Applying the UML to the ODP Enterprise Viewpoint

M.W.A. Steen and J. Derrick
Computing Laboratory, University of Kent at Canterbury, UK
{M.W.A.Steen,J.Derrick}@ukc.ac.uk

July 8, 1999

Abstract

Now that the Reference Model for Open Distributed Processing (RM-ODP) has stabilised, attention is shifting towards the definition of specific notations for the ODP viewpoints. The objective of this paper is twofold. Firstly, we analyse the current definition of the ODP enterprise viewpoint language. Using the UML, a meta-model of the core concepts and their relationships is constructed. Secondly, we investigate to what extent the UML can be used for enterprise viewpoint specification by means of a small case study. We conclude by discussing the main open issues with regard to enterprise viewpoint specification.

Keywords: RM-ODP, open distributed processing, distributed systems, enterprise viewpoint language, enterprise modelling, communities, roles, enterprise behaviour, UML.

1 Introduction

More than ever organisations depend on complex distributed systems to support their business. The purpose of these systems is to improve the quality, reduce the cost, or increase the timeliness of the business processes they support. In order to meet these goals, it is important that the way in which the business processes are organised are taken into consideration when designing distributed systems. The *enterprise viewpoint* of the Reference Model for Open Distributed Processing (RM-ODP) addresses this issue by providing a language for modelling distributed object systems from the perspective of the enterprise. More specifically, the enterprise viewpoint deals with the purpose, scope and policies for a system and its environment.

The RM-ODP defines a holistic framework for the specification of distributed systems. In order to deal with all aspects and complexities of such systems, the reference model defines five different abstractions (referred to as *viewpoints*) from which distributed systems may be modelled. For each viewpoint, it provides a *viewpoint language* that defines concepts and structuring rules for specifying ODP systems from the corresponding viewpoint.

Of the five ODP viewpoints, the enterprise viewpoint is currently the least well defined. Initial efforts of the standardisation community concentrated on the computational and engineering aspects of open distributed processing. Nevertheless, there is growing awareness that enterprise specification has an important role in the development of open distributed systems and in their integration into the enterprise which they serve. In recognition of this trend, the enterprise viewpoint language is currently undergoing extension and refinement within the ISO in order for the enterprise viewpoint to meet these requirements. In this paper, we analyse the current definition of the ODP enterprise viewpoint language. To this end, we construct a meta-model of the core concepts and their relationships using the Unified Modelling Language (UML) [2, 6].

It is important to note, at this point, that the RM-ODP is not prescriptive about the use of any particular notation for the viewpoints. The viewpoint languages, defined in the reference model, are abstract languages in the sense that they define what concepts should be supported, not how these concepts should be represented. Currently, the UML is emerging as a *de facto* standard for

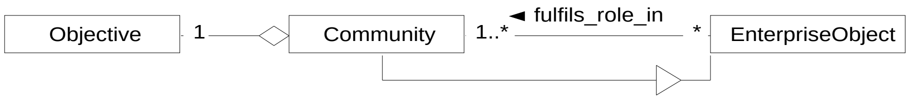

Figure 1: The community concept

object-oriented modelling. A second objective of this paper, therefore, is to investigate to what extent the UML can be used as a concrete notation for enterprise viewpoint specification.

The rest of this paper is organised as follows. In section 2, we analyse the current definition of the ODP enterprise viewpoint language. The result of this analysis is a meta-model of the enterprise language in the UML, which can be seen as the abstract syntax for a concrete enterprise specification language. In section 3, we then discuss how to represent a substantial subset of the enterprise concepts in the UML. We wind up, in section 4, by summarising our findings.

2 The Enterprise Viewpoint Language

In the RM-ODP, the *enterprise viewpoint* is defined as "a viewpoint on an ODP system and its environment that focuses on the purpose, scope and policies for that system" [4]. The *enterprise viewpoint language* provides the concepts and structuring rules for the specification of an ODP system from the enterprise viewpoint. The ODP enterprise language is an abstract language in the sense that it does not prescribe the use of any particular notation. The ISO is currently in the process of refining and extending the enterprise viewpoint language. In this section, we analyse the current definition of the enterprise language by constructing a meta-model for the language. The meta-model is represented in the UML. The enterprise language makes use of various concepts defined in part 2 (Foundations) of the RM-ODP [3]. In so far necessary, we will include these concepts in the meta-model.

