Role-Based Modelling of Interactions in Database Applications

Milivoje Petrovic, Michael Grossniklaus, and Moira C. Norrie
Institute for Information Systems,
ETH Zurich,
8092 Zurich, Switzerland
{petrovic, grossniklaus, norrie}@inf.ethz.ch

Abstract. Modern information systems interact with a wide variety of users ranging from people with specific roles in business processes to end-users who access information in various ways and in different environments. Therefore, an application has to adhere to a well defined security policy on one hand and be highly adaptable to context on the other. We have extended the OM data model, with its rich support for role modelling, with concepts for modelling interactions between users and applications. In this paper, we show how the resulting interaction model can be used for role-based access control, as well as for modelling interactions in context-aware applications.

1 Introduction

Interaction with users is one of the central aspects of designing an information system. Any interaction between a user and an application has to take care of two things. First, the application should be able to adapt to the users and their environment, and consequently provide only relevant information. Second, only authorised users should be allowed to interact with the application in a particular way. Depending on the type and purpose of a system, one or the other component is dominant. However, in every multi-user environment both context-awareness and security are always present to some extent.

An information system which supports an organisation’s business processes has to support interaction with users having different responsibilities. In such systems, it is important to deliver the right information to the right users within given time constraints, and to ensure that only users with certain roles are allowed to perform a task. In contrast, recent developments in mobile information systems focus more on delivering relevant information to users depending on contextual parameters such as location, preferences and client device. In content management systems, these two aspects come together as nowadays they are expected to offer multi-channel and personalised access, while at the same time enforcing strict access control to content authoring based on user roles.

Interaction depends on characteristics of users and application entities. Often the same characteristics are relevant to both security and context-awareness. Moreover, relevant information and services delivered to a user will always be
subsets of the information and services, respectively, that they are authorised to access. For these reasons, it is necessary to reason about interactions on an abstract level, describing them in a way that covers both aspects. Both security and context are orthogonal to the application’s functionality and therefore, instead of being modelled in the core application, they should be described in a separate model that enables their influence on runtime behaviour to be handled with well-defined algorithms based on that model.

In this paper, we present such a model and show how it can be used to capture both the context-dependent and access control aspects of information systems. A role-based approach to modelling interactions is used as role hierarchies map well to organisational hierarchies and their expressiveness makes them suitable both for capturing requirements and system design. However, role-based models are rather coarse-grained, and thus should be augmented with means to address fine-grained features.

Section 2 motivates our approach through an example scenario and discusses related work. The OM model and its support for role modelling is described in Sect. 3. Section 4 introduces the interaction model based on OM concepts. Section 5 discusses the role-based access control model obtained from the interaction model, while Sect. 6 discusses modelling interactions in context-aware applications. In Sect. 7 we briefly deal with implementation issues. Concluding remarks are given in Sect. 8.

2 Background

Throughout the paper, we will use a simplified version of an on-line conference management system to explain our approach. The different roles that users may have are Programme Committee (PC) members, some of whom are PC chairs, reviewers and authors. An author submits a paper before the submission deadline and classifies it by topics. PC members register their preference for topics and, given a list of papers associated with these topics, indicate their preferences for papers to review. The PC chairs then assign papers to reviewers who include PC members but may also include other nominated reviewers. Reviews should be submitted before a review deadline, after which, the PC decides whether or not a paper is accepted.

Each of the user roles has clear responsibilities and associated permissions. Reviewers can review papers but the system must ensure that only a reviewer who is assigned to the paper may actually activate review permission on it. Users may have multiple roles as a reviewer can also be an author. In this case, the protection mechanism has to take care of possible conflicts of interest and disallow that a paper is reviewed by its author. Further, permissions change over time according to the phases in reviewing process. For example, authors should not be allowed to submit or update papers after the submission phase is over and reviewers should not be able to change their reviews in the decision phase.

A conference management system not only has to be secure, but also should provide its users with information relevant to their role and task. For example,
when a PC member wants to select papers for review, the system should not only exclude the papers that they authored, but also present those which match their preferences by default. The system should also be proactive and push information to the user. For example, one day before the review deadline, the system could automatically send a reminder to reviewers who have not completed all of their reviews.

