Proceedings of the 15th Canadian Conference on Artificial Intelligence
Calgary, Alberta, Canada. May 27-29, 2002

Modeling Organizational Rules in the Multiagent Systems Engineering Methodology

Scott A. DeLoach

Department of Computing and Information Sciences, Kansas State University
234 Nichols Hall, Manhattan, KS 66506
`sdeloach@cis.ksu.edu`

Abstract. Recently, two advances in agent-oriented software engineering have had a significant impact: the identification of interaction and coordination as the central focus of multiagent systems design and the realization that the multiagent organization is distinct from the agents that populate the system. This paper presents detailed guidance on how to integrate organizational rules into existing multiagent methodologies. Specifically, we look at the Multiagent Systems Engineering models to investigate how to integrate the existing abstractions of goals, roles, tasks, agents, and conversations with organizational rules and tasks. We then discuss how designs can be implemented using advanced as well as traditional coordination models.

1 Introduction

Over the last few years, two conceptual advances in agent-oriented software engineering have had a significant impact on our approach toward building multiagent systems. The first of these was identification of interaction and coordination as the central focus of multiagent systems design. That is, interaction and coordination play a central role in the analysis and design of multiagent systems and makes the multiagent approach significantly different from other approaches towards building distributed or intelligent systems. This realization lead to several new methodologies for building multiagent systems that focused on the interaction between agents as the critical design aspect. Several agent-oriented methodologies fit this form including MaSE [3], Gaia [10], and MESSAGE [7].

The second, more recent advancement is the division of the agents populating a system from the system organization [11]. While agents play roles *within* the organization, they do not constitute the organization. The organization itself is part of the agent's environment and defines the social setting in which the agent must exist. An organization includes *organizational structures* as well as *organizational rules,* which define the requirements for the creation and operation of the system. These rules include constraints on agent behavior as well as their interactions. There are separate responsibilities for agents and organizations; the organization, not the agents, should be responsible for setting and enforcing the organization rules.

Organizational design has many advantages over traditional multiagent systems design methods. First, it defines a clean separation between the agent and the organization in which the agent works, which in turn simplifies each design. In traditional agent-oriented approaches, the rules that govern interaction must be incorporated into the agents themselves, thus intertwining the organizational design in various agent designs. Secondly, separating the organization from the agent allows the developer to build a separate organizational structure that can enforce the organizational rules. This is especially critical in open systems where we do not know the intent of the agents working within the system.

While these advances are rather recent, there have been some discussions on how to incorporate them into existing multiagent systems methodologies. For instance, there is a proposal to modify the Gaia multiagent systems methodology to incorporate the notion of social laws [12]. Other approaches view the organization as a separate *institutional agent* [9]. However, these proposals have been made at a high level and do not provide concrete guidance on how to use existing analysis and design abstractions with advanced coordination models and organizational concepts. Also, the advent of more powerful coordination models, such as hybrid coordination media, have allowed us to imagine new ways of implementing organization rules. With these advanced models, we can now embed organizational rules in the coordination media instead of implementing them internal to the individual agents [1].

The goal of this paper is to present more detailed guidance on how to integrate organizational rules into existing multiagent methodologies. Specifically, we will look at the Multiagent Systems Engineering (MaSE) analysis and design models to investigate how to integrate the existing abstractions of goals, roles, tasks, agents, and conversations with organizational rules. We will also briefly take a look at how we can use advanced coordination models to implement multiagent systems that separate agents from the organizational rules that govern them. We believe that extending existing conversation-based multiagent analysis and design approaches with organizational rules is a major step toward building coherent, yet adaptive multiagent systems in a disciplined fashion. While one might be tempted to simply throw out the concept of conversations altogether in favor of some of the more powerful models being proposed, we resist that urge for two basic reasons. First, conversation-based approaches are widely understood and provide an easily understandable metaphor for agent-to-agent communication. Second, conversation-based approaches have shown that they are verifiable and give designers some measure of system coherence [5]. Using the full power of these coordination models without restraint could lead to multiagent system designs that are not understandable, verifiable, or coherent.

In Section 2, we discuss how to model organizational rules MaSE. In Section 2.1, we look at the analysis phase where we add the notion of organizational rules to the existing MaSE analysis models. In Section 2.2 we show how to map the various analysis artifacts, including organizational rules, into an enhanced design model that explicitly models the organization through the notion of organizationally based tasks. Finally, in Section 3 we show how these organizational tasks might be implemented. We end with a discussion of our results and conclusions in Section 4.

2 Modeling Organizational Rules In MaSE

In this section we show how we have extended the MaSE analysis and design phases to take advantage of the concept of organizational rules. In the analysis phase, we add a new model, the organizational model, to capture the organizational rules themselves, while in the design phase, we introduce the concept of organizationally-based tasks to carry out specific tasks that are part of the organization and do not belong to a specific agent. These tasks are often used to implement and enforce the organizational rules defined during analysis.

