

An architecture for access control management in collaborative enterprise systems based on organization models

F.L. Gutiérrez Vela^{a,*}, J.L. Isla Montes^b, P. Paderewski Rodríguez^a, M. Sánchez Román^a,
B. Jiménez Valverde^a

^a *Department of Computer Languages and Systems, University of Granada, Spain*

^b *Department of Computer Languages and Systems, University of Cádiz, Spain*

Received 5 March 2006; received in revised form 31 August 2006; accepted 15 October 2006

Available online 19 December 2006

Abstract

One of the most important characteristics of current enterprise systems is the existence of collaborative processes where different users/subsystems communicate and cooperate in order to carry out common activities. In these processes, shared resources are often used and there are complex relationships between activities and users, so the definition and administration of different security levels (tasks, users, resources, etc.) is necessary.

In this article, we shall focus on an important dimension related to the security aspect of collaborative systems: access control. We shall use an organization model that considers the necessary elements to represent authorization and access control aspects in enterprise systems. This model is used in a service-oriented architecture (SOA) in order to facilitate the implementation of a service which is responsible for these important functions. Finally, we shall propose the use of a pattern definition language at a conceptual level to facilitate the modelling of the organizational structures of an enterprise system. We shall specify organization patterns that will help us define general models which can be applied in different situations.

© 2006 Elsevier B.V. All rights reserved.

Keywords: Collaborative system; Software architecture; Enterprise architecture; Role-based access control; Organization patterns

1. Introduction

Collaborative processes enable user groups to communicate, coordinate and cooperate in order to perform common activities. In this kind of system, there are a wide range of applications (e.g. applications for collaborative document sharing/editing, electronic learning or workflow management systems). This type of application is broadly used in enterprise systems [11]. Nowadays, companies build their services using the resources that are provided by new technologies and more specifically by the Web. It is normal to find that company business increases or changes when these technologies are incorporated.

* Corresponding author.

E-mail addresses: fgutierr@ugr.es (F.L. Gutiérrez Vela), jose Luis.isla@uca.es (J.L. Isla Montes), patricia@ugr.es (P. Paderewski Rodríguez), miguesr@ugr.es (M. Sánchez Román), beajv@ugr.es (B. Jiménez Valverde).

When the applications are set up on the Web, it is necessary to pay special attention to security aspects and although we normally only pay attention to security when transmitting information between servers and for user authentication, access control to activities and shared resources is also very important. In modern enterprise systems, there are an increasing number of collaborative processes and in such cases, it is necessary to include complex security policies within the definition of the company organizational structure. The organizational structure and the control policy for accessing resources and activities are two of the most dynamic elements in a company, and it is necessary to be able to define non-rigid organizational structures that are capable of quickly adapting to changes.

Ellis [2] proposed four basic requirements that a system should have if an access control subsystem is to be implemented:

- The mechanism should be simple.
- The mechanism must be unobtrusive to users. It should be naturally integrated with the rest of the system. The system modelling should not increase the complexity of the modelling elements.
- At any moment in the system's life, it should be easy to check authorization for any system resource access.
- The effects that access control causes on the rest of the system should be clear and easy to understand.

In our opinion, however, it is necessary to add a fifth requirement since the software systems implemented in current companies require dynamic software that can be evolved. Changes take place in the business itself, in the organization that supports it, in resource access policies, in the business rules which govern the processes and at any given moment, these changes should be propagated to the software systems so that they may continue to be effective. It is necessary to develop mechanisms for access control and security that may easily be adapted to the changes occurring both at the business level and in the access policies implemented.

The cost of system development in terms of changes or new requirements is a considerable part of the total cost. In many cases, security-related elements have been considered during the final development phases and this greatly increases the final cost. Since security aspects are very important and complex, it is necessary to analyse and model them from the first steps of development with models that facilitate their dynamism. In our opinion, one important requirement is the integration of security elements in the initial models which we use to describe system functionalities.

In this paper, we will show the integration of an access control model in a service-oriented architecture in order to develop applications for complex organizations (enterprises systems) with collaborative activities; this model facilitates dynamic aspect expression in a collaborative environment. In Section 2, we will analyse the necessary elements for efficient access control, and will present a role-based model. In Section 3, we will propose the implementation of an access control model as a set of Web services of a service-oriented architecture. In Section 4, we will then describe how the model used to describe a collaborative system includes the necessary elements for modelling system access control policies. In order to make modelling easier, in Section 5 we will propose a notation that can be used to define and apply organization patterns. Finally, we will focus on future work and our conclusions.

2. Access control model

Controlling access to resources and activities is a key element in system security and an important complement to the definition of the interaction between users and/or systems. The first work into access control modelling in collaborative systems was carried out by Shen and Dewan [17]. Using this as a basis, different models have been applied [20] and attempts have been made to adapt them to the characteristics of this kind of system.

The necessary requirements for collaborative systems access control models [20,2] may be summarized as follows:

- Access control models must be easy to use and transparent for end users.
- Access control-produced effects on the rest of the system must be clear and easy to understand.
- Access control models must allow us great expressiveness (taking into account aspects such as roles, execution tasks, access request data, etc.) and these models should enable us to specify complex access policies at different levels of detail.
- Models must be dynamic so as to enable specification, delegation, revocation and management of access policies in runtime (Meta Access Control).

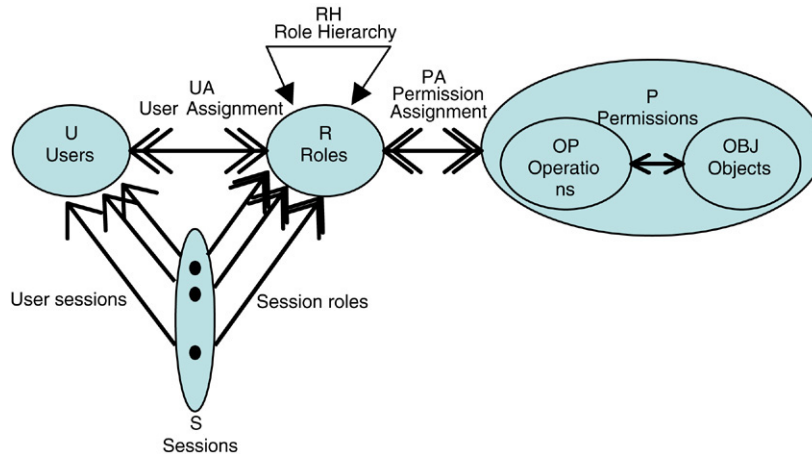


Fig. 1. RBAC concepts.

