle \ ,$$

where goal model gm_svc, role model rm_svc and achieves function $achieves\_svc$ are defined as described in Fig. 4, entry connection points set $incp\_svc = \{cp\_Lead\}$, and exit connection points set $outcp\_svc = \{\}$.

$$\boldsymbol{cram} = \langle gm\_app, rm\_app, achieves\_app, incp\_app, outcp\_app\rangle,$$

where goal model gm_app, role model rm_app and achieves function $achieves\_app$ are defined as described in Fig. 5, entry connection points set $incp\_app = \{\}$, and exit connection points set $outcp\_app = \{cp\_Janit\}$.

By composing *cram* with *cleaning* over operation *clean*, we have:

$$cram \vdash^{clean} cleaning = cram \vdash^{cp\_Janit,clean} cleaning = cram\_clean,$$

such that $\boldsymbol{cram\_clean} = \langle gm, rm, achieves, incp, outcp\rangle$, where:

gm, rm, achieves are defined as described in Fig. 6,

$incp = incp\_app \cup incp\_svc - \{cp\_Lead\} = \{\}$,

$outcp = outcp\_app \cup outcp\_svc - \{cp\_Janit\} = \{\}$.

Hence, by composing the *cram* and *cleaning* organizations (Fig. 4 and Fig. 5) over the *clean* operation specified in the *Cleaning Service*, we obtain the composed organization *cram_clean* modeled in Fig. 6.

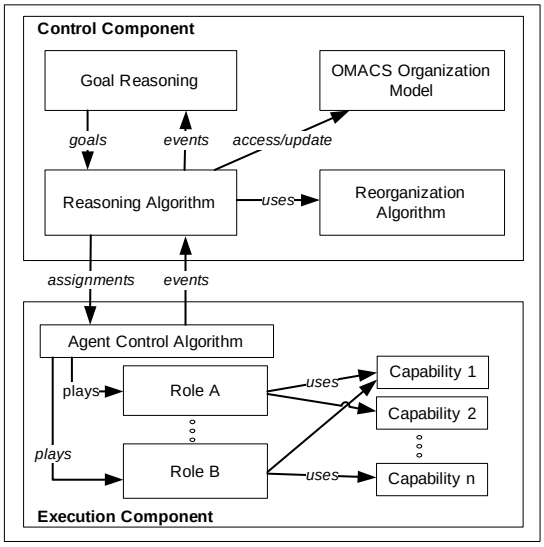

Fig. 7. Organization-based Agent Architecture

6.4 Implementation Overview

In this section, we give a brief overview of how the composite organization is implemented. Our implementation is based on design models obtained from the Organization-based Multiagent System Engineering (O-MaSE) process framework [9], which allows designers to create custom agent-oriented development processes. The implementation of the CRAM organization is based on several design models: the goal model and role model obtained from the composition process as shown in Fig. 6, and an agent model, a capability model, a protocol model and a plan model. However, depending on the actual process used, some of those models are optional [9]. The Capability Model identifies the capabilities and specifies their actions on the environment as state machines. The Agent Model captures agent types, along with the capabilities they posses. The Protocol Model specifies all the protocols that will be executed by agents enacting their assigned roles. Finally, the Plan Model provides a plan that agents execute in order to play a certain role. Plans are expressed as state machines.

At runtime, four main components allow the system to be autonomous and adapt to its environment. Those components are part of the *Control Component* of an agent (Fig. 7). The *Goal Reasoning* takes the goal model as input and tells the system which goals are active and can be pursued by the organization. Details about this component can be found in [16]. The *Organization Model* stores all the knowledge about the structure of the organization [5]. The *Reorganization Algorithm* takes as input a set of active goals and returns a set of assignments [26]. An assignment assigns an agent to play a role in order to achieve a particular goal. Assignments are made based on the capability possessed by agents and

some utilities functions. Once an agent has been assigned to play a role, it follows the role's plan in order to achieve its goal. Further details about how assignments are made can be found in [6]. The Goal Reasoning component and the Reorganization algorithm can be implemented in a centralized or distributed way. In a centralized version, only one agent is in charge of the organization. In a distributed version, all agents participate in the organization decision process. Finally, the *Reasoning Algorithm* interacts with the other components in order to make decisions about the next state of the organization [5]. All decisions taken at the *Control Component* level are passed on to the *Execution Component* that is in charge of executing the appropriate roles while using the required capabilities (Fig. 7).