Throughout this paper, we will use the conference management example as defined in [11]. The conference management system is an open multiagent system supporting the management of various sized international conferences that require the coordination of several individuals and groups. There are five distinct phases in which the system must operate: submission, review, decision, and final paper collection. During the submission phase, authors should be notified of paper receipt and given a paper submission number. After the deadline for submissions has passed, the program committee (PC) has to review the papers by either contacting referees and asking them to review a number of the papers, or reviewing them themselves. After the reviews are complete, a decision on accepting or rejecting each paper must be made. After the decisions are made, authors are notified of the decisions and are asked to produce a final version of their paper if it was accepted. Finally, all final copies are collected and printed in the conference proceedings. The conference management system consists of an organization whose membership changes during each stage of the process (authors, reviewers, decision makers, review collectors, etc.). Also, since each agent is associated with a particular person, it is not impossible to imagine that the agents could be coerced into displaying opportunistic, and somewhat unattractive, behaviors that would benefit their owner to the detriment of the system as a whole. Such behaviors could include reviewing ones own paper or unfair allocation of work between reviewers, etc.

2.1 The Analysis Phase

The purpose of the MaSE analysis phase is to produce a set of roles whose tasks describe what the system has to do to meet its overall requirements. A role describes an entity that performs some function within the system. In MaSE, each role is responsible for achieving, or helping to achieve specific system goals or sub-goals. Because roles are goal-driven, we also chose to abstract the requirements into a set of goals that can be assigned to the individual roles. Our approach is similar to the notions used in the KAOS [6]. The overall approach in the MaSE analysis phase is fairly simple. Define the system goals from a set of functional requirements and then define the roles necessary to meet those goals. While a direct mapping from goals to roles is possible, MaSE suggests the use of use cases to help validate the system goals and derive an initial set of roles. As stated above, the ultimate objective of the analysis phase is to transform the goals and use cases into roles and their associated tasks since they are forms more suitable for designing multiagent systems. Roles form the foundation for agent classes and represent system goals during the design phase, thus the system goals are carried into the system design. To support organizational rules, the MaSE analysis phase was extended with an explicit organizational model, which is developed as the last step in the analysis phase and is defined using concepts from the role and ontology models.

Role Model

Due to space limitations, we will skip the goal and use case analysis for the conference system example and jump right to the role model. The MaSE role model depicts the relationships between the roles in the conference management system, as shown in Figure 1. In Figure 1, a box denotes each role while a directed arrow represents a protocol between roles, with the arrows pointing away from the initiator to the responder. Notice that while we referred to the PC chair and PC members in the problem description, we have intentionally abstracted out the roles played by those typical positions into partitioning, assigning reviews, reviewing papers, collecting reviews, and making the final decision. As we will see later, this provides significant flexibility in the design phase. The system starts by having authors submit papers to a paper database (PaperDB) role, which is responsible for collecting the papers, along with their abstracts, and providing copies to reviewers when requested. Once the deadline has past for submissions, the person responsible partitioning the entire set of papers into groups to be reviewed (the Partitioner role) asks the PaperDB role to provide it the abstracts of all papers. The Partitioner partitions the papers and assigns them to a person (the Assigner) who is responsible for finding *n* reviewers for each paper. Once assigned a paper to review, a Reviewer requests the actual paper from the PaperDB, prepares a review, and submits the review to the Collector. Once all (or enough) of the reviews are complete, the Decision Maker determines which papers should be accepted and notifies the authors.

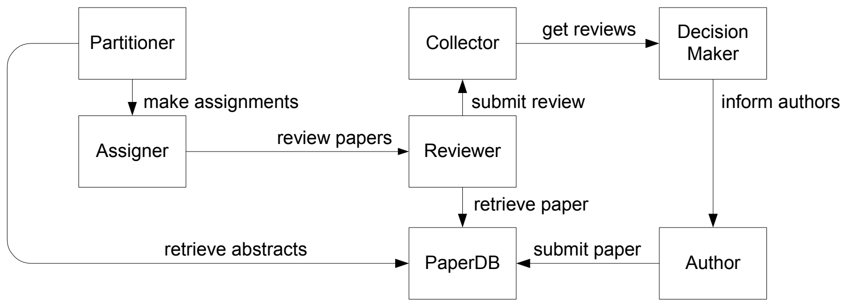

Figure 1. Role Model for Conference Management System

Thus, we have identified seven explicit roles. However, in MaSE, we do not stop at simply identifying the roles, we also identify the tasks that the roles must perform in accomplishing their goals. Therefore, a more detailed version of the conference management system role model is shown in Figure 2. In MaSE, we have extended the traditional role model by adding the tasks (shown using ellipses attached to each role). Generally, each role performs a single task, whose definition is straightforward and documented in a concurrent task diagram (not discussed here due to space limitations), which define agent behaviour and interaction via finite state machines. However, some roles, such as the Paper DB or Reviewer roles have multiple tasks. For instance, the Paper DB role has three tasks: Collect Papers, Distribute Papers, and Get Abstracts. While the tasks are related, they are distinct and are thus modelled separately. The Collect Papers task accepts papers, ensures they are in the right format and meet all the eligibility requirements. The Get Abstracts task extracts the abstract from submitted papers and sends them to a Partitioner. The Distribute Papers task simply distributes accepted papers to the appropriate Reviewers when requested.