Probably the most widely used models for capturing interactions between users and applications are UML use case and interaction diagrams [1]. Use case diagrams are used to model the requirements of an application and to clearly define the boundaries of a software system in interaction with actors which can be either humans or other software systems. On the other hand, sequence and communication interaction diagrams model a sequence of messages passed between objects on the level of method calls. The level of abstraction that we need is between those of use case and interaction diagrams. Similar to use case diagrams, our goal is to capture interactions between users and applications. However, to be able to model access control and context requirements, application entities with which the user interacts have to be present in the model. We are not interested at this level in the order of the messages and low level interactions in the implementation of application logic. Rather, we want to focus on characteristics of a user and application entity which influence interaction, and to change the runtime behaviour accordingly.

Research in access control in the past was focused on mandatory (MAC) and discretionary access control (DAC). MAC associates security labels to both subjects and objects and manages access rights in a centralised fashion, whereas DAC leaves the administration of access rights to the object’s owner. Role-based access control (RBAC) [2,3] allows for simplified management of permissions since they are specified on the level of roles and not of individual users. RBAC is policy neutral and can also be used to support MAC and DAC policies [4]. Role hierarchies further simplify the management of permissions by allowing inheritance of access rights. However, as shown in [5], in some cases such inheritance may lead to violation of organisation’s control principles, such as separation of duty. In addition to generalisation based role hierarchies described in [2,3], some other types of role hierarchies have been identified in organisation structures. In [5], activity hierarchies were introduced to model inheritance of the responsibility for activities that are part of larger activities, whereas supervision hierarchies were used to represent user positions in the organisation hierarchy. In [6], contextual and seniority role hierarchies of users and asset hierarchies of accessed objects were introduced. Contextual roles are used to model the fact that, in order to access an asset, a user not only has to possess the required ability, but also needs to be linked to it through some situational role, such as a common project.

In [7], a model-driven security approach is taken where security and a design language are combined to enable automatic generation of access control infrastructures on target software platforms. SecureUML is a security modelling language for role-based access control built as an extension of UML and thus
with an intuitive graphical notation. It can be combined using a dialect with any design language in order to introduce access control on its resources. A security design model obtained by merging SecureUML and a structure model enables usage of OCL authorisation constraints to express activation of permissions depending on the state of a system, such as the existence of a relationship between a user and an accessed object.

The roles and properties of individual users and objects may also affect interaction between the user and the system in terms of the form and content of information delivered to the user in a particular context. In web engineering [8,9], a role-based approach for adaptation to user interests and access control is taken on the navigational and presentational level in a manner similar to adaptive hypermedia [10]. It is usually assumed that the user logs in with one role and navigates through the site view designed for this role. Personalisation is done by querying information associated with a particular user through that role. For our requirements, it is important to integrate access control and context awareness in the core interactions between users and application entities. Further, we need to take all user roles into account when filtering information. Finally, we want to deliver the relevant information by means of an algorithm at runtime based on the model, rather than with queries written in application logic. Note that in this paper, we will not deal with the gathering, representation and interpretation of context information, but rather assume that context is already present in the data model and use it to specify conditions which influence interactions.

There have been several attempts to bring together security and context. In [11], a dynamic access-based control model is presented with the focus on the activation and deactivation of user roles in the session based on changes in user context. In [12], a generalised role-based access control (GRBAC) model is proposed with the notion of environment roles. Environment roles represent conditions which have to hold for some permission to be granted. These approaches are concerned with using context information, such as location and time, to make access control decisions. Our primary goal, however, is to provide a model and an algorithm which can be used to achieve goals focused either on context awareness, access control or both. GRBAC is also interesting because it classifies not only users, but also objects into roles. Such a feature allows for greater flexibility in modelling.

We want to make several points in this paper. Firstly, security and context-awareness are related and consequently can be modelled using the same concepts. They are both system aspects [13] and should be represented in terms of a model rather than in application logic. Further, extensive use of role modelling for both user and application entities can satisfy complicated interaction requirements in a simple way. The modelling of fine-grained behaviour on the level of individual objects can be achieved by the integration of the interaction and structure model. Having users as application entities makes such integration simpler and leads to flexible user management. Finally, querying which takes multiple user and object roles into account can be expressed in such model and can be handled by means of an algorithm.
3 The OM Data Model

The OM data model combines conceptual modelling constructs present in extended entity relationship models with object-oriented constructs, by introducing a clear separation of typing and classification \([14, 15]\). While types specify the actual representation of objects, collections and associations represent their semantic groupings.

