Modeling the ODP Computational Viewpoint with UML 2.0: The Templeman Library Example

José Raúl Romero and Antonio Vallecillo
Dpto. de Lenguajes y Ciencias de la Computación
Universidad de Málaga, Spain
{jrromero,av}@lcc.uma.es

Abstract

The advent of UML 2.0 has provided a new set of concepts more apt for modeling the structure and behavior of distributed systems. These concepts and mechanisms can be effectively used for representing the ODP concepts, in particular those from the Computational Viewpoint. In this paper we present an example that uses the UML 2.0 profile for the ODP computational viewpoint to illustrate its benefits and limitations.

1 Introduction

The *computational viewpoint* describes the functionality of an ODP system and its environment through the decomposition of the system, in distribution transparent terms, into objects which interact at interfaces. In the computational viewpoint, applications and ODP functions consist of configurations of interacting computational objects.

Although the ODP reference model provides abstract languages for the relevant concepts, it does not prescribe particular notations to be used in the individual viewpoints. The viewpoint languages defined in the reference model are abstract languages in the sense that they define what concepts should be used, not how they should be represented. Several notations have been proposed for the different viewpoints by different authors, which seem to agree on the need to represent the semantics of the ODP viewpoints concepts in a precise manner [2, 4, 8, 12, 10, 11]. For example, formal description techniques such as Z and Object-Z have been proposed for the information and enterprise viewpoints [21], and LOTOS, SDL or Z for the computational viewpoint [8, 20]. Lately, rewriting logic and Maude have also shown their adequacy for modeling the ODP languages [6, 5, 19].

However, the formality and intrinsic difficulty of most formal description techniques have encouraged the quest for more user-friendly notations. In this respect, the general-purpose modeling notation UML (Unified Modeling Language) is clearly the most promising candidate, since it is familiar to developers, easy to learn and to use by non-technical people, offers a close mapping to implementations, and has commercial tool support.

Until the advent of UML version 2.0, both the lack of precision in the UML definition and the semantic gap between the ODP concepts and the UML constructs hindered its application in this context. The UML (1.4) Profile for EDOC [13] tried to bridge this gap. But from our perspective, the gap was so big that the Profile ended up being too large and difficult to understand and use by both ODP and UML users. With the advent of UML 2.0 the situation may have changed, since not only its semantics have been more precisely defined, but it also incorporates a whole new set of concepts more apt for modeling the structure and behavior of distributed systems.

In addition, the wide adoption of UML by industry, the number of available UML tools, and the increasing interest for model-driven development and the MDA initiative, motivated ISO/IEC and ITU-T to launch a joint project in 2004, which aims to define the use of UML for ODP system specifications (ITU-T Rec. X.906 — ISO/IEC 19793 [9]). Thus, ODP modellers could use the UML notation for expressing their ODP specifications in a standard graphical way, and UML modellers could use the RM-ODP concepts and mechanisms to structure their UML system specifications.

In this paper we explore the use of the UML 2.0 profile for modeling the ODP computational viewpoint concepts presented in [17] and detailed in [18]. More specifically, we show how this profile can be used to model operational ODP systems by representing, as an example, the Templeman's library management system.

The structure of this document is as follows. First, sections 2 and 3 serve as a brief introduction to the computational viewpoint and UML 2.0, respectively. Section 4 presents a summary of the UML 2.0 Profile for the ODP Computational Viewpoint, describing how to model compu-

tational specifications in UML. This profile is used in Section 5 for specifying the Templeman's library system. Finally, Section 6 draws some conclusions and outlines some future research activities.

2 Computational Viewpoint in RM-ODP

The computational viewpoint is directly concerned with the distribution of processing but not with the interaction mechanisms that enable distribution to occur. The computational specification decomposes the system into objects performing individual functions and interacting at well-defined interfaces.

The heart of the computational language is the object model which defines the form of interface that an object can have; the way that interfaces can be bound and the forms of interaction which can take place at them; the actions an object can perform, in particular the creation of new objects and interfaces; and the establishment of bindings.

The computational object model provides the basis for ensuring consistency between engineering and technology specifications (including programming languages and communication mechanisms) thus allowing open interworking and portability of components in the resulting implementation.

2.1 Computational language concepts

In the ODP Reference Model, the computational language uses a basic set of concepts and structuring rules, including those from ITU-T Recommendation X.902, ISO/IEC 10746-2, and several concepts specific to the computational viewpoint.

Objects and interfaces. ODP systems are modeled in terms of *objects*. An object contains information and offers services. A system is composed as a configuration of interacting objects. In the computational viewpoint we talk about *computational objects*, which model the entities defined in a computational specification. Computational objects are abstractions of entities that occur in the real world, in the ODP system, or in other viewpoints [8].

Computational objects have *state* and can interact with their environment at *interfaces*. An interface is an abstraction of the behavior of an object that consists of a subset of the interactions of that object together with a set of constraints on when they may occur. ODP objects may have multiple interfaces.