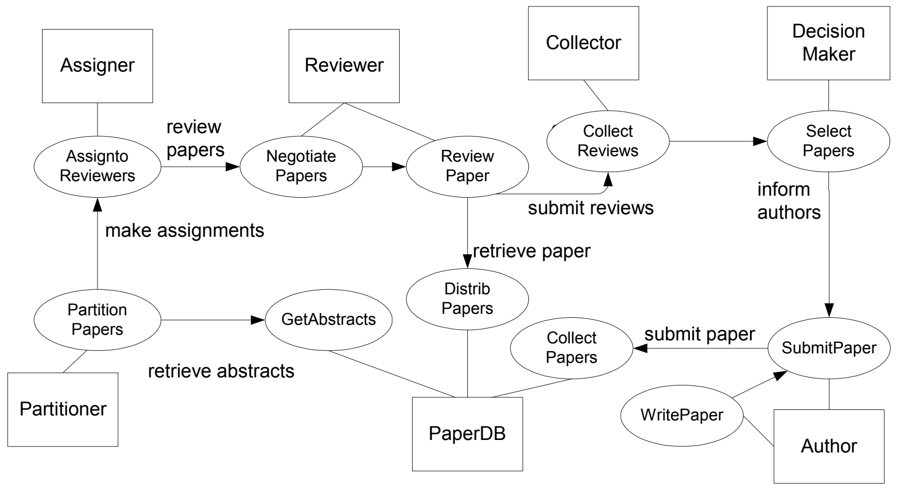

Figure 2. Expanded MaSE role model

Ontology Model

The next step in the MaSE analysis phase is to develop an Ontology Model, which defines the data types and their relationships within the system [4]. Figure 3 shows an ontology model for the conference review system. The ontology is focused around the central data type, a paper, each with an associated abstract and a set of reviews. Given the ontology, we can talk about the reviews a paper has received *paperReview(p)* or a paper's abstract *paperAbstract(p)*, etc. There are also constraints placed on the data via the ontology. For instance, each abstract must have exactly one paper and each paper must have exactly one abstract. Also, a review can only exist on a single paper, while a paper may have any number of reviews on it (including none). Thus several organizational constraints can be defined in the ontology itself. Using the ontology model, we can extract a number of functions to describe the data in our system. The functions and their resulting types for the conference management system are shown in Table 1. These functions can be used in conjunction with protocol functions to describe many relationships, as we will see in the next section.

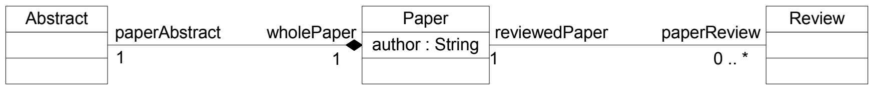

Figure 3. Conference Management Ontology

Table 1. Functions Derived from Ontology

paperReview(p)	{Review}
paperAbstract(p)	Abstract
reviewedPaper(r)	Paper
wholePaper(a)	Paper

Organizational Model

In our previous treatments of MaSE, we would go to the design phase at this point. However, this is precisely the point at which we can effectively begin to identify organizational rules. By definition, organizational rules define constraints on agent behavior and their interactions. At the analysis level, this equates to restrictions on the roles an agent may play or how an agent may interact with other agents. To state these rules in a formal manner, we must have a language based on analysis artifacts. This language is defined by the role model, the ontology model, and a set of meta-predicates.

We can use the protocols and roles defined in the Role Model to describe how the system will operate, which will be very useful when defining organizational rules. For instance, we can refer to an agent playing a particular role. We annotate this using a data type like notation, for instance, *r:Reviewer*, which states that agent r is of type (i.e., plays the role of a) Reviewer. Thus if we wanted to state that the agent making final decisions cannot be an author of any papers for the conference, we could say

$$\forall \texttt{ a:Author, d:DecisionMaker } \quad d \neq a$$

Another way to state the same requirement would be through the use of a meta-predicate *Plays*, which states that a particular agent plays a particular role. Therefore, we could state the same requirement as

$$\forall \texttt{ a:Agent } \quad \neg(\texttt{Plays(a, Author)} \wedge \texttt{Plays(a, DecisionMaker)})$$

The use of meta-predicates can be useful in stating requirements. For instance, if we want all agents in the system to be authors, we can simply state, $\forall$ `a: Agent Plays(a, Author)`, which is simpler than using the data type notation.

