BY Bipin Sir, YBN University, Ranchi

ENGINEERING

BY Bipin Sir, YBN University, Ranchi

Exploiting Reusable Organizations to Reduce Complexity in Multiagent System Design

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 incomplex interactions with each other and their environment. Consequently, providing a correct and effective design for such systems is a difficult task. Toreduce 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 [9, 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. 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 for developing.

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 _*G,Ev, parent, precedes, triggers, root*_ where:

• *G: set of organizational goals (where the set GL represent the leaf goals).*

• *Ev: set of events.*

• *parent:* $G \rightarrow G$ *; defines the parent goal of a given goal.*

• *precedes:* $G \rightarrow 2G$ *; indicates all the goals preceded by a given goal.*

• *triggers:* $Ev \rightarrow 2G \times G$*;* _$g1, g2$_ $\in triggers(e)$ *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 *_R, P, participants_* where:

• *R: set of internal and external roles*

• *P: set of protocols*

• *participants:P* → 2*R*×*R; indicates the set of role pairs connected by a protocol*

Finally, we define a multiagent organization *org* as a tuple *_GM,RM, achieves, INCP,OUTCP_* where:

• *GM: Goal Model*

• *RM: Role Model*

• *achieves: R* → 2*GL ; 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)*

*

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 ServiceModel

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 possesses a *connector* that is used to connect connection points exposed by 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 cp1 is an exit connection point from org1 using op and org2 exposes a connection point cp2 providing op. This composition is denoted *org*1 _*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 org1 using an operation op, we have:

*org*1 _*op org*2 = (...((*org*1 _*cp*1,*op org*2) _*cp*2,*op org*2) _... ... _*cpn*,*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 = _*GM*1,*RM*1, *achieves*1, *INCP*1,*OUTCP*1_, *org*2 = _*GM*2,*RM*2, *achieves*2, *INCP*2,*OUTCP*2_ , an operation *op* and two connection points *cp*1 from org1 and *cp*2 from org2 such that *cp*1 *uses op* and *cp*2 *provides op*. Given that *org*3 = _*GM,RM, achieves, INCP,OUTCP*_, such that *org*3 = *org*1 _*cp*1,*op org*2, we define the composite organization org3 in the next subsections. 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 = _*G*1,*Ev*1, *parent*1, *precedes*1, *triggers*1,*GR*_, and *GM*2 = _*G*2,*Ev*2, *parent*2, *precedes*2, *triggers*2,*GR*_, we define the composite goal model *GM* = _*G,Ev, parent, precedes, triggers, root*_ such that:

root = *GR*, **G** = *G*1 ∪ *G*2, ***Ev*** = *Ev*1 ∪ *Ev*2,

parent: $\forall g \in G$, $parent(g) = parent1(g) \cup parent2(g)$,

precedes: $\forall g \in G$, $precedes(g) = precedes1(g) \cup precedes2(g)$,

triggers: $\forall e \in Ev$, *triggers(e)*=

= ⎭
⎧⎪⎫
⎪⎩

*triggers*1(*e*) ∪ *triggers*2(*e*) if *e* = *op.event*,

*triggers*1(*e*) ∪ *triggers*2(*e*) ∪ {(*cp*1.*goal*, *cp*2.*goal*)}

−{(*cp*1.*goal*,∅), (∅, *cp*2.*goal*)} if *e* = *op.event*.

Note that cp1.goal is an exit goal in GM1 and cp2.goal is an entry goal in GM2.

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 = _*R*1, *P*1, *participants*1_, *RM*2 = _*R*2, *P*2, *participants*2_ , let e1 and e2 be two external roles such that (*cp*1.*role*, *e*1) ∈ *participants*1(*op.protocol*) and (*e*2, *cp*2.*role*) ∈ *participants*2(*op.protocol*), where cp1.role is an exit role in RM1 and cp2.role is an entry role in RM2. We define *RM* = _*R*, *P*, *participants*_ such that:

R = *R*1 ∪ *R*2 − {*e*1, *e*2}, ***P*** = *P*1 ∪ *P*2,

participants: ∀*p* ∈ *P*, *participants(p)*=

=

*participants*1(*p*) ∪ *participants*2(*p*) if *p* _= *op.protocol*,

*participants*1(*p*) ∪ *participants*2(*p*) ∪ {(*cp*1.*role*, *cp*2.*role*)}

−{(*cp*1.*role*, *e*1), (*e*2, *cp*2.*role*)} if *p* = *op.protocol*. 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 org3, we need to define the *achieves* function along with the connection points. The *achieves* function is defined as:

achieves(r) = *achieves*1(*r*) ∪ *achieves*2(*r*), ∀*r* ∈ *R*.

The sets of entry and exit connection points exposed by org3 are:

INCP = *INCP*1 ∪ *INCP*2 − {*cp*2}.

OUTCP =*OUTCP*1 ∪ *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 [7] 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 [16, 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 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): *_CleanFloor, Janitor_*.

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 points *_CleanFloor, Janitor_* from *cram* and *_DivideArea,Leader_* from *cleaning* respectively. We have:

cleaning = *_gm svc, rm svc, achieves svc, incp svc, outcp svc_* ,
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 = {}*.

cram = *_gm app, rm app, achieves app, incp app, outcp app_*,
where goal model *gm app*, rolemodel *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 _clean cleaning = *cram _cp Janit,clean cleaning* = *cram clean*,

such that ***cram clean*** = *_gm, rm, achieves, incp, outcp_* , where:

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

incp = *incp app* ∪ *incp svc*− *{cp Lead}* = *{}*,

outcp = *outcp app* ∪ *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.