A simplified OM model of the conference management system is shown in Fig. 1. Shaded rectangles represent semantic collections of objects and ovals associations between the members of collections. Generalisation hierarchies are built from subcollection relationships between collections which are represented by directed edges between collections. For example, Committee, Reviewers and Authors are all subcollections of Users. The association authors captures the relationship between authors and their papers. Members is a subcollection of both Committee and Reviewers, expressing the fact that PC members are both reviewers and members of the committee together with its chair. The subcollections of Papers and Reviews correspond to the phases in the reviewing process. The partition constraint ensures that an object is always in exactly one phase. The system collections Objects and Users are parent collections for all application and user collections, respectively. The collection Users is also a subcollection of Objects, but this is not shown in the figure for reasons of clarity. Every application object has its owner, which is represented by the system association owns. Associations of interest between different user roles and Papers are represented. Since reviewers create reviews, the relationships between them are captured implicitly by the association owns.

The separation of typing and classification allows flexible role modelling to be supported as the multiple classification of objects simply corresponds to membership of multiple collections. At the type level, the OM model supports multiple
instantiation as objects are dynamically composed from type units depending on a particular type view. Every collection has an associated type which restricts membership to objects of that type and determines the default type view of an object accessed through that collection. Thus, if a user object is accessed through the collection Reviewers, the properties of type reviewer will be accessible, while if they are accessed through collection Authors then it will be the properties of type author. Further, associations can be defined between two roles without having to change the types of the associated objects. This allows for extensive use of role hierarchies and associations without impact on the representation.

Many object-oriented models lack the semantic expressiveness of OM in terms of role modelling. Another important factor is that OM is not simply a modelling language but also a model for data management and offers a well defined operational model inclusive of an algebra over collections and associations. Various implementations exist, such as OMSwe, a database system designed to support context-aware web publishing [16]. The OM model therefore offers better support for modelling interactions according to our requirements than UML and OCL.

4 Interaction Model

In the previous section we presented the structure model of the conference management system. Having identified user roles and the most important application entities, we can model interactions based on the requirements presented in Sect. 2. We first introduce the core concepts of the interaction model. An interaction represents a set of semantically correlated messages which can be exchanged between a user and an accessed object. An interaction constraint is a logical predicate which determines the validity of the interaction. To specify that an interaction is possible between a member of a user role and a member of the object role, we use an interaction relation which is a tuple consisting of a user role, an object role, an interaction and a set of interaction constraints. If interaction constraints are defined on the interaction relation, then all of them have to be satisfied for the interaction between two objects to be valid.

The model of interactions between user roles and papers is shown in Fig. 2. An interaction relation is graphically represented as a labelled, directed line between two roles. An interaction constraint is graphically represented as an annotation to the line. From the diagram, we see that a user who possesses the role of a reviewer may issue a review interaction for a paper if the interaction constraint assigned_to is satisfied for the two objects. Interaction constraints are expressed in terms of the OM collection algebra extended with the keywords caller and callee to refer to the user and the accessed object, respectively. Interaction constraints may accept additional parameters if they are defined for the interaction as in the case of the interaction assign_to(member). A list of the interaction constraints used for the model is given below. We assume there is a global object conference which holds information on submission_deadline, whereas parameter global_time represents the current system time. We see that
some interaction constraints are dependent on relationships between objects in the structure model, while others depend on context information such as time.

\[
\begin{align*}
\text{authored}_{\text{by}}(\text{param}) := & (\text{param}, \text{callee}) \text{ in a}"\text{authored}_{\text{by}}" \\
\text{authored}_{\text{by}} := & (\text{caller}, \text{callee}) \text{ in a}"\text{authored}_{\text{by}}" \\
\text{assign}_{\text{to}} := & \text{a}"\text{assigned}_{\text{to}}" \\
\text{before}_{\text{submission}}_{\text{deadline}} := & \text{global}_{\text{time}} < \text{conference}[\text{submission}_{\text{deadline}}]
\end{align*}
\]

The relationships in \text{assigned}_{\text{to}} are created in interaction \text{assign}_{\text{to}}. Both associations and interactions link two objects. However, associations represent static, persistent relationships between objects, while interactions are dynamic, transient relationships which exist only during the exchange of messages.