We will also need to refer to the relationships between agents (or roles) in the system. Since the only relationships we have defined in MaSE are via protocols, we use protocol instances to specify relationships. We refer to a protocol between two agents as *prototocolName(initiator, responder, data)*, which states that a protocol exists between two roles, initiator and responder, and concerns a particular piece of data. The initiator and responder must be capable of playing the appropriate roles and the data must refer to data passed between roles via the protocol. Thus the expression, *reviewPapers(a, r, p)*, states that a protocol named *reviewPapers* exists between the roles *a* and *r* (involving a paper, *p*), which must be capable of playing the Assigner and Reviewer roles respectively. Thus if we wanted to state that a Reviewer can only review papers for one Assigner, we could make the following rule.

```
∀ a1, a2:Assigner, r:Reviewer, p1, p2:Paper
      reviewPapers(a1, r, p1) ∧ reviewPapers(a2, r, p2) ⇒ a1 = a2
```

Although we can state some requirements using only concepts from the role model, there are other times where we must relate roles and their relationships based on particular data in the system. For instance, in the conference management system we are interested in the relationships between roles based on the papers they submit, review, or collect. Thus we must be able to talk about the data in the system as well, which is defined by the ontology model.

In the original paper describing the conference management system in terms of organizational rules [11], the authors defined seven organizational rules. While the authors stated the rules using a formal notation, there was no real definition of how the rules mapped to the artifacts of their analysis and design. Here we will redefine them using the notation presented above based on the role and ontology models. The rules as originally presented are shown below using the temporal operators as defined in Table 2.

```
1.  ∀p : #(reviewer(p)) ≥ 3
2.  ∀i, p : Plays(i, reviewer(p)) ⇒ ○ □ ¬Plays(i, reviewer(p))
3.  ∀i, p : Plays(i, author(p)) ⇒ □ ¬Plays(i, reviewer(p))
4.  ∀i, p : Plays(i, author(p)) ⇒ □ ¬Plays(i, collector(p))
5.  ∀i, p : participate(i, receivePaper(p))
                             ⇒ ◇ initiate(i, submitReview(p))
6.  ∀i, p : participate(i, receivePaper(p))
                             𝓑 initiate(i, submitReview(p))
7.  ∀p : [submittedReviews(p) > 2] 𝓑 initiate(chair, decision(p))
```

The first rule states that there must be at least three reviewers for each paper (# is cardinality) while rule two keeps a reviewer from reviewing the same paper more than once. Rules three and four attempt to limit selfish agent behaviour by ensuring that a paper author does not review or collect reviews of his or her own paper. The last three rules describe appropriate system operation. Rule five states that if a paper is received, it should eventually be reviewed. Rule six requires that a paper must actually be received before a review can be submitted on it while rule seven requires that there be at least two reviews before a paper can be accepted or rejected.

Table 2. Temporal Operators

$\bigcirc\ \varphi$	φ is true next
$\Box\ \varphi$	φ is always true
$\Diamond\ \varphi$	φ is eventually true
$\varphi\ \mathcal{B}\ \phi$	φ is true before ϕ is true

The first organizational rule states that each paper should have at least three reviewers. While we might be tempted to use the ontology model to say that each paper should have three or more reviews, this does not adequately capture the requirement. What we want to state is that three agents, playing the part of reviewers should be assigned to each paper, which requires more knowledge than is in the ontology. It requires that we combine relationships and data definitions from the ontology with relationships (defined by protocols) defined in the role model. What we need to say is that for a given paper, p, there must be at least three reviewers assigned. Since the review assignment process is accomplished via the reviewPapers protocol between the Assigner role and the Reviewer role, there must be three instances of that protocol for paper p. Thus we can state the requirement as

```
∀ p:Paper, a:Assigner, r:Reviewer #{r | reviewPapers(a,r,p)} ≥ 3
```

The second rule keeps a reviewer from reviewing the same paper more than once. While this appears be subsumed by our first rule, in fact it is not. Our first rule states that we must have three unique reviewers, but it does not stop them from submitting multiple reviews on the same paper. To accomplish this, we must limit the number of *submitReview* protocols that can exist between the Reviewer role and any Collector roles for a given paper. This is formalized as

```
∀ r1, r2:Review, r:Reviewer, c1, c2:Collector
    submitReview(r,c1,r1) ⇒ ○ □ (¬submitReview(r,c2,r2)
                               ∨ reviewedPaper(r1) ≠ reviewedPaper(r2))
```

The next two rules (three and four) limit selfish agent behavior by ensuring that a paper author does not review or collect reviews of his or her own paper. The first of these rules states that an author may not review his or her own paper while the second does not let the author acts as a collector of the reviews on his or her paper. There two approaches to modeling an author. As defined in [11], we could assume that the author is the one who submits the paper and identify the author as the role that submits the paper to the PaperDB role via the submitPaper protocol. The second approach would be to use the author attribute of the paper object and compare it to the reviewer. This would require the ability to identify the name of the Reviewer role, which would require an extension to the MaSE role model. Therefore, we will use the first approach and define the third rule as

```
∀ a:Author, d:PaperDB, p:Paper, s:Assigner, r:Reviewer, c:Collector,
  r1:Review  submitPaper(a,d,p) ⇒
               ¬(submitReview(r,c,r1) ∧ a = r ∧ r1 = paperReview(p))
```