- Models must allow us to protect different levels of granularity on any type of information and resources. In other words, models must be able to provide a strong protection for shared environments.
- Context is an important element in collaborative work and something which access control models must have in order to establish access permissions. Models may grant access by considering the current context user.

Most of the previously used models were designed to analyse security aspects and system functionalities separately. One of the most used models is role-based access control (RBAC) by Sandhu et al. [18] which quickly emerged in the 1990s as a technology for managing and enforcing security in large-scale corporate systems. This model uses the concept of role instead of assigning privileges directly to the users. This facilitates the classification of privileges and enables a finer access control to resources.

The RBAC model is based on the definition of a series of elements and the relationships between them (Fig. 1). This model describes a user group that can play a series of roles and carry out operations. Users need to use objects as resources in each operation.

The following relationships appear between these four elements:

- Relationships between users and roles: this relationship models the different roles a user can play.
- Operation set that can be performed on each object: the elements of this relationship are called permissions.
- Relationships between permissions and roles: these relations describe each user's permissions to use specific objects when they are playing a certain role.

The RBAC model includes a sessions set. Each session is a mapping between a user and any possible roles. When users establish a session, they can activate roles that they have been assigned. Each session is associated with a single user although each user can be involved in one or more sessions. The permissions available to the user are those assigned to the roles that are activated for all the user's sessions, independently of the sessions established by other system users.

It is also possible to include a role hierarchy so that generalizations and specializations in access controls can be modelled. This allows us to define two types of role hierarchies: *general* role hierarchies and *limited* role hierarchies. General role hierarchies provide support for an arbitrary partial order role. This type can be used to include the concept of multiple inheritance permissions and user membership between roles. Limited role hierarchies impose a restriction set, resulting in a simpler tree structure (i.e. a role may have one or more ascendants, but it is restricted to a single descendant). It should be noted that an inverted tree is also possible.

Another important aspect in the RBAC model is the possibility of specifying constraints on the relation between users/roles and on the activation of role sets for users. These constraints are a strong mechanism for establishing high level organizational policies. These restrictions can be of two types: static or dynamic. Static constraints allow us to solve interest conflicts and role cardinality rules from a policy perspective. User association with a role can be subject to the following constraints:

- A user is authorized for a role only if that role is not mutually exclusive with any user's authorized roles (Static Separation of Duty).
- The number of users authorized for a role cannot exceed the role's cardinality (Role Cardinality).

Dynamic constraints define rules to avoid a pair of roles being designated as mutually exclusive regarding role activation (Dynamic Separation of Duty). In other words, a user may be active in only one of the two distinct roles designed in this way.

We believe, however, that RBAC has a series of deficiencies for modelling access control for collaborative processes:

- In RBAC, the roles can be said to be static because of the lack of flexibility and responsiveness to the environment.
- RBAC supports the notion of user active roles with the concept of session, it obtains available permissions set from these active roles, but it does not take into account the sessions that are open by other users. Consequently, the model does not include the full context associated with the system. By way of example, in an educational environment, RBAC does not allow us to model how temporary permissions normally corresponding to the Headteacher may be granted to the Deputy Headteacher in the Headteacher's absence.
- RBAC lacks the ability to specify a fine-grained control on individual users in certain roles. One example might be in a hospital situation where a group of health workers are attending to a patient. Members of this group would therefore have access to the patient's medical record, although if one of the members of this group were to play the role of *hospital porter*, there is no reason for that member to have access to this record.
- In the previously described situation, we can observe the need to establish common permissions for the user group. RBAC can resolve this problem by adding a specific role and assigning it to every user in the group that needs it. The problem is that in collaborative systems there may be a very large number of user groups and the majority of these groups will require the allocation of temporary access permissions making access control more difficult to understand and control.

We therefore propose an extension of the RBAC model in order to integrate these elements into system functionality and structure. When we are describing the organizational structure of a system, we include a series of similar elements to those proposed by this model, so that starting with these we can derive the necessary information to manage access control.

3. Web service role-based access control architecture

In recent years, there has been a change in the majority of business processes and this is mainly due to market changes and to the integration of new technologies. These changes have provided new forms of client services and inter-operating businesses. As a result, information systems have been developed which use the infrastructure provided by Internet. On an architectural level, there are different interconnected subsystems which collaborate in many cases in order to carry out activities that were previously performed in a centralized way. Again on the same level, there has also been a sharp increase in the use of Web service architectures (WSA) to integrate business located in different companies or to describe collaborative processes carried out by different company users/subsystems. From an architectural point of view, it is important to separate the security aspects from other application characteristics and the Web service architectures are a good platform for performing this separation.

In the case of functions related with security and more precisely with task authorization and the control of access to activities and resources, we propose that a subsystem be included specifically for these functions. In many cases, this subsystem must interact with different systems. We think that it is better to use a Web service that can be used by the other systems without knowing its internal process.

It is, however, necessary to know the business process definition and the organizational work flow as well as the execution context of these business processes (users/active roles, tasks or activities in execution, etc.) so that access control can be improved to allow adaptation at any given moment to the current state of the organization. This can result in the implementation of two Web services: one for process management and another to know and manage the execution context of the business process.

Thanks to our previous work in software architectures [14,15] and more precisely in architectures for collaborative systems [7], in Fig. 2 we present a diagram of a partial system architecture which reflects three security-related

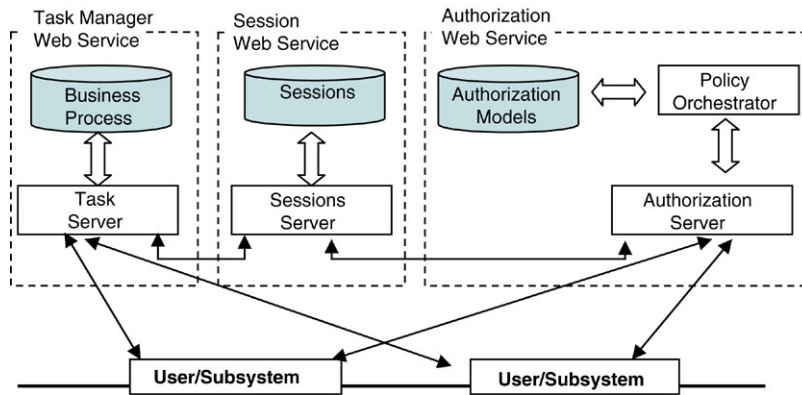


Fig. 2. Architecture diagram.

services: Authorization Web Service, Task Manager Web Service, and Session Web Service. In order to observe the operation of this architecture, we will describe how communication and coordination are necessary between these three services in an enterprise system of a bank administration. We will focus on the bank loan process.