2.1 Communities

Central to the enterprise viewpoint is the concept of a *community*. From a business modelling perspective, on the one hand, a community may be viewed as the participants (both people and systems) in a business process. From a system modelling perspective, on the other hand, we can view a community as a system and its environment. The former perspective may be relevant for analysis of the enterprise before a system has been developed and deployed, or for purposes of business process re-engineering (BPR). The latter perspective is useful for identifying the scope of a system.

In the reference model, a community is defined as "a configuration of [enterprise] objects that is formed to meet an objective." The objects in an enterprise model are (abstractions of) the people, systems and other entities that have a role in achieving the community's objective. An enterprise object may itself be refined as a community at a more concrete level of detail. Dually, a community may be expressed as a composite object appearing at a more abstract level of detail.

These structuring rules for communities are captured in the UML diagram in figure 1. Note that the enterprise language allows communities to have no associated enterprise objects, which does not make much sense. Perhaps a minimum cardinality of one or two should be imposed. Also note that the enterprise objects are not related to communities by containment. Instead an association (fulfils_role_in) is used to model the fact that enterprise objects may exist outside a particular community. However, enterprise objects taking part in a community, i.e., fulfilling roles in that community, are bound by an implicit or explicit agreement to work together towards the objective of that community. This may mean that these objects have to constrain their potential behaviour to satisfy the ruling policy of the community.

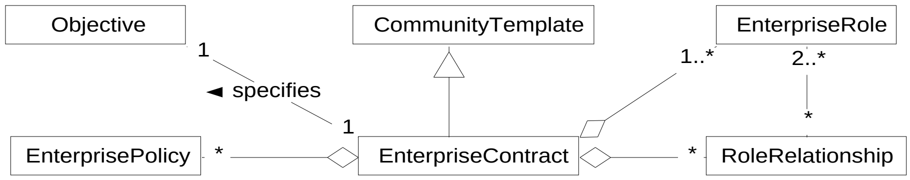

Figure 2: The community contract

2.2 Specification of communities

Communities are specified by means of a contract or community template, which identifies the different roles that objects may play in the community and a policy that governs the behaviour of these objects while fulfilling roles in the community.

There is a subtle distinction between an enterprise *model* and an enterprise *specification*. A model is a description of an existing situation — a real enterprise — and will describe the specific objects making up the community and their actual behaviour. A specification is a more abstract description of a desirable or future situation. It specifies how an enterprise should be organised and how it should ideally behave. If a specific model is consistent with a specification, then we say that the model satisfies or conforms to the specification. For a particular specification, there may be many ways of implementing it, resulting in different models.

One possible way of interpreting the structuring rules for community specifications in the draft enterprise viewpoint language document [5], is depicted in figure 2. The community contract is a template specifying the structure and potential behaviour of a community by defining the community's roles and their associations. A *role* is an "identifier for a behaviour" [3, Clause 9.14]. *Role relationships* define the associations that may exists between roles and determine the ways in which roles can interact. This concept is currently not defined in the enterprise language! In addition to the structure, a contract specifies the objective and the policies of the community. In the remainder of this section, the identified enterprise concepts are further refined.

2.2.1 Enterprise roles

There are four sub-classifications of roles that are of interest to enterprise specifications: *actor* roles and *artefact* roles, and *principal* roles and *non-principal* roles. It is not made explicit that these classifications are pair-wise mutually exclusive and total. Thus, all roles can be classified as *non-principal actor*, *non-principal artefact*, *principal actor*, or *principal artefact*. This classification is depicted in figure 3.

An *actor role* is defined as "a role with significant behaviour in that it initiates at least one interaction." An *artefact role* is defined as "a role for objects that are referenced by some community behaviour, and initiate no interaction in this community." An enterprise object fulfilling an

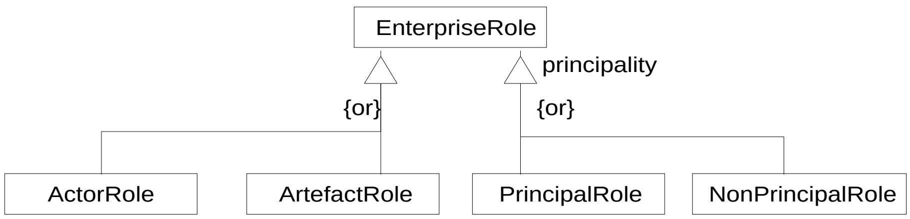

Figure 3: Enterprise roles

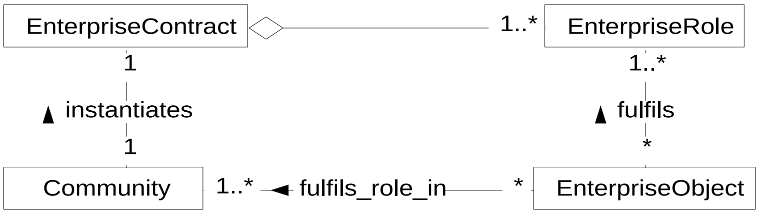

Figure 4: Instantiation of communities

actor role participates in the actions of that role, whereas an enterprise object fulfilling an artefact role only appears in an enterprise specification because it is referred to in some action of an actor. In a library community, for example, a borrower is an actor, whereas a book is an artefact (also see section 3.2).