Likewise, the fourth rule ensures the author does not participate as a collector.

```
∀ a:Author, d:PaperDB, p:Paper, r:Reviewer, c:Collector, r1:Review
        submitPaper(a,d,p) ⇒ ¬(submitReview(r,c,r1)
                                      ∧ a = c ∧ r1 = paperReview(p))
```

Finally, the last three rules define the way in which the system should operate. Rule five simply requires that if a paper is submitted via the sumbitPaper protocol, a review should eventually be submitted to a collector by via the submitReview protocol. This rule is state straightforwardly using the appropriate temporal operator.

```
∀ a:Author, d:PaperDB, p:Paper, r:Reviewer, c:Collector, r1:Review
       submitPaper(a,d,p)
                   ⇒ ◇ submitReview(r,c,r1) ∧ r1 = paperReview(p))
```

Rule six, requiring the paper be submitted before it can be reviewed can be defined as

```
∀ a:Author, d:PaperDB, p:Paper, r:Reviewer, c:Collector, r1:Review
    submitPaper(a,d,p) 𝓑 (submitReview(r,c,r1) ∧ r1 = paperReview(p))
```

Finally, the last rule requiring at least two submitted reviews per paper before a decision can be rendered can be encoded as

```
∀ r: Reviewer, c:Collector, r1:Review, m:DecisionMaker, a:Author,
p:Paper   #{r1 | submitReview(r,c,r1) ∧ r1 = paperReview(p)} ≥ 2
                                                 𝓑 (informAuthor(m,a,p))
```

During the analysis phase, these organizational rules are collected and defined in terms of the ontology and role model; however, they are integrated into the overall system design in the next stage. It is at this point that the designer must decide how to monitor or enforce these rules. As we will see, the rules can be assigned to a particular agent in the design or they can be implemented via conversational, monitoring, or enforcement tasks as organizational tasks.

2.2 The Design Phase

The initial step in the MaSE design phase is to define agents from the roles defined in the analysis phase. The product of this phase is an Agent Class Diagram, as shown in Figure 4, which depicts the overall agent system organization defined by agent classes and conversations between them. An agent class is a template for a type of agent in the system and is analogous to an object class in object-orientation while an agent is an instance of an agent class. During this step, agent classes are defined in terms of the roles they will play and the conversations in which they must participate. In the diagram, boxes denote agent types (with the roles it plays listed under its name) while directed arrows represent conversations between agent types with a similar semantics to role model protocols.

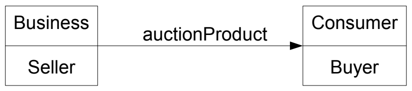

Figure 4. Agent Class Diagram

In this paper we extend the Agent Class Diagram with organizationally based tasks, which is a new concept that allow us to model aspects of the organization independently of the agents. Organizationally based tasks are tasks that are assigned to the organization (as opposed to a particular agent) and can be used to implement social tasks, monitor system and individual agent behavior, and enforce organizational and security rules. An example of an organizationally based task is shown in Figure 5. The Seller and Buyer boxes are agents while the rounded rectangle denotes the organization. The ellipse in the organization box is an organizationally based task, Auction, which was derived from a task belonging to a role in the role model. In the initial step of the design phase, the designer determines the roles each agent type will play as well as which roles (and tasks) will be relegated to the organization. The designer may also create new organizationally based tasks to implement and enforce the organizational rules defined during the analysis phase.

In the remainder of this section, we take our analysis of the conference management system, including the organizational rules, and show how it can be developed into a number of different designs using organizationally-based tasks in conjunction with conventional MaSE Agent Class Diagrams. The goal here is to show a number of different options that are available with the notion of organizationally based tasks, not to advocate a particular approach as being necessarily better in all instances.

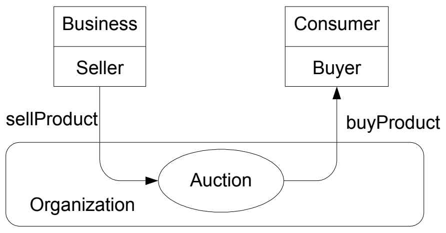

Figure 5. Organizationally-Based Task

Design 1 - Traditional

Traditional multiagent design approaches, as advocated in [3], might result in the design shown in Figure 6. In this design, various roles are combined into agents. For instance, the PC Chair agent plays the Partitioner, Collector, and Decision Maker roles while the PC Member agent plays both the assigner and reviewer roles. Outside of author agents, the only other agent is the DB agent, which provides an interface to the database containing the papers, abstracts, and author information, etc.