The bank administration application is part of our system and when we designed it, a part of its functionality was moved to the services that provide the access control management in the system. When a client needs a loan, the application must communicate with the Task Manager Web Service to know what initial activity must be performed (in this example, this might be “Check data of client requesting the loan”). Previously, at the beginning of the process, a bank user was connected to the application and the application generated a request to the Authorization Web Service. This determined whether the user could initiate a new loan with his/her current role.

In order to carry out this action, Authorization Web Service will use the Session Web Service to discover the current context of the system since it is necessary to know which tasks have already been completed. The application will then continue with the remaining processes, which are received by the Task Manager Web Service, and activity and resource access control will be derived on the Web Services offered by the architecture.

The proposed architecture is based on a set of models, which describes the system. The access control service is driven by a system organization model, which contains all the information that the authorization service needs in order to carry out its operations. One of the advantages of this approach is that changes in the model can be carried out dynamically and the general functionality of the system can be modified at runtime.

There are two levels of changes:

- System evolutionary changes: changes in the model that are dynamically recalculated in the system. For example, we can change the permission assignment policies for a certain resource by adding a new role within the organization and distributing functions between the new role and pre-existing ones.
- Model adaptive changes: changes in the organizational structure that are predetermined in the model. For example, we can describe the policy of delegating existing authorizations in a bank for the granting of a loan: i.e. an assistant bank manager is authorized to play a bank manager’s role in his/her absence.

In the following sections, we will describe each of these services in more detail, including the main operations that they offer to the rest of the system.

3.1. Authorization Web Service

Authorization Web Service stores information about system authorization policies. Other subsystems and applications in the enterprise will use this service to know if they have access to resources according to activities currently executed and to the users and their roles. Due to the dynamic characteristics of enterprise system processes and their adaptation to the current context (lack of personal, organizational structural changes, etc.), a wide range of information (roles, tasks in execution, moment when the access is requested, etc.) is necessary to determine access to the tasks and resources by means of the Web Service. This service uses RBAC as a model and adds new characteristics to allow the elements which are necessary for dynamic enterprise systems. The service offers the following operations:

- Register users and roles — This operation stores information about the users belonging to the enterprise and the responsibilities that can be carried out by these users (roles).
- Link users and permissions to roles — This is used to model part of the organizational system structure and to associate users with their responsibilities (roles). We also use this operation to indicate resource and task access permissions for each role.
- Modify roles currently played by a user — This allows the structure of responsibilities in the organization to be modified, i.e. establish or remove user/role assignment.
- Verify access to resources according to the user’s active permissions — This is used to indicate whether a user who carries out responsibilities (roles) within the system has permission to access a specific resource.
- Check access to activities according to the user’s active permissions — This is used to indicate whether a user playing a specific role has permission to carry out a task or activity.

Two types of access are used to facilitate service management: “user mode access” used by applications to control access to shared resources, and “administrator mode access” with which modifications in the authorization models can be performed.

3.2. Task Manager Web Service

Since we think that it is important to control the logic of communication and coordination in business processes, it is therefore necessary for the Task Manager Web Service to coordinate the tasks (activities and aims that are defined by business processes) which are performed by the elements within the system (either individually or collaboratively) and to store a task model with information about tasks, activities and subactivities comprising them.

In order to manage a business is necessary to consider the following two groups of aspects [19]:

- *Declarative* aspects: these relate to ‘what’ needs to be done. These aspects include the input/output specification for each activity and the relationships between activities and actions for each task. In addition, these aspects can include the description of aims that should be reached in collaborative processes which cannot be described using a task sequence.
- *Operational* aspects: these relate to ‘how’ processes must be done. These aspects represent the detailed specification of every step in the sequence of subactivities. For collaborative tasks, we have a detailed specification of how the aim may be reached.

Since this service attempts to cover certain aspects related with the task flow in an application, part of the logic for coordinating the subsystems can be sent to the service, and with the incorporation of this service, application development and adaptation to the corporate business model is facilitated. According to this service, we can differentiate two types of operations: *registry* operations, which are used to store information about the task model, and *query* operations, which are used to obtain information about the model and therefore to coordinate the activities with this information.

Registry operations are:

- Register a task — This is used to incorporate a new functionality into the system.
- Link activities to tasks and roles — This is used to associate activities to a task and the set of permissions which are necessary for their execution.
- Register interruption events — This operation registers the tasks and/or roles that can be interrupted in other tasks. In our opinion, a task can be interrupted during its execution by a condition related with the task.
- Link resources to a task — This is used to associate resources needed to execute an activity, resources which must be available at the beginning of an activity.
- Register final states for activities — This is used to obtain final states for an activity which will determine the sequence of activities and therefore the associated work flow.
- Register initial states for activities — This is used to obtain possible initial states for an activity.
- Register an aim — This is used to obtain the specification of an aim as a series of individual sub-aims.
- Register the aim associated with a task — This is used to associate the necessary aims with the task for task completion.

- Register transaction rules — This is used to register the rules which determine navigation between activities based on temporary expressions. In a first approximation, we use the temporary operators defined in the CONCURTASKTREE [16] model.

Associated with each of these operations are another two operations which can be used to modify and delete the model information. One of the most important requirements of these operations will be to constantly leave the system workflow in a consistent state. Changes that are performed in the context of the system can quickly be applied to the model and indirectly to the task services so this allows the system to be rapidly adapted to the new changes.

Query operations are:

- Query the initial activity for a task — This operation is used to obtain the initial task activity.
- Query the following activity in the work flow for a task — Given a previous activity and its final state, we can use this operation to obtain the following activity to carry out according to the work flow. This service is one of the most important because it is used to control the sequence of activities that must be executed by different subsystems involved in a cooperative task
- Query interruption events — either implicitly or explicitly.
- Query partial aims for an aim — This is used to obtain the list of partial aims that once completed enable the global aim to be reached.
- Query the task associated with an aim — Given an aim, we can obtain the task which is necessary to reach it.

3.3. Session Web Service

In previous sections we have described the importance of knowing the current context of the enterprise system (roles, active users, tasks or activities in execution, etc.). This information is necessary for managing user and task access control. In our architecture, we include a Session Web Service in order to prevent this functionality. The Session Web Service maintains a representation of the dynamic use of the system (current context and record of the finishing process). Its main objectives are to register finished and active tasks performed by each user playing a specific role in the system. The services register a set of elements in relation to the context in which the activities are carried out. We can therefore control the state of each user/subsystem using the information stored during a session.