By agreeing to fulfil a principal role, an enterprise object undertakes the obligations of that principal role and agrees to internally constrain its behaviour accordingly. Enterprise objects fulfilling non-principal roles are not agreeing to undertake obligations.

2.2.2 Relationships between roles

The concept of role relationship is not explicitly defined in the current version of the enterprise viewpoint language [5]. The following therefore represents our own interpretation of this concept. There are at least two different ways in which roles can be related. Firstly, roles can be related because they interact. In a library community, for example, the role of borrower and the role of loans desk assistant are related because they share the actions of borrowing and returning items. These relationships can usually be left implicit. However, there are other relationships between roles that should be made explicit for modelling purposes. For example, when a borrower borrows a book, a conceptual relationship (a loan) is established between that borrower and that book (also see section 3.2).

2.2.3 Instantiation

Communities are created by instantiating the corresponding contract. Instantiation of a contract involves the assignment of enterprise objects to roles. In general, an object may fulfil many roles, in any number of communities. For example, university staff may fulfil a teacher role in an educational community and also fulfil a borrower role with respect to the university library. Alternatively, a single role could be fulfilled by more than one enterprise object — a library has many borrowers, a company may have more than one network administrator, etc.

The relationships between the specification concepts of *community template* and *role*, on the one hand, and their instantiations, i.e., *communities* and *enterprise objects*, on the other hand, is depicted in figure 4.

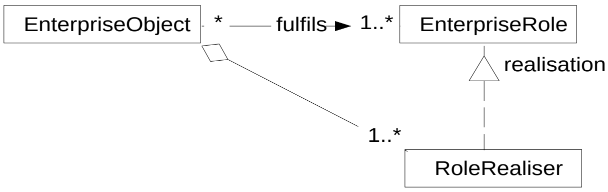

Figure 5: Role realisation

Note that the fulfils_role_in association between enterprise objects and communities is refined by this diagram to the *enterprise object* fulfilling a *role* which is contained in the *community template* of the *contract* that is instantiated by the *community.*

In figure 5, the fulfils association between enterprise objects and roles is further refined to the *enterprise object* containing a bit of behaviour, for each *role* that it fulfils, that realises or implements the *role.* We call this bit of behaviour the *role realiser.*

2.3 Enterprise behaviour

Roles are used to identify the behaviour of enterprise objects in the community. To this end, actions are associated with roles. In fact, all [enterprise] actions are identified by at least one actor role (the initiator). Since behaviour is defined to be "a collection of actions with a set of constraints on when they may occur," a role definition should also specify these constraints. Thus, a role specifies the behaviour that an object fulfilling that role may exhibit. Note that the concept of *behavioural constraint* is not defined in the RM-ODP! The reason for this is that the type of constraints that may be specified depends very much on the specification language that is being used. In a specification language like LOTOS [1], for example, constraints on behaviour are specified in terms of a temporal ordering of actions, which can then be composed with a synchronisation operator. In a model-based specification language, such as Z [7], on the other hand, behaviour is constrained, amongst others, through invariants on a state space.

Figure 6 depicts our interpretation of the clauses pertaining to role behaviour. An *enterprise role* defines some *enterprise behaviour*, which consists of a set of *enterprise actions* and a set of *constraints* on these actions, with each *constraint* relating to at least one *enterprise action.* At least one *enterprise role* participates in each *enterprise action.* In addition, each *enterprise action* is initiated by precisely one *actor role.* Actions that involve more than one role are called *interactions.*

2.4 Enterprise policies

Policies constrain the behaviour of enterprise objects that fulfil actor roles in communities. The policy concept is closely related to that of enterprise behaviour: both represent constraints on the behaviour of objects. They are not the same though. A specification of enterprise behaviour defines the physically possible behaviour of roles in a community. A policy specification may then constrain this behaviour further in order to achieve the objectives of the community. Thus, in general an enterprise specification will include a description of enterprise behaviour together with a policy specification. Another way of viewing the difference is to think of policies as describing the ideal and desirable behaviour within an enterprise, whereas the enterprise behaviour is a model of the actual behaviour. The latter may or may not conform to the ideal expressed in the policies. Policy specifications, therefore, often contain prescriptions of what to do in case some rule is violated.