For brevity, we only describe partially the runtime of the model shown in Fig. 6. Detailed implementations of similar OMACS-based systems have been presented in our previous work [6, 17]. During the execution, the goal *Clean Floor* eventually becomes active. A reorganization then occurs which results in the assignment of an agent (say the *Maintenance* agent) to play the *Janitor* role to achieve the newly created *Clean Floor* goal. During the enactment of the *Janitor* role, the event *clean_event* (cf. Fig. 6) occurs whenever an area that needs to be cleaned is detected. This event results in the creation of the goal *Divide Area*, which is the entry point of the clean service. Hence, the trigger acts as a request for the clean service. After reorganization, the *Divide Area* goal is assigned to an agent (say *Clean Manager* agent) playing the *Leader* role. The *Clean Manager* agent plays its role by dividing the area to be cleaned into subareas. Each subarea will be eventually assigned to an agent and when all agents complete, the area is clean. The *Manager* agent (part of the Service Provider organization) can then notify the *Maintenance* agent (part of the Service Consumer organization) through the protocol *clean_protocol* (cf. Fig. 6). At the same time, the *Maintenance* agent can continue looking for other areas to clean and request the cleaning service as often as necessary.

7 Conclusion and Future Work

We have presented an approach to support the development of complex OMAS by developing reusable OMAS components. While many current approaches use decomposition to support separation of concerns at design, they lack effective support for the composition process to recombine the sub-organizations. Our approach defines a composition framework that allows MAS designers to reuse predefined OMAS modeled as components. We described how to design OMAS components so they expose the necessary interfaces to potential consumers so they can request the desired operations. Moreover, we presented a composition process that combines multiple multiagent organizations into a single organization. Finally, we illustrated our approach by composing a Cooperative Robotic Airport Management application with a multiagent cleaning service to produce a single complete system design.

A significant advantage of our approach is the ability to compose multiagent organizations to develop a wide variety of complex applications. In addition, independent OMAS components are easily modifiable, offer a better approach to reusability of design models, help reduce development time and provide better structuring of large and complex MAS. Designers have more flexibility as service providers can easily be replaced with little or no change in the core organization.

We are currently working on extending the O-MaSE process framework [9] and the agentTool (aT^3) development environment [4] to support the systematic design of OMAS by using our compositional approach. We are also interested in offline exploration of alternative designs created using different service providers and verifying them for robustness, flexibility and efficiency. Having predictive data on the quality of alternate designs will help designers choose the best service providers for their applications.