This service offers the following *registry* operations:

- Register the finish of an activity within a task — so as to register the final state of an activity. This information allows us to evolve the work flow and to make organizational changes in the roles.
- Register active activities in a task — This operation is used to register the activation of one activity in a task.
- Register roles linked to users — Registering these links allows us to monitor and control activities in the system
- Register connected users in the system showing their active roles in the current session — so as to register connected users and their linking roles to enable us to control the interaction between users and activities in the system.

And *query* operations:

- Query finishing activities in a task — This is used to obtain the point reached in the work flow sequence given a task.
- Query partial aims reached up to a link to a global aim — so as to obtain the partial aims reached and the link with the global aim given a task.
- Query active activities in a task — These operations can be used to obtain how many activities are active in a task.
- Query resources which are being accessed for an active activity — so as to obtain resources which are being used by an active activity to enable us to control and monitor the relation of other activities in the system.
- Query the active role or roles for a user performing an activity — so as to obtain user roles in an activity. This information is used to allow us to make decisions that will determine whether other activities are carried out by one role or another.
- Query final state of an activity in a task — in order to obtain the final state of an activity, it will be necessary to determine the work flow to be followed which will be determined in many situations by the result of each activity.
- Query active activities in a task, users executing the task and the group to which the users belong — This enables us to determine the access control for a user on a task, if a series of activities have been completed.

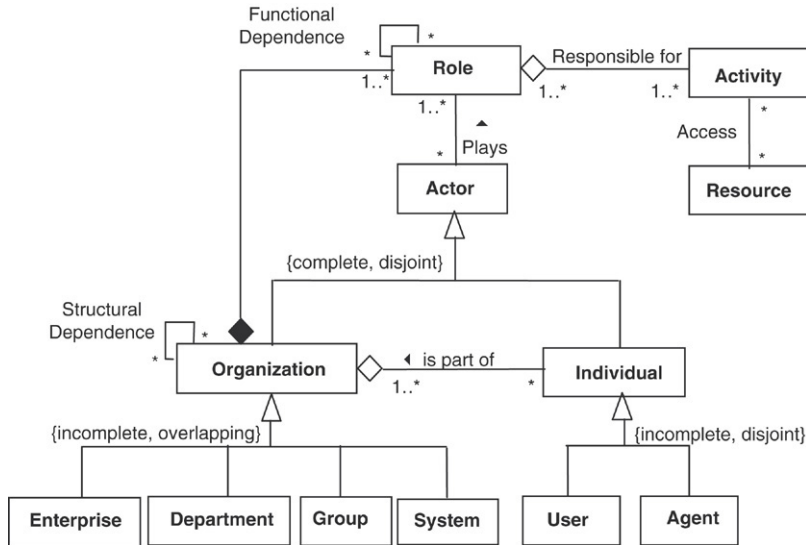


Fig. 3. Conceptual organization model.

- Query active roles for group users — obtaining this relation determines the functional dynamism in relation to making decisions about changes in roles in the organization.
- Query resources which are being used by a user, user role and user group — Obtaining this relation will determine both decision-making about role changes and also the availability of shared resources.

4. Organization model

In order to define the access authorization model, we use an organization model of the system actors (users, agents, subsystems, etc.) and consider static and dynamic aspects. While static issues are related with the organization structure, dynamic aspects should cover temporal changes in responsibilities, organization laws, capabilities, etc. Fig. 3 shows a conceptual model (using a UML class diagram) which allows the social organization of a system to be described. This model reflects the most important elements that appear in any organization, such as those which have traditionally been used in collaborative system modelling [21].

This conceptual model defines an *organization* (for example a company, a department, a group of users who temporarily take part in common tasks, etc.) as a set of *roles* and *functional dependencies* between them. In this way, we can model associations of a different nature, e.g. the possibility of a user passing from one role to another. From a structural point of view (*structural dependencies*), one organization can be included in other organizations (for example, a department can be part of a company).

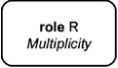
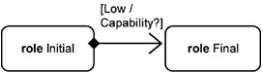

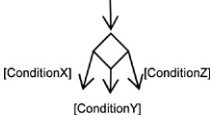
The *actor* concept includes both *individuals* (a user, a software agent, a robot, etc.) and *organizations*. An individual actor is at least part of one organization, and at any time, an actor (organization or individual) plays at least one *role* in an organization. Playing a role implies that the actor is responsible for performing activities associated with such a role. We can implicitly assume that an actor must have the necessary *capability* and *permission* to carry out the corresponding *activities* and to use its associate *resources*.

Unlike the RBAC model, functional dependencies in this model enable us to specify and control the role changes that an actor can undergo in a system. For this reason, we can say that this model allows dynamic role-based access control. In addition, our model allows the assignment of permissions to groups of users working on a common activity since an organization can play a role.

4.1. Notation

In order to define an organization model, we use *COMO-UML* [6] (a UML-based notation) and add several notational elements in order to capture concepts of a higher level, e.g. group, role, actor, organization, etc. This notation is integrated in *AMENITIES* [5,6], a methodology developed in our research group to analyse and design

Table 1
Notation for organization modelling

Symbol	Semantic
	Role — R is a role that a number of actors (limited by <i>Multiplicity</i>) can play at a given moment in an organization. It is a state belonging to a state machine which represents the dynamism of roles in an organization.
	Role Addition Transition — An actor who is playing an <i>Initial</i> role may also carry out the <i>Final</i> role. If this transition is labelled with a constraint (law or capability) it must be fulfilled.
	Role Change Transition — An actor who is playing an <i>Initial</i> role abandons it to adopt a <i>Final</i> role. If this transition is labelled with a constraint (law or capability) it must be fulfilled. In this case the actor has lost the responsibilities over the activities of <i>Initial</i> role.
	Decision Box. — Conditions labelling the outgoing transitions determine the different alternatives with respect to the roles to be played. When various alternatives become true, the system or the actor is responsible for choosing the alternative.

cooperative systems, based on user behaviour and task models. This methodology provides different system views (*organization, cognition, interaction* and *information*) to describe a system independently of its implementation, providing a better understanding of the problem domain.