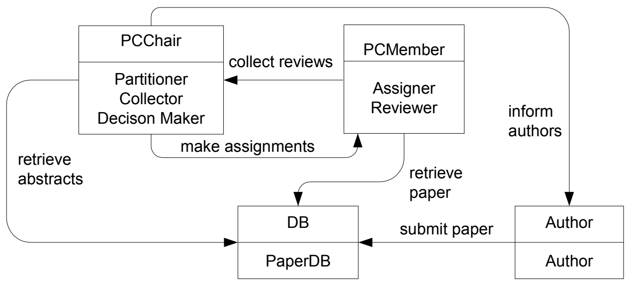

Figure 6. Traditional design

Unfortunately, the traditional multiagent design described above does not provide the separation of agent tasks from social, or organizational, tasks, which is desirable for extensible, open multiagent systems [2]. To ensure the organizational rules are enforced, we must interweave the organizational rules into the individual agents themselves. For example, the only place we can check to ensure that at least two reviews were completed before the decision to accept or reject a paper was made (rule 7) is in the PC Chair agent itself. This forces us to rely on self-policing agents, which, if we assume the possibility of self-interested agents, is a less than desirable approach to ensuring the enforcement of organizational rules.

Design 2 – Assigning Tasks to the Organization

As advocated by some [2], the appropriate place to monitor and enforce organizational rules is in the organization itself. Thus, using the same analysis, we have created a new design that uses organization-based tasks to implement the PaperDB and Collector roles. Figure 7 shows the details of this new design. Notice that the tasks of the PaperDB and the Collector roles have been assigned to the organization. In effect, their tasks become part of the organization as organizationally based tasks.

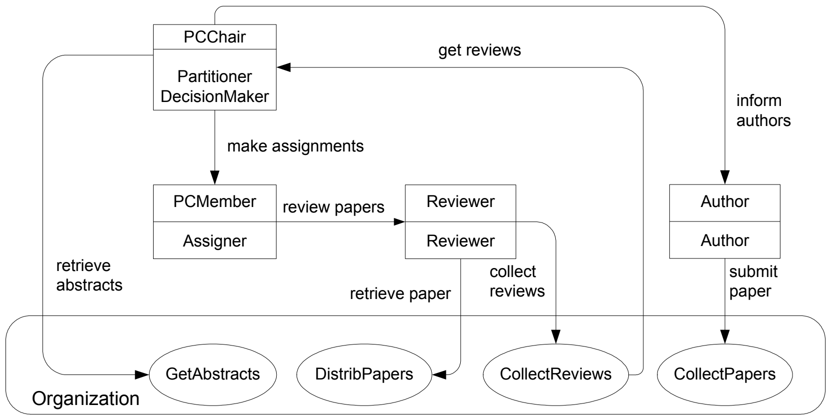

Figure 7. Design with explicit tasks

By being part of the organization, the Get Abstracts, Distribute Papers, Collect Papers, and Collect Reviews tasks can more easily support the conference management organizational rules. This is because the information collected and used by these tasks can easily be shared through a common database. For instance, The Distribute Papers task can enforce rule 3 (an author cannot review his or her own paper) by simply checking the reviewer against the paper author. Likewise, the Collect Reviews task can monitor rule 5 (if a reviewer receives a paper, he or she must eventually submit a review) and send warnings if reviews are not submitted in a timely fashion. The same task can also enforce rule 6 (the paper must be received by a reviewer before the review is submitted) by not accepting reviews until the paper has actually been requested, as well as rule 7 (there must be at least two reviews before the chair can make a decision) by only sending reviews once there are at least two of them. This design approach also allows the organizational rules to be updated without necessarily affecting the individual agent designs.

Design 3 – Designing New Organizational Roles

A third design that does not assign tasks from the role model to the organization is shown in Figure 8. However, we still use organizationally based tasks to monitor and enforce the organizational rules presented above. We do this by creating new tasks in the design to implement the organizational rules. For instance, in Figure 8, there are three organizational tasks (Monitor Num Reviews, Monitor Decisions, and Monitor Reviewers) that did not exist in the role model, but were added by the designer to monitor/enforce organizational rules 2, 3, and 7. The dashed line between the tasks and the conversations denote that the tasks *monitor* those conversations by executing when the conversations are started. These tasks may simply monitor the communication between agents and either display or log the information of interest. For instance, the Monitor Decision task might monitor the *inform author* conversations and log only those decisions that are made without the required number of reviews being made. Note that the Monitor Decision task would have access to this information via tuples shared by the Monitor Num Reviews task.