Both users and papers are classified into role hierarchies. When an interaction relation is defined between a user and an object role, it is inherited by all combinations of child user and object roles, including the user and object roles themselves. This means that the \text{view} interaction relation defined between \text{Chairs} and \text{Papers} implies that PC chairs can view papers in all phases. Similarly, the \text{review} interaction relation between \text{Reviewers} and \text{Assigned} implies that PC members can also review papers in that phase. In the case that a user or object role is a child of more than one role, then it inherits interaction relations from all of its parent roles. Nevertheless, in some cases, roles need to redefine the interaction relation defined between parent roles because of different interactions or interaction constraints, or both. An example is given in Fig. 3 where the interaction relation \text{view} between \text{Reviewers} and \text{Reviews} is redefined by the interaction relation between \text{Members} and \text{Finals}, to allow only PC members to view all other finalised reviews once they have finalised their own.
To model the creation and deletion of objects, we assume that, for each OM type, there is a collection corresponding to the extent of that type. We introduce system interactions create and delete which are only allowed to be specified on collections that correspond to type extents e.g. Reviews. Creation and deletion are considered equivalent to insertion into and removal from such collections. As reviewers create reviews, we define an interaction relation with the interaction create between the roles Reviewers and Reviews, as shown in Fig. 3. Interaction constraints can be specified in combination with the create interaction, in which case they are checked after the interaction has been carried out and the object created. In this example, the constraint of_assigned_paper given below checks whether the reviewer is assigned to the paper for which the review is created. In the case of instance deletions, all interaction constraints are checked before the interaction. When it comes to other collections, insertion and removal have different semantics as they correspond to granting and revoking roles. The system interactions grant and revoke are allowed to be used only when the accessed collection does not correspond to the extent of a type.

\[ \text{of\_assigned\_paper} := \]  
\[ (\text{caller, callee}) \text{ in inverse (a"assigned\_to") compose a"has\_reviews"} \]

The metamodel of the interaction model is presented in Fig. 4. An interaction relation connects a user role, an object role, an interaction and a set of interaction constraints. Both user and object roles can take part in many interaction relations. An interaction constraint expresses conditions from the structure model and thus can be used for many interaction relations. Since an interaction is just a set of messages, the same interaction can exist between many combinations of user and object roles. Thus, an interaction can be related with many interaction relations. An interaction relation can redefine zero or one interaction relations between parent roles, with the constraint that both the source and target roles are subcollections of the source and target parent roles. Further, an interaction relation can be redefined by many interaction relations defined between different combinations of user and object child roles. Interactions which correspond to a
single message are called *atomic interactions*. An interaction may be composed of many interactions which in turn can either be composite or atomic. Any interaction can be part of many interactions. Limitations in composing interactions come only from the evaluation of the interaction constraints in interaction relations. Any interaction constraint which is valid on the parent interaction has to be valid on the child interactions.

Both a user and an accessed object are classified into many roles in parallel inheritance paths. The resolution algorithm given below takes this into account to check the validity of interaction \( i \), issued by user \( u \) on object \( o \), by passing a set of parameters \( P \). A more general version of the algorithm receives a collection of objects and returns those for which the requested interaction is valid. This is useful when querying for objects which can provide a certain interaction.

1. let \( R_u \) and \( R_o \) be sets of user and object roles of \( u \) and \( o \), respectively
2. let \( IR \) be the set of interaction relations \( IR_k \) for which \( (\text{has\_caller}(IR_k), \text{has\_callee}(IR_k)) \in R_u \times R_o \) and \( i \in \text{has\_interaction}(IR_k) \)
3. let \( IR' \) be the subset of \( IR \) obtained by leaving out all \( IR_k \) for which exists some \( IR_j \) such that \( \text{redefines}(IR_j) = IR_k \)
4. accept if there is \( IR_k \in IR' \) for which for all \( c \in \text{has\_constraints}(IR_k) \), \( c(u,o,P) = \text{true} \)

5 Role-Based Access Control Model

The interaction model of the conference management system is in fact a role-based access control model. We specified which interactions are allowed for user roles in different phases of the reviewing process. In that sense, interactions are permissions that are activated if all interaction constraints evaluate to true for
objects in the interaction. Thus, interaction constraints play the role of authorisation constraints in this case.

We will show how the concepts of our interaction model relate to those proposed as a possible standard for role-based access control [3]. The first important difference stems from the OM feature that not only user objects, but also objects accessed by them may have multiple roles. This means that our model is role-based both from the point of view of the user and the accessed object. Permissions correspond to interactions, and they can be used in many interaction relations defined between different user and object roles. Thus there exists a chain of many-to-many relations between users, user roles, permissions, object roles and objects. Interactions can be decomposed to the level of atomic interactions which correspond to operations on objects. As atomic interactions can be part of many interactions, there exists a many-to-many relation between permissions and operations as well. Such generality results in great flexibility in modelling security requirements. Especially useful is the usage of object roles. We used this feature to change access rights according to the reviewing phases. Otherwise, the state information would have to be maintained in the object itself, leading to non-intuitive diagrams and complicated authorisation constraints composed of a large number of logical conjunctions and disjunctions.