The group structure and behaviour is modelled in the *organization view*. This view uses an extension of UML state machine diagrams to represent the organization according to the different roles that the actors can perform in the system and the role changes that can take place. Each state is a role in the organization and the transitions can be labelled with constraints.

AMENITIES introduce two kinds of constraints that allow us to model the dynamic aspects of the organizational structure: *laws* imposed by the organization and *capabilities* that an actor may acquire in the organization. In this way, participants could acquire new capabilities, apply new work strategies, etc. In all cases, it is necessary to satisfy the laws which govern the general behaviour of the organization. The activities related with each role are described in the cognitive view. In this case, we use activity diagrams to represent the workflow, participants, resources and interaction protocols of the cooperative scenario.

Table 1 briefly describes some of the COMO-UML notation elements for modelling an organization.

5. Organizational patterns

Patterns have become a valuable instrument for describing and reusing useful models in software engineering [1, 4]. In this respect, the modelling of the organizational structure and behaviour can benefit from the systematic use of specific conceptual patterns (*organizational patterns*).

Different studies [3,13] have proposed general organizational structures which often govern complex systems. For example, organization styles such as structure-in-5, joint venture, vertical integration, pyramid, etc. These structures are suitable for modelling the entire organization focusing on the distribution of its components (organizational units or individuals) in order to obtain common goals. Nevertheless, other (fine-grained) social structures, such as broker, mediator, embassy, etc., can often appear within organizations.

We have encapsulated common organizational structures as organizational patterns which can be reused in different modelling situations [8,10]. They provide a common vocabulary that improves the communication and discussion of the organization models. In addition, the final models are easier to understand and maintain. We have also defined a complete UML profile [9] to model software patterns in general. We therefore use this profile (together with COMO-UML notation) to model organizational patterns. In order to understand the case study in the following section, Table 2 details some of the notation elements of this profile that we shall use later:

Table 2
Some notation elements for patterns

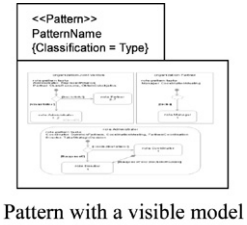
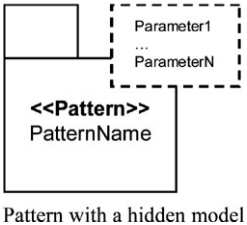
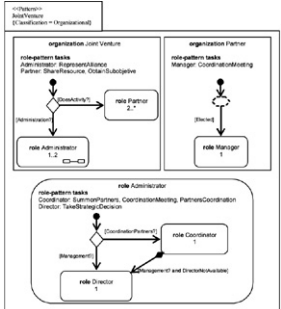
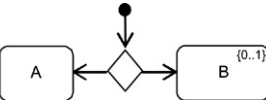

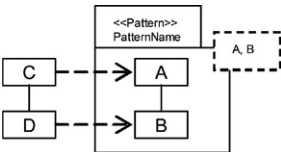
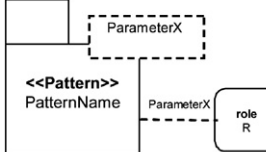
Symbol	Stereotype description
 <p data-bbox="205 485 452 508">Pattern with a visible model</p>	<p data-bbox="566 284 1358 409"><<Pattern>> — A parameterized package which defines a reusable generic model. The definition of this model allows us to use it as an expandable and adaptive template for the construction and/or description of similar concrete models (the instances of the pattern). The models that represent the pattern can either be presented within the package or not. The name of the pattern and its classification is indicated.</p>
 <p data-bbox="205 735 452 757">Pattern with a hidden model</p>	
 <p data-bbox="186 1089 467 1115">PatternElements contained in a pattern</p>	<p data-bbox="566 771 1358 982"><<PatternElement>> — Each element that defines the model of a pattern. These are abstractions that represent concrete model elements of its instances. When a PatternElement belongs to the group of parameters of the pattern, it can be bound to particular elements (arguments) of an instance. The binding will be valid when the conditions that the pattern imposes are completed, i.e. the type of each parameter coincides with the type of its arguments, the number of bound arguments matches the multiplicity of the corresponding parameter, and each argument and its parameter maintains the relationship with the other elements in the same way.</p>
 <p data-bbox="186 1236 452 1262">B can be optionally bound (multiplicity = {0..1})</p>	<p data-bbox="566 1129 1358 1182"><<MultiplicityBind>> — Constraint of a PatternElement that defines the number of model elements (arguments) that can be bound by means of a PatternBind dependence.</p>
 <p data-bbox="205 1381 452 1407">R is an accessible state (role)</p>	<p data-bbox="566 1276 1358 1357"><<UncertainElement>> — Part of a diagram can be ignored. These unknown model elements can be represented with a dotted hexagon. It allows us to specify the context of the elements that define the pattern.</p>
 <p data-bbox="186 1574 467 1600">Binding using UML dependencies</p>	<p data-bbox="566 1415 1358 1576"><<PatternBind>> — The tail element of the dependence (argument) is represented (within the pattern) by the PatternElement (parameter) of the head. When the symbol of the PatternElement is not visible, the dependence can point directly to the symbol of the pattern that contains it, but this dependence will have to be labelled with the corresponding parameter. An alternative representation consists in connecting the elements by means of a dotted line.</p>
 <p data-bbox="186 1793 452 1819">R is bound to ParameterX using a dotted line</p>	

Table 3
Pattern description template

Name	It should be significant, brief and to the point.
Alias	Another name by which this pattern is known
Classification	According to some previously established taxonomy
View	AMENITIES Cooperative Model view where the pattern can be used
Problem	What is the scenario that we need to describe?
Context	In what situations can it be applied? How can these situations be recognized? This shows the preconditions under which the problem and its solution can occur.
Participants	Description of the elements that take part and their responsibilities
Solution	A model which describes the participants, structure and behaviour, using the pattern profile and COMO-UML notation. It can include variants.
Explanation	Description of the proposed solution
Example	Application to a real case
Related Patterns	Other related patterns belonging to the same catalogue. For example, patterns that can be applied (before or later), alternatives, etc.

5.1. Pattern description template

We shall use a structured template divided into sections in order to show the different aspects and requirements for a pattern (Table 3). This also facilitates learnability, comparison and use.

5.2. Case study

According to the above template, we will describe the *Production Line* pattern. This defines a typical organizational structure that appears in many contexts (e.g. an assembly line, the sequence of treatments that a raw material requires to render it useful, etc.). We shall apply the *Production Line* pattern to guide the organizational modelling of a garage. The main goal in this kind of organization is car repair and normally, this process is divided into several sequential tasks assigned to different roles. This pattern guides role modelling (static and dynamic), facilitates the assignment of responsibilities, and consequently access control to activities and resources (see “Example” section).