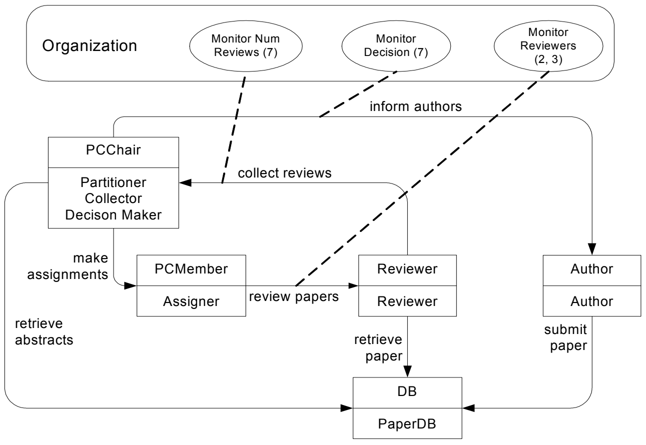

Figure 8. Design with monitoring/conversational tasks

A task that simply monitors a conversation is shown in Figure 9. In modelling monitoring tasks, we assume that the task receives a message before the agent on the other end of the conversation and must forward the message before the intended recipient can receive it. In Figure 9a this is shown by the *receive* event that initiates the transition from the start state. Once the message is received, the Monitor Decision task validates it (in this case, that it has had at least two reviews) and, if valid, passes the message along to the intended recipient.

We can use the same basic design as shown in Figure 8 but use tasks that do more than just monitor the conversations; they may actually interrupt the conversation or modify the data being passed between agents, thus providing correction either directly or indirectly with the offending agents. For example, Figure 9b defines a task that intercepts the *notice* message being sent to an author; if the correct number of reviews has not been accomplished, the task sends the PC Chair a message stating that the decision was invalid instead of forwarding the notice message on to the author.

Of course, a task that communicates directly with agents in a conversation forces the agents involved to be able to handle additional communication. Thus, the original *inform authors* conversation (from the viewpoint of the PC Chair) must be modified to work with this type of task. Specifically, the PC Chair's side of the conversation must be able to handle an *invalidDecision* message from the organization. Thus, in Figure 10, we have modified the conversation to accept the *invalidDecision* after sending the original notice. This is an example of the strength of using a conversation based design approach. Using conversations, it is possible to trace the sequence of possible messages through the system and thus automatically verify that all conversations and tasks are consistent and do not cause unwanted side effects such as deadlock.

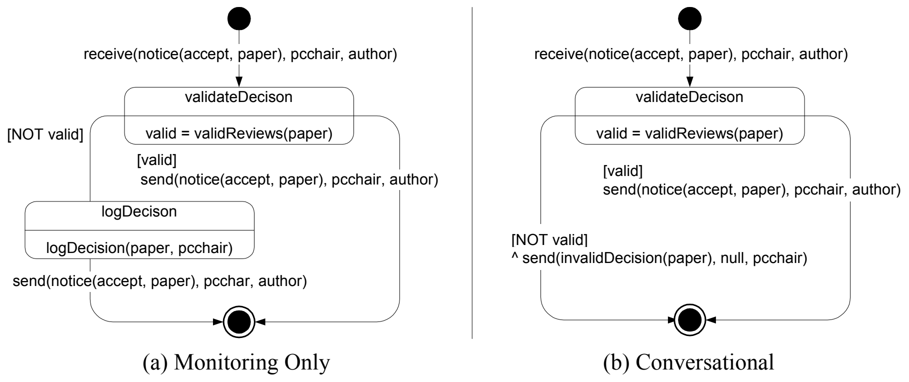
receive(notice(accept, paper), pcchair, author)
validateDecison
valid = validReviews(paper)
[NOT valid]
[valid]
send(notice(accept, paper), pcchair, author)
logDecison
logDecision(paper, pcchair)
send(notice(accept, paper), pcchar, author)
(a) Monitoring Only
receive(notice(accept, paper), pcchair, author)
validateDecison
valid = validReviews(paper)
[valid]
send(notice(accept, paper), pcchair, author)
[NOT valid]
^ send(invalidDecision(paper), null, pcchair)
(b) Conversational

Figure 9. Monitor Decision Task

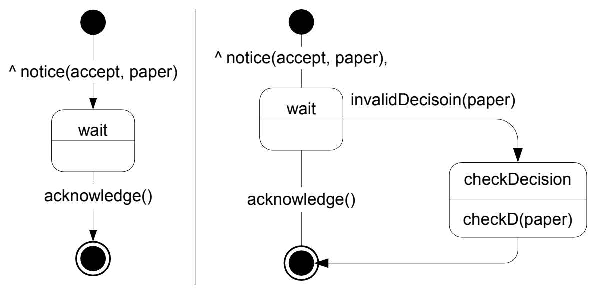

Figure 10. Inform Authors conversation (original & modified)

3 Implementation

Ideally, organization based tasks would be implemented using a coordination model that has equivalent structures, such as hybrid coordination media. *Hybrid* *coordination models* are data-centered coordination models that include (1) a logically centralize repository where the agents read and write data and (2) a set of *reactions* that are functions that react to, and can read and modify data in the data store [1] [8]. In a hybrid coordination media, the media itself has the ability to see the communication between agents and perform tasks in reaction to those communications. Thus we could easily model organizationally based tasks as reactions in hybrid coordination media. For role model tasks that are assigned to the organization, the hybrid model is ideal since the reaction is not under control of an individual agent, but is part of the organization itself and thus is started at system initialisation. Such tasks may include controlling the introduction of new agents in the system. Tasks that intercept messages and forwarding them on if they are valid, as well as those that just monitor messages are also easily implemented in hybrid models. The ability of reactions to read all data in the data store allows them to monitor messages and take action when necessary. For example, if authors are required to submit papers in PDF format, we could enforce this rule via a reaction that would automatically convert non-PDF formats to PDF; the reaction would simply extract any non-conforming papers and replace them with the appropriate PDF version.