Policies are designed to meet the objective of the community. Policies are specified by the community contract. In addition, part 2 of the reference model [3, Clause 11.2.7] states that a

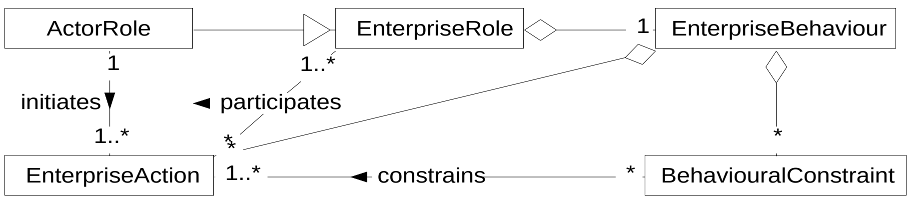

Figure 6: Role behaviour

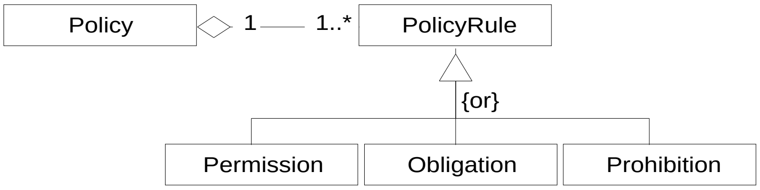

Figure 7: Policy structure

policy is "a set of rules related to a particular purpose" and that "a rule can be expressed as an obligation, a permission or a prohibition." The structure of policies is depicted in figure 7.

There are basic definitions of the concepts of *obligation*, *permission*, and *prohibition* in part 2. They have their usual meaning, but pertain to behaviour only. For example, an obligation is a prescription that a particular behaviour is required. These definitions are clearly too restrictive for enterprise specification and have to be refined to include the concept of agency. In the enterprise viewpoint we are often more interested in questions such as *who* has the obligation to perform a certain behaviour, rather than the question *which* behaviour is required. In addition, we would like policies to be first-class concepts, so that we can make statements such as "X has the obligation to grant Y permission to do A." Furthermore, enterprise policies not always pertain to behaviour. A more in depth discussion of such behavioural enterprise policies and a language for policy specification can be found in a companion paper [8].

Instantiation or population rules form another important class of policies. An example of such a policy is that the role of treasurer and the role of auditor cannot be fulfilled by the same person. In section 3.5, we present an example of such an instantiation constraint.

2.5 Summary

As a summary of our analysis of the ODP enterprise viewpoint language, figure 8 depicts the complete meta-model. In the next section, we use this as the abstract syntax for developing a concrete notation for enterprise specification based on the UML.