The second important difference stems from the redefinition of interaction relations. In the proposed standard for role-based access control, a partial order relation exists between roles: senior roles acquire the permissions of their juniors and junior roles acquire users of senior roles. Senior roles correspond to the child roles of the OM model and junior roles correspond to its parent roles. The acquisition of users is achieved using OM subcollections, while the acquisition of permissions is obtained in the interaction model. If a child role has more than one parent role, it inherits the permissions from all of them. Also, a parent role may have many child roles which then inherit its permissions. Thus, role hierarchies are general as they allow multiple inheritance of both users and permissions. The same observations are valid for object role hierarchies. In our model, interaction relations can be redefined by child roles, thus changing permissions and authorisation constraints. This allows for situations where a senior role has less permissions than the junior role. This breaks the partial order but allows for greater flexibility. Without this feature, non-conceptual roles would have to be introduced, as is the case with private roles in [2], thus leading to non-intuitive models.

Authorisation constraints partition objects with the same role into those which are authorised to participate in a given interaction and those which are not. This allows for the expression of fine-grained security requirements. A particularly important authorisation constraint is owner since, in many security policies, the object’s owner has special rights on it. The owner constraint stems from the system association owns defined between system collections Users and Objects. It can be placed on any interaction relation since the user and the accessed object must be members of the corresponding system collections.
The constraint and operational model of OM allows for the specification of complex constraints such as separation of duty, where a task is divided into smaller ones which have to be done by different persons. Simple cases can be handled on the level of the structural model using OM constraints. For instance, the disjoint constraint could be used to ensure that a user object cannot possess more than one role from a set of child roles. However, more complex cases require the use of authorisation constraints. Since, in our example, a user may have both the author and the reviewer role, a conflict of interest is prevented by controlling permission activation using authorisation constraints. In similar fashion, authorisation constraints could be used to solve conflicts between control principles and inheritance of access rights posed in [5]. Relationships could be used to track performers of activities. Separation of duty would then be enforced by placing an authorisation constraint which would be negation of the existence of relationship between user and object in the previous activity.

Requirements stemming from the presence of different types of role hierarchies in organisations [6] can be expressed in terms of our model by using associations and authorisation constraints. User and object role hierarchies of our model correspond to functional roles and asset hierarchies, respectively. We do not need to explicitly define seniority and contextual roles that a user needs to possess in addition to functional roles in order to access an asset. Rather, we can introduce an association between user roles to model seniority, and associations between user and object roles instead of contextual roles. Then, we can use authorisation constraints based on these associations to achieve the same effect. Our model is general in the sense that it provides a framework where arbitrary associations between roles can be defined and then used for expression of authorisation constraints in terms of the OM algebra. This enables us to also support delegation and responsibility introduced in [5], since they are in fact based on relationships between user roles and sets of activities.

User objects are not treated differently from any other objects in the OM model. This is convenient for user management and administration as permissions on users can be modelled and controlled in the same way and using the same rules as other objects. The model shown in Fig. 5 states that both PC chairs and members can create reviewers, and that only chairs can promote them to PC members. In this way, we can model and control the creation of users and propagation of privileges in a fine-grained way.
Our model is similar to a security design language obtained by a merging of SecureUML [7] and a static structure design language in that it is graphically intuitive and makes use of authorisation constraints. The difference is that in such a language, users and roles from SecureUML are not part of the structure model, and thus have to be connected with the application entities which represent them using a convention. This complicates the expression of authorisation constraints and, more importantly, prevents the administration of users in the same way as application entities.

6 Context-Aware Interactions

To be able to express context in terms of a model, we generalise the notion of context to any information present in the data and interaction model that is relevant for the interaction between a user and an application. This is in accordance with the definition of context given in [17]. Relevant information and its influence on interaction is expressed in terms of interaction constraints, which, in this case play, the role of context constraints. Any property that characterises a user, an accessed object, associated objects or the global environment can be context for a particular interaction. Often the fact that some relationship exists influences the interaction, thus being context itself.