Name: *Production Line Pattern*

Alias: Unknown

Classification: Organizational

View: Organization

Problem: To describe an organization where various actors collaborate in order to achieve a common goal in several sequential stages. Each step is viewed as a partial elaboration of a product or service (for example, an assembly line).

Context:

- The common goal is achieved in several sequential stages.
- In order to achieve the objective of one stage, it is necessary to have achieved the objectives of the previous stages.
- There is at least one actor who is responsible for coordinating the actors through the different stages.
- Actors are assigned to the stages according to their capabilities.
- The results of a stage are sent to the following stage.

Participants:

Link (role)

- Responsible for performing the tasks associated to some of the stages in the chain (*ExecuteStage* task).

Coordinator (role)

- Responsible for coordinating the actors who play the link roles (*CoordinateLinks* task).

Link::InitialLink (role)

- This is the role of an actor who participates in the first stage in the chain. They must send the resulting product to the next link (*SendResultToNext* task).

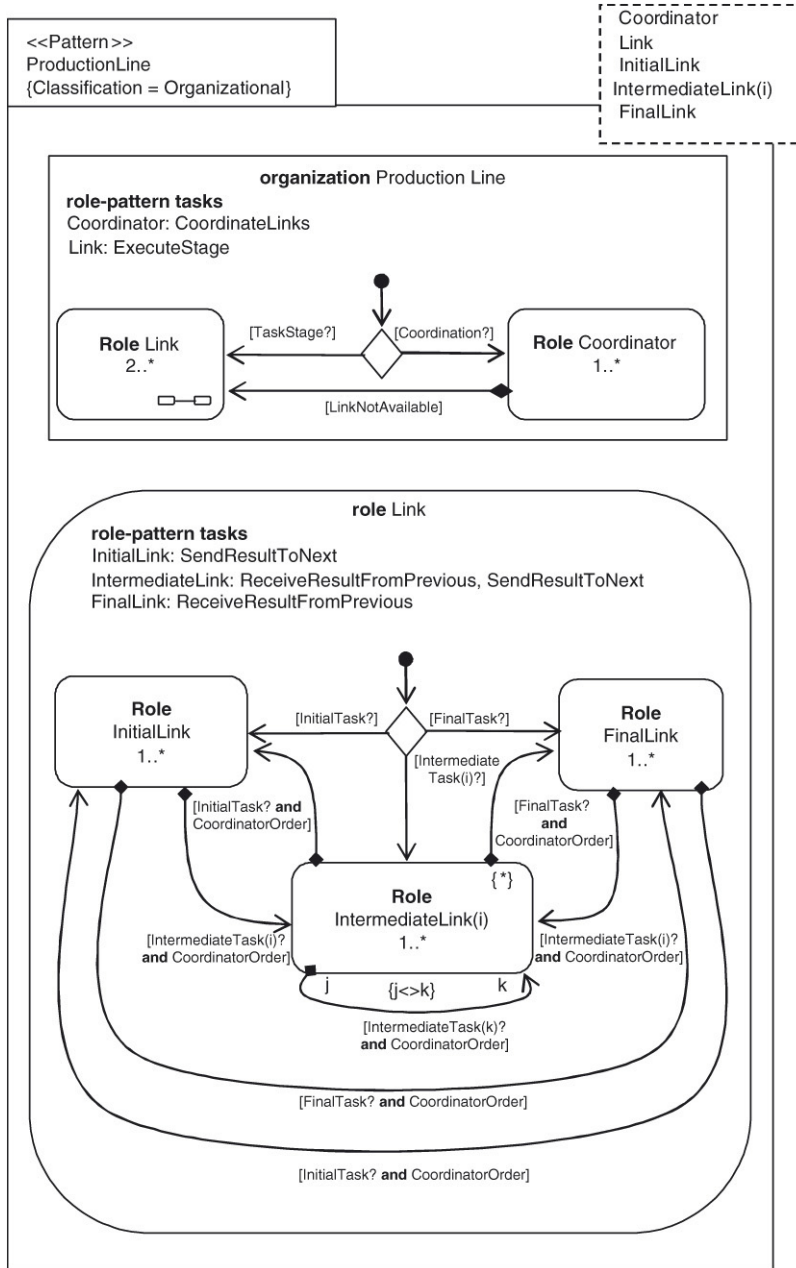


Fig. 4. Production line pattern.

Link::IntermediateLink (role)

- This is the role of an actor who participates in some of the intermediate stages in the chain. They are responsible for the *SendResultToNext* and *ReceiveResultFromPrevious* tasks.

Link::FinalLink (role)

- This is the role of an actor who participates in the last stage of the chain. They are responsible for the *ReceiveResultFromPrevious* task.

Solution: (See Fig. 4)

Explanation:

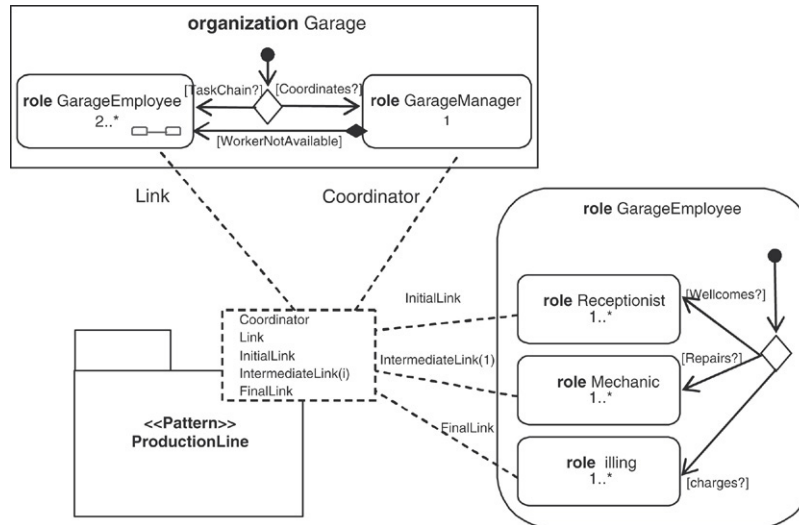


Fig. 5. Garage organization.

The actors in this kind of organization can basically play the *link* role or the *coordinator* role in the chain. It is necessary for at least two actors (2..*) to carry out the link role (an initial link and a final link). There should also be at least one actor (1..*) for link coordination (*Coordinator* role).