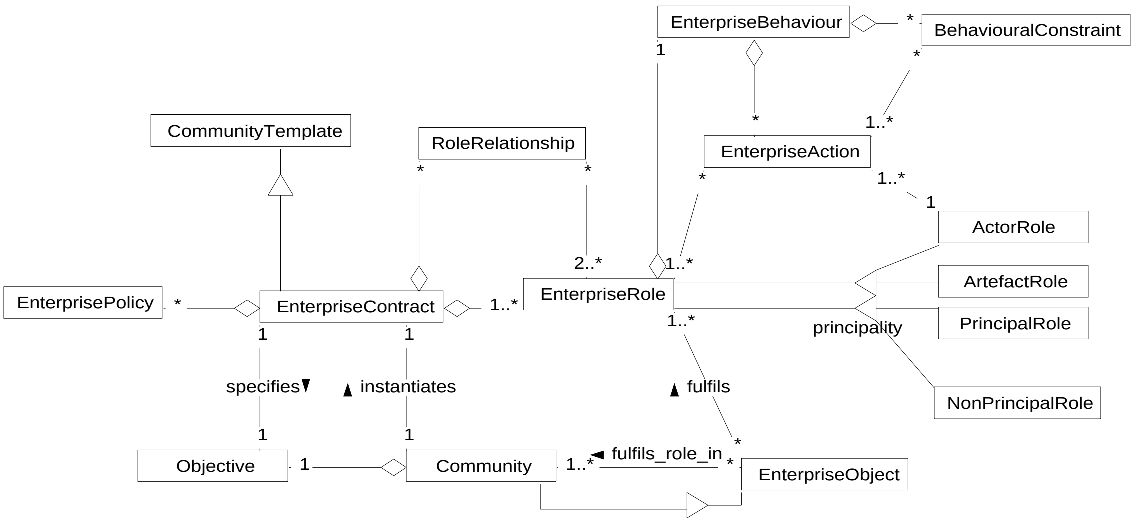

Figure 8: A meta-model for the ODP enterprise viewpoint language

3 The UML as Enterprise Viewpoint Language

The ODP enterprise viewpoint language is an abstract language in the sense that it does not prescribe the use of any particular notation. It merely identifies a set of concepts and a set of structuring rules. These have been captured in the meta-model presented above, which effectively defines an abstract syntax for the language. In this section, we investigate how an existing and popular notation can be used to provide a concrete syntax for (parts of) the enterprise viewpoint language.

The Unified Modelling Language (UML) is rapidly becoming the industry standard for object-oriented modelling. It brings together a number of widely used object oriented diagramming techniques, such as class diagrams, use case diagrams, state charts and sequence diagrams. The purpose of this section is to investigate how these diagramming techniques can be used to represent ODP enterprise viewpoint specifications. Using the UML for enterprise specification has the advantage that many people are already familiar with it and can read enterprise specifications without much additional instruction.

The investigation is carried out by means of a small case study in enterprise specification.

3.1 An example community

As an example of an enterprise viewpoint specification, we specify the enterprise of a university library. This example is loosely based on the Templeman Library of the University of Kent at Canterbury, but most of the specification will apply to any library.

Anyone will have some idea of what goes on in a library and there clearly is scope for distributed information systems to support the processes of the library. Nowadays, most libraries have one or more automated systems in place to keep track of their collection, the outstanding loans and the borrowers. In the following, we will consider an ODP enterprise viewpoint description of such a system and its environment.

In essence, a library maintains a collection of books, periodicals, and other items, that may be borrowed by its members. A library community comes into being with the establishment of its collection. In the case of a university library, this may have been ordered by decree when the university was founded, but a collection can also be established simply by some people getting together and putting their books in a shared space. In any case, the primary objective of a library community is to share this collection amongst the members.

In the UML, a community can be represented by a package (see figure 9). The objective, specified in natural language, can be attached to the community in a description field of the package.

3.2 Modelling roles and their relationships

The roles of the library community are easily identified. From the Templeman Library Regulations (see http://www.ukc.ac.uk/library/about.htm), for example, we can derive the following descriptions.

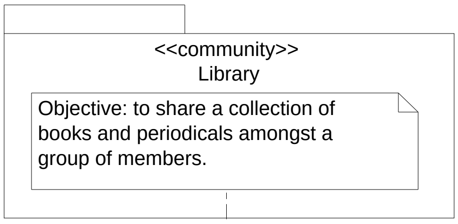

Figure 9: The library community and its objective

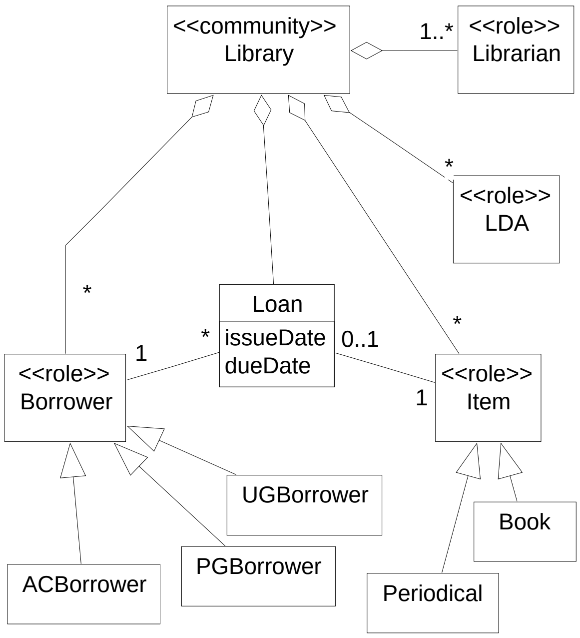

Figure 10: The library community structure

- The role of **Borrower** may be fulfilled by academic staff, and postgraduate and undergraduate students of the University, a related community. In addition, other people and libraries may request special permission to use the Library. They can fulfil the **Borrower** role if such permission is granted.

- The role of **Loans Desk Assistant (LDA)** may be fulfilled by any member of the Library staff. In this role they may issue and receive books and other items available for loan to users.

- Within the library community, the **Librarian** role represents the person with ultimate power. This power may be delegated: "The Librarian may delegate all or any powers under these Regulations to such member or members of the Library staff as may be appropriate."