References

1. F.M.T. Brazier, et al., DESIRE: Modelling Multi-Agent Systems in a Compositional Formal Framework. IJCIS, 1997. 6(1): p. 67-94.
2. L. Cao, C. Zhang, and M. Zhou, Engineering Open Complex Agent Systems: A Case Study. Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on, 2008. 38(4): p. 483-496.
3. M. Cossentino, et al., A Holonic Metamodel for Agent-Oriented Analysis and Design, in Holonic and Multi-Agent Systems for Manufacturing. 2007. p. 237-246.
4. S.A. DeLoach. The agentTool III Project. [cited 2009; Available from: `http://agenttool.cis.ksu.edu/`].
5. S.A. DeLoach. OMACS: a Framework for Adaptive, Complex Systems. in Virginia Dignum (ed.) Multi-Agent Systems: Semantics and Dynamics of Organizational Models. IGI Global: Hershey, PA. (2009).
6. S.A. DeLoach, W.H. Oyenan, and E. Matson, A capabilities-based model for adaptive organizations. Autonomous Agents and Multi-Agent Systems, 2008. 16(1): p. 13-56.
7. A. Estefania, J. Vicente, and B. Vicente, Multi-Agent System Development Based on Organizations. Electronic Notes in Theoretical Computer Science, 2006. 150(3): p. 55-71.
8. J. Ferber, O. Gutknecht, and F. Michel, From Agents to Organizations: An Organizational View of Multi-agent Systems, in Agent-Oriented Software Engineering IV. 2004. p. 443-459.
9. J.C. Garcia-Ojeda, et al. O-MaSE: A Customizable Approach to Developing Multiagent Development Processes. in The 8th International Workshop on Agent Oriented Software Engineering 2007: p. 1-15.
10. M.P. Huget, Representing Goals in Multiagent Systems, in Proc. 4th Int'l Symp. Agent Theory to Agent Implementation. 2004. p. 588-593.
11. M.N. Huhns and M.P. Singh, Service-oriented computing: key concepts and principles. Internet Computing, IEEE, 2005. 9(1): p. 75-81.
12. M.N. Huhns, et al., Research Directions for Service-Oriented Multiagent Systems. IEEE Internet Computing, 2005. 9(6): p. 65-70.
13. N.R. Jennings, An agent-based approach for building complex software systems. Commun. ACM, 2001. 44(4): p. 35-41.

14. T. Juan, A. Pearce, and L. Sterling, ROADMAP: extending the gaia methodology for complex open systems, in Proceedings of the first international joint conference on Autonomous agents and multiagent systems: part 1. 2002, ACM: Bologna, Italy. p. 3-10.
15. C. Luo and S.X. Yang. A real-time cooperative sweeping strategy for multiple cleaning robots. in Intelligent Control, 2002. Proceedings of the 2002 IEEE International Symposium on. 2002 : p. 660-665.
16. M. Miller, A Goal Model for Dynamic Systems. Mastersthesis, Kansas State University, April 2007.
17. W.H. Oyenan and S.A. DeLoach, Design and Evaluation of a Multiagent Autonomic Information System, in Proceedings of the 2007 IEEE/WIC/ACM International Conference on Intelligent Agent Technology. 2007.
18. L. Padgham and M. Winikoff, Prometheus: A Methodology for Developing Intelligent Agents, in Agent-Oriented Software Engineering III. 2003. p. 174-185.
19. L.E. Parker, ALLIANCE: an architecture for fault tolerant multirobot cooperation. Robotics and Automation, IEEE Transactions on, 1998. 14(2): p. 220-240.
20. S.J. Harmon, et al., Leveraging Organizational Guidance Policies with Learning to Self-Tune Multiagent Systems, in The Second IEEE International Conference on Self-Adaptive and Self-Organizing Systems. 2008.
21. M. Sims, D. Corkill, and V. Lesser, Automated organization design for multi-agent systems. Autonomous Agents and Multi-Agent Systems, 2008. 16(2): p. 151-185.
22. A. van Lamsweerde, et al., The KAOS Project: Knowledge Acquisition in Automated Specification of Software. Proceedings AAAI Spring Symposium Series, 1991: p. 59-62.
23. I.A. Wagner, et al., Cooperative Cleaners: A Study in Ant Robotics. Int. J. Rob. Res., 2008. 27(1): p. 127-151.
24. M.F. Wood and S.A. DeLoach, An overview of the multiagent systems engineering methodology, in First international workshop, AOSE 2000 on Agent-oriented software engineering. 2001.
25. F. Zambonelli, N.R. Jennings, and M. Wooldridge, Developing multiagent systems: The Gaia methodology. ACM Trans. Softw. Eng. Methodol., 2003. 12(3): p. 317-370.
26. C. Zhong and S.A. DeLoach, An Investigation of Reorganization Algorithms, in International Conference on Artificial Intelligence (IC-AI'2006). 2006, CSREA Press: Las Vegas, Nevada.