The *role-pattern tasks* section in the model indicates the responsibilities of each role. These tasks have been explained in “Participants” section.

Some transitions have conditions. These restrictions indicate organization laws or capabilities that an actor should complete in order to play the final role of the transition. The model illustrates a work chain which allows changes in the actors’ roles when certain events happen. For example, a *coordinator* can also play the *link* role when an actor of the work chain is not available (*LinkNotAvailable*), an actor performing an *intermediateLink* role can play the *initial* role if the actor is capable of doing the necessary task (*InitialTask?*) and has received an order from the coordinator (*CoordinatorOrder*), etc.

IntermediateLink role is constrained (the role symbol contains a MultiplicityBind = {*}). It indicates that zero or more concrete roles can be bound to *intermediateLink* role (multiple binding). In this case, the parameter identifier appends a variable, or index, in parentheses (*IntermediateLink(i)*). In this way, each parameter in a multiple binding will be distinguished by means of serial numbers (*IntermediateLink(1)*, *IntermediateLink(2)*, ..., *IntermediateLink(n)*). This is especially necessary when there is an order relation between the elements as there is in this case (the order of the stages is very important).

Example:

The organizational requirements of a garage can be modelled (Fig. 5) using this pattern as a template. In this case, the organization model is constructed by binding the pattern parameters (*Coordinator*, *Link*, *InitialLink*, *IntermediateLink* and *FinalLink* roles) to specific model elements (arguments) in the garage organization context (*GarageManager*, *GarageEmployee*, *Receptionist*, *Mechanic* and *Billing* role, respectively).

Each role is responsible for carrying out different activities:

- The *GarageManager* role is responsible for coordinating the garage employees (*CoordinateEmployees* task). We can observe that an actor in this role can also play a role in the work chain when an employee is not available (*WorkerNotAvailable* condition in the role addition transition).
- The *GarageEmployee* role is responsible for carrying out certain stages of the repair service (*ExecuteStageService* task).
- The *GarageEmployee::Receptionist* role makes a note of the car, client and problem (*GetInformation* task) and creates a repair report (*CreateRepairReport* task) which is then sent to the mechanic in charge of the repair (*SendRepairReport* task).

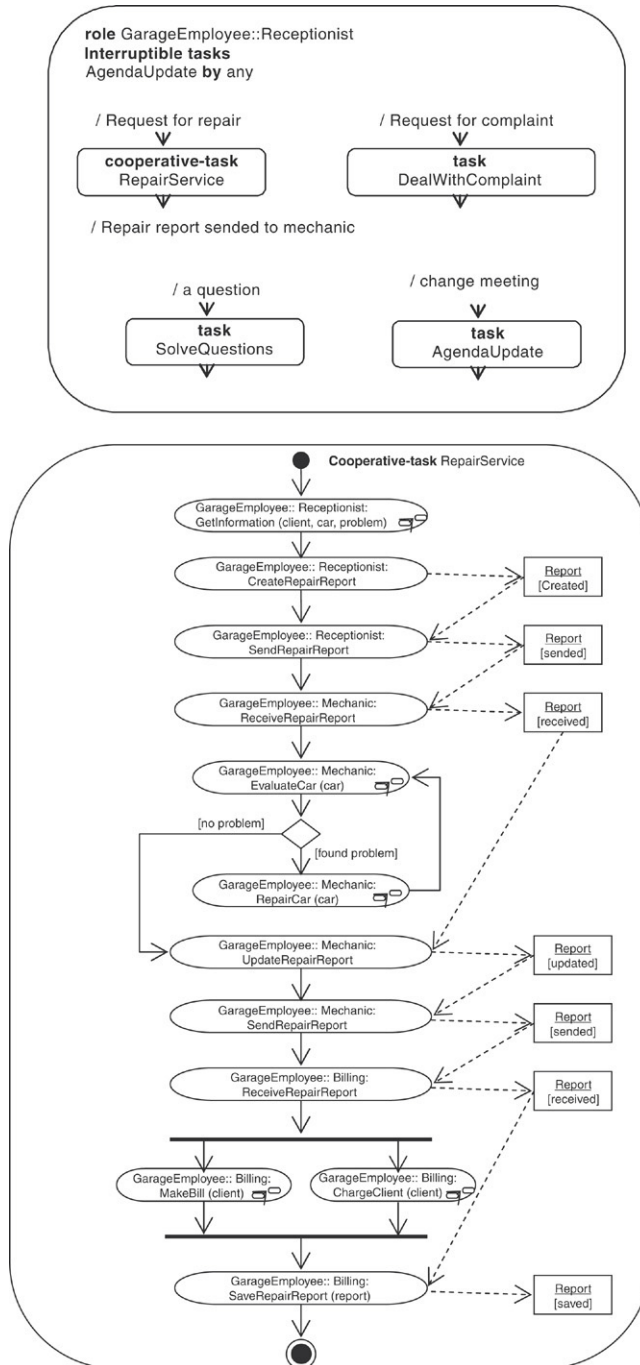


Fig. 6. RepairService cooperative task.

- On receiving the repair report from the receptionist (*ReceiveRepairReport* task), the *GarageEmployee::Mechanic* role evaluates the car (*EvaluateCar* task) and repairs it (*RepairCar* task) if there is a problem. Having finished, the report is updated (*UpdateRepairReport* task) and sent to the billing role (*SendRepairReport* task).
- The *GarageEmployee::Billing* role participates in the final stage in the chain and receives the repair report from the mechanic (*ReceiveRepairReport* task), makes out the bill (*MakeBill* task), charges the client (*ChargeClient* task) and keeps the repair report (*SaveRepairReport* task).

Related patterns:

Circular Production Line pattern

Fig. 6 shows a piece of the cognitive view representing the *RepairService* cooperative task by means of a COMO-UML activity diagram. It represents the previously mentioned sequence of activities, the responsible roles and the resources used in this cooperative task.

The Authorization server can use the activity diagram information to determine access control to activities and resources: e.g. the *Receptionist* role uses the *Report* resource once it has been created (*CreateRepairReport* activity) and subsequently when it is sent to the mechanic (*SendRepairReport* activity); the mechanic role updates the *Report* resource when the problem is resolved (or does not exist) and having received it (*UpdateRepairReport* task), etc.