- The **Item** role is fulfilled by all books and periodicals in the library's collection.

The roles of borrower and librarian are clearly actor roles: they may initiate interactions. Borrowers can, for example, borrow or return items, and librarians can recall items. It is less obvious that loans desk assistants are actors or artefacts. They are involved in the borrow and return actions, but they do not initiate them. Still, we view loans desk assistant as an actor role, because it is fulfilled by human-users. Item clearly is an artefact role. Items do not initiate any interactions, but are involved in most interactions between borrowers and loans desk assistants.

The Loan class, also depicted in figure 10, is an example of a conceptual relationship between the borrower and item roles. Whenever a borrower borrows an item, an instance of this relationship is established. The relationship will again be dissolved upon the return of the item. The advantage of explicitly modelling such conceptual relationships between roles is that we can attach certain attributes with the relationship. In the case of a loan, we may be interested in the issue date and the date the item is due for return.

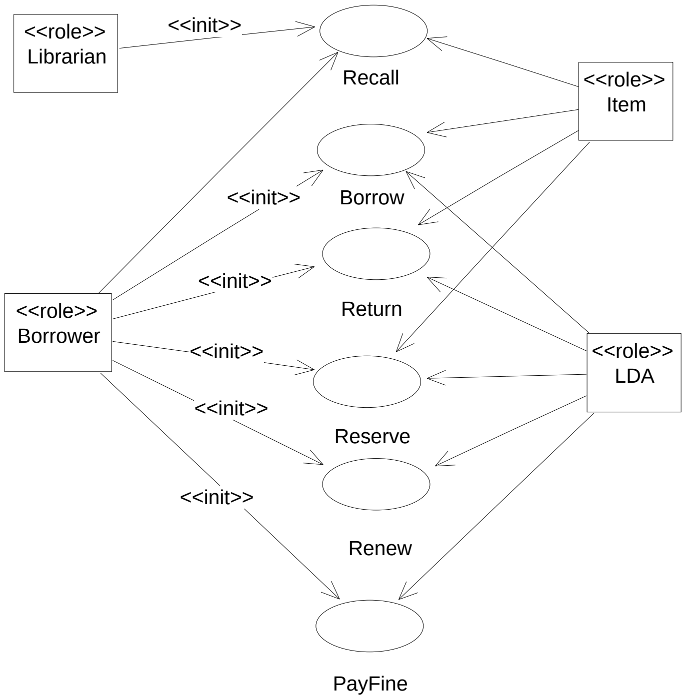

Figure 11: Library Community

3.3 Modelling enterprise behaviour

The objects fulfilling roles in the library community may be involved in the following enterprise actions:

- The **Borrow** action in which a Borrower requests to borrow a number of items. The request will be handled by a Loans Desk Assistant who may issue the items or deny the request depending on the status of the Borrower and the items.
- The **Return** action in which a Borrower returns previously borrowed items to the loans desk. A Loans Desk Assistant will verify that the items are returned in time and, if necessary, collect a fine.
- The **Reserve** action in which a Borrower places a reservation for an item on loan to another Borrower.
- The **Renew** action in which a Borrower requests to extend the loan period for an item already on loan to him or her. The request will be handled by a Loans Desk Assistant who should refuse the renewal in case the item has been reserved by another Borrower.
- The **Recall** action in which a Borrower is informed to return an item on loan to him or her. After an item has been in the possession of any Borrower for a week, it may be recalled if required by another user. In addition, the Librarian may recall an item on loan at any time.
- The **PayFine** action in which a Borrower pays off an certain amount of his outstanding fines at the loans desk.

The library community can now be refined to a use case diagram (see figure 11) in which the library roles are depicted as UML actors and the enterprise actions they are involved in as use cases. The association stereotype ≪init≫ is introduced to indicate which role initiates the interaction.

The use case diagram in figure 11 simply identifies the possible enterprise actions. It does not define the enterprise behaviour. However, we can give a simple example of how enterprise behaviour might be specified. Consider the behaviour of the borrower role. Its actual behaviour will consist in part of borrowing books and then perhaps returning them. However, a borrower can only return a book if it was previously borrowed by that borrower. Therefore we could describe the borrower's behaviour as a causal or temporal ordering of borrow and return actions, the latter happening after the former. We could for example use an activity diagram to specify this constraint on the borrower's behaviour. However, this would not quite capture the fact that this sequence of actions can be interleaved with similar actions for a different loan (of a different item). The precise behaviour can be captured more precisely in LOTOS [1] as follows:

Borrower := borrow ?item:Item; (return !item; **stop** ||| Borrower)

This specifies that a borrower can only return an item once she has borrowed it, and that this temporal constraint for a particular loan is repeated for every loan, but that these loans can be interleaved. For example, the sequence

borrow(book1) ; borrow(book2) ; return(book2) ; return(book1)

is possible according to this specification, assuming book1 and book2 were books that the borrower was entitled to borrow.