While useful, such an advanced coordination models are not required to take advantage of an organizational design approach. While it might be less efficient, these designs could also be implemented using a more traditional message oriented middleware component. One approach would be to build an "organization" agent (or agents) that would handle all the organization tasks that would normally be assigned to reactions in a hybrid coordination media. Using this approach, all critical communications can be routed through organizationally based tasks to ensure the organizational rules are adhered to. Whether using a hybrid coordination media or organizational agents, the advantages based on separating organizational tasks rules from the agents would remain.

4 Results and conclusions

The goal of this paper was to present our approach toward integrating organizational rules within the MaSE methodology. To accomplish our goal, we extended the MaSE analysis phase with an explicit organizational model, which defines organizational constraints based on concepts defined in the role and ontology models. In the design phase, we extended the MaSE Agent Class Diagram with an explicit organization artifact, which contains its own organizationally based tasks. We also showed various approaches toward integrating the organizational rules defined in the analysis model. We also discussed various approaches to implementing organizational tasks including both hybrid coordination media as well as traditional message passing media.

While we originally developed MaSE to design closed multiagent systems, the incorporation of organizational rules moves it toward being useful for the analysis and design of open systems as well. While MaSE still requires specific coordination protocols, designers no longer have to rely on incorporating organizational rules into the agents themselves. The concept of organizational tasks provides a mechanism to allow agents to enter the system, monitor their behavior, and ensure compliance with organizational rules and protocols.

REFERENCES

[1] Cabri, G., Leonardi, L., and Zambonelli, F. Implementing Agent Auctions using MARS. Technical Report MOSAICO/MO/98/001.
[2] Ciancarini, P., Omicini, A., and Zambonelli, F. Multiagent System Engineering: the Coordination Viewpoint. Intelligent Agents VI. Agent Theories, Architectures, and Languages, 6th International Workshop (ATAL'99), Orlando (FL), May 1999, Proceedings. LNAI 1757, Springer-Verlag, 2000.
[3] DeLoach, S.A., Wood, M.F., and Sparkman, C.H. Multiagent Systems Engineering, The International Journal of Software Engineering and Knowledge Engineering, Volume 11 no. 3, June 2001.
[4] Dileo, J.M. Ontological Engineering and Mapping in Multiagent Systems Development. MS thesis, AFIT/GCS/ENG/02M-03. School of Engineering, Air Force Institute of Technology, Wright Patterson Air Force Base, OH, 2002.
[5] Lacey, T.H., and DeLoach, S.A. Automatic Verification of Multiagent Conversations. in Proceedings of the Eleventh Annual Midwest Artificial Intelligence and Cognitive Science Conference, pp. 93-100, AAAI Press, Fayetteville, Arkansas, April 2000.
[6] Letier, E. Reasoning about Agents in Goal-Oriented Requirements Engineering, Phd Thesis, Université Catholique de Louvain, Dépt. Ingénierie Informatique, Louvain-la-Neuve, Belgium, May 2001.
[7] MESSAGE: Methodology for Engineering Systems of Software Agents. Deliverable 1. Initial Methodology. July 2000. EURESCOM Project P907-GI.
[8] Omicini, A., Denti, E. From Tuple Spaces to Tuple Centres. Science of Computer Programming 41(3). Elsevier Science B. V., November 2001.
[9] Wagner, G. Agent-Oriented Analysis and Design of Organizational Information Systems. Proceedings of the 4th IEEE International Baltic Workshop on Databases and Information Systems, Vilnius, Lithuania, May 2000.
[10] Wooldridge, M., Jennings, N.R., and Kinny, D. The Gaia Methodology for Agent-Oriented Analysis and Design. Journal of Autonomous Agents and Multi-Agent Systems. Volume 3(3), 2000.
[11] Zambonelli, F., Jennings, N.R., and Wooldridge, M.J. Organisational Rules as an Abstraction for the Analysis and Design of Multi-Agent Systems. International Journal of Software Engineering and Knowledge Engineering. Volume 11, Number 3, June 2001. Pages 303-328
[12] Zambonelli, F., Jennings, N.R., Omicini, A., and Wooldridge M.J. Agent-Oriented Software Engineering for Internet Applications. Coordination of Internet Agents: Models, Technologies, and Applications, Chapter 13. Springer-Verlag, March 2001.