To illustrate how we can support the delivery of relevant information and services based on the interaction model, consider the interaction in which a PC member indicates interests in papers. Although PC members have rights to access any paper, the system should provide them with the list of papers that matches their topic preferences as a default. To achieve this, we can classify PC members into many roles based on their preferences and expertise. Similarly, authors classify papers by topic. Then, we can introduce an interaction which indicates user preferences and specify it using interaction relations between the corresponding user and object roles, as shown in Fig. 6. When a user issues interaction list\_preferred on the collection Papers, the resolution algorithm takes all user roles into account, finds the corresponding object roles through the interaction relations and returns only papers having one of these roles. For further fine-grained modelling of preferences, it is possible to create multilevel user and object role hierarchies, and introduce context constraints on interaction relations. We believe that a role based approach to the modelling of user profiles and object categories is graphically intuitive and allows for multiple user and object roles to be taken into account when querying for preferred objects.

Further, when a PC member accesses a paper, the relevant functionality should be offered to them. If they access a paper in which they have already indicated an interest, the system should offer an update operation. Otherwise, it should offer them the option to indicate their interest. This behaviour is achieved by introducing the interactions indicate\_interest and update\_interest in combination with the context constraint indicated based on the association interested\_in.
We have shown how the runtime resolution algorithm can use the interaction model to adapt the system when a user makes a request. Nevertheless, context-aware applications need to be proactive. They should be able to push information to users and execute actions on their behalf when certain conditions are satisfied. Such conditions can be expressed using context constraints. This requires that the application is equipped with an environment that periodically checks all context constraints to trigger proactive interactions. Examples of such proactive interactions in our example are shown in Fig. 7. Interactions which push information are directed from the object to the user role, while interactions which represent automatic execution do not have a direction. In this case, the system will send a reminder to a reviewer one day before the review deadline if a review has not been finalised. The interaction `auto_finalise` states that the system will automatically move the review to the finalised state after the deadline, if a reviewer has set the `automatic_finalise` setting in their preferences.

\[
\text{automatic_set} := \text{caller[automatic_finalise]}
\]
\[
\text{reminder_time} := \text{conference[review_deadline]} - \text{global_time} < 24\text{h}
\]
7 Implementation

We have shown in Sect. 4 how interactions between roles are modelled on the conceptual level. However, the behaviour of individual objects is specified by the methods of the underlying object-oriented type model. Each interaction can be recursively decomposed to the level of atomic interactions and then mapped to method declarations in an object-oriented language. The interaction model does not deal with the method implementation, and redefinition of interaction relations is fundamentally different from method redefinition. Further, to ensure that interaction constraints are valid, object types need to have the corresponding properties. For example, given the model of Fig. 7, type `review` would need to provide an attribute for `review_deadline` and a common method for interactions `finalise` and `auto_finalise`. Finally, system interactions `create`, `delete`, `grant` and `revoke` correspond to the methods `insert` and `remove` of the collection type.

By developing the interaction model as an extension of OM model, we could integrate it into the core of eOMS, a new data management platform under development based on the OM model. The operational model of eOMS is based on a specially developed object-oriented database programming language OML, which brings together the algebraic operators of the OM query language with procedural constructs. Application logic is provided in OML methods and access control is enforced by the runtime checking of the validity of method calls by applying the resolution algorithm on the application database. To support context-awareness, we are integrating concepts developed in OMSwe, an OM-based platform for web publishing [16].

The interaction model could be also used together with other object-oriented platforms and languages. Similarly to the approach in [7], security infrastructures could be generated for existing software platforms which already support role-based access control to some extent, as is the case with application servers. However, because of the generality of the interaction model, complex mappings would need to be defined in order to bridge the gap between the models. Another approach would be to integrate the interaction model with an existing object-oriented language such as Java. Aspect-oriented frameworks could be used to instrument application code in order to perform security checks on method invocation. However, this would result in issues of how to handle the impedance mismatch arising from the fact that OM supports multiple instantiation and inheritance. Other issues would be how to keep the information on object roles and deal with OM operations in the case of unidirectional Java references.

8 Conclusions

We have shown that it is possible to have a unified role-based interaction model that caters for both the security and context-aware aspects of user interaction with database applications. This was achieved by extending the OM object data model with concepts of interaction relations, interactions and interaction constraints and providing mechanisms for the inheritance and refinement of interactions in role hierarchies.
We plan to facilitate modelling of complex application domains, by providing design patterns to address often occurring requirements. Also, we are working on combining of the interaction model with our extended version model in order to model interactions in collaborative and mobile environments.

References