The advantage of representing enterprise actions as use cases is that these may be further refined into interaction diagrams (sequence or collaboration diagrams). In the library case, it is not clear though whether such a detailed description of community behaviour would be desired or appropriate. One has to be careful not to be lured into a computational description of the borrowing process.

3.4 Relating communities

A complete enterprise specification consists of a number of related community specifications. So far, we have considered the library community in isolation, but it is of course closely related to the university community that it serves. In the following, we briefly consider this relationship.

In a university community, we find students, researchers, teachers, heads of department, support staff, etc. In order to relate the university community to the library community, we need at least to identify:

- the role of **Academic**, which is fulfilled by all research and teaching staff of the university; and
- the role of **Student**, which is fulfilled by all people on undergraduate or postgraduate degree courses.

These roles are fulfilled by people, who may at some point also fulfil the role of borrower in the library community. This relationship, of enterprise objects (in this case: people) fulfilling roles in different communities, between the university and the library community is depicted in figure 12.

3.5 Expressing policies

The Templeman Library regulations (also see http://www.ukc.ac.uk/library/about.htm) contain a large number of rules relating to the borrowing of material from the library. Together these define the permissions, obligations and prohibitions for borrowers and the other roles in the library. Most of these policy rules represent constraints on the behaviour of the objects fulfilling those roles. Examples are that "undergraduate borrowers are forbidden to borrow periodicals", that "academic borrowers are permitted to borrow 24 items" and that "borrowers are obliged to return the books they borrow." This last example nicely demonstrates the difference between policies and enterprise behaviour. The enterprise behaviour given for the borrower role in section 3.3 places constraints

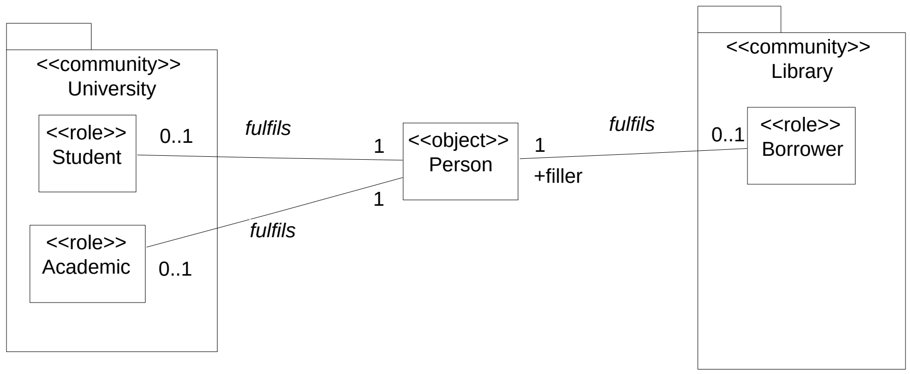

Figure 12: Relating communities

on the temporal ordering of the borrow and return actions. However, no limit is placed on the number of items that can be borrowed at any one time. The latter constraint is a policy.

Unfortunately, there does not seem to be a suitable way of expressing such behavioural policies with the UML. The main problem is that the interaction model of the UML is based on message exchange between objects, whereas interactions in the enterprise viewpoint are seen as bits of shared behaviour. As policies constrain the behaviour of roles, we could consider using the OCL [9], the Object Constraint Language, originally developed by IBM and now incorporated into the UML, in which constraints may be expressed as invariants, or as pre- and post-conditions. However, OCL is also limited in a number of respects: it does not provide powersets, uses the same interaction model as the UML, and lacks a formal semantics. Finally, policy statements often involve timing constraints, (e.g., an item has to be returned by its due date), for which there is currently no provision in the UML/OCL combination. Hence, we have developed a dedicated language for expressing policies to complement the UML in this respect, which is described in [8].

Nevertheless, there are other types of policy statement that can be expressed within the UML. We refer here to the, so called, instantiation policies, which constrain the way in which roles can be populated. The following rule represents a typical example.

- Borrowing rights are given to all academic staff, and postgraduate and undergraduate students of the University.