6. Conclusions and future work

This paper outlines an automatic, portable and dynamic Web service for enforcing control access policies on collaborative systems. Having presented RBAC (an access control model based on role concept), we then presented a Web service role-based access control. This service uses part of a system model to control access to resources and activities by users and the subsystem. One of the main characteristics of our service is the possibility of making changes at different levels (evolutionary changes, adaptive changes, etc.).

We have presented a company organization model in order to define the access authorization model by considering static and dynamic aspect of systems.

Our future work will focus on the extended organization model in order to describe more aspects of system security.

Security management in current enterprise systems is a complex process and it must be analysed like other important aspects during development.

In this article, we have focused on the problem of the management of resource and activity access control within an enterprise system. In this kind of system, one of the greatest difficulties appears when managing collaborative processes since access control is much more difficult to model and implement because different users/applications take part in the processes and it is necessary to coordinate more complex concurrent activities.

Commercial software systems are currently available which automate and control part of the security needs in an enterprise system and the “Microsoft Active Directory” [12] is one example of such systems. This service stores information about the existing resources in the enterprise network infrastructure and about the access policies that are defined on these resources. One problem of such systems is that they consider security at a high level of granularity and this means that they are useless when security affects shared tasks and resources used by the different users and applications.

In this article, we have proposed a WEB service-based architecture that provides the necessary services for access control management. It is necessary to define complex models for the system’s organizational structure and the workflows represented by the enterprise process so that the proposed architecture works correctly. For this reason, we have extended the RBAC model with the elements that we think are necessary to define collaborative systems and dynamic access policies that can be adapted to the changes occurring in the context of the company.

In the final stage of this work, we have presented a pattern language used to define enterprise organizations at a conceptual level. We think that the use of patterns facilitates and accelerates the complex process related to the model construction of the organizational structures of an enterprise. These patterns allow us to define structures and situations that are repeated within different companies.

Our future work will focus on the extension of the organizational model in order to describe further aspects of system security.

We are also working on a general collaborative system architecture in which we will integrate the services defined in this article.

Acknowledgements

This research has been funded by the *Comisión Interministerial para la Ciencia y la Tecnología* (CICYT) (Spain), project AMENITIES, grant number TIN2004-08000- C03-02.

References

- [1] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, M. Stal, *Pattern-Oriented Software Architecture: A System of Patterns*, John Wiley and Sons Ltd, Chichester, UK, 1996.
- [2] C.A. Ellis, S.J. Gibbs, G.L. Rein, Groupware: Some issues and experiences, *Communications of the ACM* 34 (1) (1991) 38–58.
- [3] A. Fuxman, P. Giorgini, M. Kolp, J. Mylopoulos, Information systems as social structures, in: *Proceedings of the 2nd International Conference on Formal Ontology in Information Systems, FOIS'01*, Ogunquit, USA, October, 2001, pp. 10–21.
- [4] E. Gamma, R. Helm, R. Johnson, J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley Professional Computing Series, Reading, MA, 1995.
- [5] J.L. Garrido, M. Gea, Modelling dynamic group behaviours, in: C. Johnson (Ed.), *Interactive System-Design Specification and Verification*, in: LNCSS, vol. 2220, Springer, 2001.
- [6] J.L. Garrido, *AMENITIES: Una metodología para el desarrollo de sistemas cooperativos basada en modelos de comportamiento y tareas*, Ph.D. Thesis, University of Granada, 2003.
- [7] J.L. Garrido, P. Paderewski, M.L. Rodríguez, M. Hornos, M. Noguera, A software architecture intended to design high quality groupware applications, in: *Proceedings of the ICSE Research and Practice*, 2005, pp. 59–65.
- [8] J.L. Isla, F.L. Gutiérrez, M. Gea, Supporting social organization modelling in cooperative work using patterns, in: W. Shen, et al. (Eds.), *Computer Supported Cooperative Work in Design II*, in: LNCSS, vol. 3865, Springer, 2006, pp. 112–121.
- [9] J.L. Isla, F.L. Gutiérrez, P. Paderewski, Un profile para el modelado de patrones de software, in: A. Toval, J. Hernández (Eds.), *Actas de las X JISBD05*, Thomson Paraninfo, Granada, 2005, pp. 265–270.
- [10] J.L. Isla, F.L. Gutiérrez, J.L. Garrido, M.V. Hurtado, M.J. Hornos, Integration of organisational patterns into a group-centred methodology, in: R. Navarro, J. Lorés (Eds.), *HCI Related Papers of Interaction 2004*, Springer, 2006.
- [11] M. Goodyear, *Enterprise System Architectures: Building Client/Server and Web-Based Systems*, CRC Press, 1999.
- [12] Microsoft, introduction to active directory application mode, Microsoft Technical Report, 2003.
- [13] H. Mintzberg, *Structure in Fives: Designing Effective Organizations*, Prentice-Hall, Englewood Cliffs, N.J., 1992.
- [14] P. Paderewski, M.J. Rodríguez, J. Parets, An Architecture for Dynamic and Evolving Cooperative Software Agents, in: *Computer Standards & Interfaces*, vol. 25, Elsevier Science, 2003, pp. 261–269.
- [15] P. Paderewski, J.J. Torres, M.J. Rodríguez, N. Medina, F. Molina, A software system evolutionary and adaptive framework: Application to agent-based systems, *Journal of Systems Architecture* 50 (2004) 407–416.
- [16] F. Paternò, ConcurTaskTrees: An engineered notation for task models, in: D. Diaper, N. Stanton (Eds.), *The Handbook of Task Analysis for Human–Computer Interaction*, Lawrence Erlbaum Associates, Mahwah, 2003, pp. 483–503 (Chapter 24).
- [17] H. Shen, P. Dewan, Access control for collaborative environments, in: *ACM Conference on Computer-Supported Cooperative Work*, 1992.
- [18] R.S. Sandhu, E.J. Coyne, H.L. Feinstein, C.E. Youman, Role-based access control models, *IEEE Computer* 29 (2) (2006) 38–47.
- [19] K. Terai, N. Izumi, T. Yamaguchi, Coordinating Web services based on business models, in: *Proceedings of the 5th international Conference on Electronic Commerce*, vol. 50, ICEC'03, Pittsburgh, Pennsylvania, ACM Press, New York, NY, 2003, pp. 473–478.
- [20] W. Tolone, G. Ahn, T. Pai, S. Hong, Access control in collaborative systems, *ACM Computing Surveys* 37 (1) (2004) 29–41.
- [21] M. Van Welie, G.C. van der Veer, An ontology for task world models, in: *Design, Specification and Verification of Interactive System'98*, Springer Computer Science, 1998.