This means that only those people fulfilling a role in the university community (academic or student) may fulfil the borrower role. The (OCL) does provides us with a suitable language for expressing such constraints. OCL constraints always have to be interpreted within the context of a UML diagram. For the above constraint, figure 12 provides a suitable context as it already relates the two communities involved. The constraint may now be expressed as follows:

context <u>**Borrower**</u>
inv: $\textbf{self}.filler.student \rightarrow \textbf{notEmpty()}$ **or**
$\textbf{self}.filler.academic \rightarrow \textbf{notEmpty()}$

This constraint is an invariant imposed on the Borrower class. It requires that each Borrower role has to be filled by a Person filling either the Student role or the Academic role. Here, **self** represents an arbitrary instance of the Borrower class. The expression $\textbf{self}.filler$ returns the instances of Person related to a particular instance of Borrower, and $\textbf{self}.filler.student$ returns the set of instances of the Student class related to that instance of Person. The latter set is tested for being non empty; it is either empty or contains one instance. Similary, the second line tests whether the borrower is an academic. The two parts are combined with **or** , so the expression will return true is the borrower is either a student or an academic.

Enterprise concept	UML construct	Stereotype
Community	Package	community
Community	Class	community
Object	Class	object
Role	Class	role
Role relationship	Class	
Action	Use Case	

Table 1: Enterprise language mappings

3.6 Summary of mappings

Table 1 summarises our proposal for using the UML as a concrete notation for the ODP enterprise viewpoint. It describes the mapping from the enterprise language concepts to UML constructs and, if appropriate, the introduced stereotypes.

A community can either be represented as a package or, when abstracted as an enterprise object, as a UML class. Enterprise objects, or actually, object templates, are represented by a class with the ≪object≫ stereotype. An ODP role is represented as a class with the ≪role≫ stereotype, not to be confused with the UML concept of role. Role relationships are also represented by a class, while enterprise actions are depicted as use cases on a use case diagram.

4 Conclusion

In this paper, we have used to UML to build up a meta-model of the ODP enterprise viewpoint language. We found that the UML is a useful instrument for such an exercise. It allowed us to depict the relationships between the various concepts defined in the enterprise viewpoint language in a clear, concise and consistent manner. Moreover, the conceptual diagrams are easily communicated between people with varying backgrounds. We managed to have meaningful discussions with both UML literates who knew nothing about ODP, as well as ODP specialists that were not familiar with the UML. Interestingly, the meta-model developed in this paper is consistent with a UML model of the enterprise viewpoint drawn up at the latest editing meeting of the ISO working group for ODP. The two models even coincide for large parts.

In the second half of the paper, we considered the use of the UML as a graphical notation for enterprise viewpoint specification. In our opinion, the UML is certainly useful for depicting the static structure of communities, i.e., the roles and their relationships. For expressing enterprise behaviour, it is only partly useful, because the enterprise viewpoint assumes a different model of interaction. It is not clear yet whether the UML is suitable for expressing behavioural policies. For this reason, we have developed a dedicated policy specification language in [8]. The UML, or actually the OCL, can, to a certain extent, be used for expressing, so called, instantiation policies and the relationship between these policies and behavioural policies is an area for further work.

Tool support is also an issue. There is clearly scope to develop tailored tools based on existing UML tools, providing perhaps structured editors for a number of interlinked viewpoints.

References

[1] T. Bolognesi and E. Brinksma. Introduction to the ISO specification language LOTOS. *Computer Networks and ISDN Systems*, 14:25–59, 1987.

[2] G. Booch, J. Rumbaugh, and I. Jacobson. *The Unified Modelling Language: User Guide*. Object Technology Series. Addison-Wesley, 1998.

[3] ISO/IEC. Open Distributed Processing - Reference Model - Part 2: Foundations. International Standard 10746-2, ITU-T Recommendation X.902, January 1995.

[4] ISO/IEC. Open Distributed Processing - Reference Model - Part 3: Architecture. International Standard 10746-3, ITU-T Recommendation X.903, January 1995.

[5] ISO/IEC. Open Distributed Processing - Reference Model - Enterprise Viewpoint. Working Draft 15414, ITU-T Recommendation X.911, January 1999.

[6] J. Rumbaugh, I. Jacobson, and G. Booch. *The Unified Modelling Language: Reference Manual.* Object Technology Series. Addison-Wesley, 1999.

[7] J. M. Spivey. *The Z notation: A reference manual.* Prentice Hall, 1989.

[8] M. W. A. Steen and J. Derrick. Formalising ODP enterprise policies. In *3rd International Enterprise Distributed Object Computing Conference (EDOC '99).* IEEE, September 1999.

[9] J. Warmer and A. Kleppe. *The Object Constraint Language: Precise Modeling with UML.* Addison-Wesley, 1998.