Computational templates. Computational objects and interfaces can be specified by templates. In ODP, an <X> *template* is "the specification of the common features of a collection of <X>s in sufficient detail that an <X> can be instantiated using it". <X> can be anything that has a type. Thus, an interface of a computational object is usually specified by a *computational interface template*, which is an interface template for either a signal interface, a stream interface or an operation interface. A computational interface template comprises a signal, stream or operation *interface signature* as appropriate; a *behavior* specification; and an *environment contract* specification.

An *interface signature* consists of a name, a causality role (producer, consumer, etc.), and set of signal signatures, operation signatures, or flow signatures as appropriate. Each of these signatures specify the name of the interaction and its parameters (names and types).

Interactions. RM-ODP prescribes three particular types of interactions: *signals*, *operations*, and *flows*. A signal may be regarded as a single, atomic action between computational objects. Signals constitute the most basic unit of interaction in the computational viewpoint. Operations are used to model object interactions as represented by most message passing object models, and come in two flavors: *interrogations* and *announcements*. An interrogation is a two-way interaction between two objects: the client object invokes the operation (*invocation*) on one of the server object interfaces; after processing the request, the server object returns some result to the client object, in the form of a *termination*. An announcement is a one-way interaction between a client object and a server object. In contrast to an interrogation, after invocation of an announcement operation on one of its interfaces, the server object does not return a termination. Terminations model every possible outcome of an operation. Flows model streams of information, i.e., a flow represents an abstraction of a sequence of interactions from a producer to a consumer, whose exact semantics depends on the specific application domain. In the ODP computational viewpoint, operations and flows can be expressed in terms of signals [8].

Environment contracts. Computational object templates may have environment contracts associated with them. These environment contracts may be regarded as agreements on behaviors between the object and its environment, including Quality of Service (QoS) constraints, usage and management constraints, etc. These QoS constraints involve temporal, volume and dependability constraints, amongst others, and they can imply other usage and management constraints, such as location and distribution transparency constraints.

An environment constraint can thus describe both requirements placed on an object's environment for the correct behavior, and constraints on the object behavior in the correct environment.

2.2 Structure of ODP computational specifications

A computational specification describes the functional decomposition of an ODP system, in distribution transparent terms, as: (a) a configuration of computational objects; (b) the internal actions of those objects; (c) the interactions that occur among those objects; (d) environment contracts for those objects and their interfaces.

A computational specification also defines an initial set of computational objects and their behavior. The configuration will change as the computational objects instantiate further computational objects or computational interfaces; perform binding actions; effect control functions upon binding objects; delete computational interfaces; or delete computational objects.

3 Unified Modeling Language 2.0

UML is a visual modeling language that provides a wide number of graphical elements for modeling systems, which are combined in diagrams according to a set of given rules. The purpose of such diagrams is to show different views of the same system or subsystem and indicate what the system is supposed to do.

There are mainly two types of diagrams: *structural* and *behavioral*. The former ones focus on the organization of the system. Structural diagrams include package diagrams, object diagrams, deployment diagrams, class diagrams and composite structure diagrams. Behavioral diagrams reflect the system response to inner and outer requests and its evolution in time, and include activity diagrams, use cases, statecharts and interaction diagrams

One of the major improvements of UML 2.0 [15, 16] is the addition of new diagrams and the enhancements made to existing ones: UML 2.0 structure, composite, communication, timing and interaction overview diagrams allow solving many of the UML 1.x limitations. Most of these improvements have been influenced by the integration of the mature SDL language within UML. In addition, UML 2.0 now provides better constructs for modeling the software architecture of large distributed systems, with concepts such as components and connectors, and has promoted the use of OCL (*Object Constraint Language*), now fully aligned with UML 2.0 [14]. Finally, the language extension mechanisms have been greatly enhanced too, with the more precise definition of UML Profiles to allow the customization of UML constructs and semantics for given application domains. These new concepts and mechanisms of UML 2.0 constitute the basis of our proposal.

4 Modeling Computational Viewpoint Concepts in UML 2.0

The UML 2.0 Profile for the ODP Computational Viewpoint (which is fully described in [18]) consists of three main parts. First, it defines the ODP computational viewpoint metamodel, which is an evolution of the metamodel presented in [19], and defines the semantics, properties and related elements of each metaclass. Second, ODP concepts are mapped to UML elements. This mapping contains information about every ODP computational concept, the UML base element that represents each concept, and the stereotype that extends the metaclass so that the specific domain terminology can be used.

This section summarizes how the main concepts of the ODP computational language are mapped to UML 2.0 concepts.

4.1 Computational objects and interfaces

Computational object templates and objects. A key concept of the ODP computational viewpoint is the *computational object*. Each *computational object* is instantiated from its corresponding *computational object template*.

A *computational object template* will be mapped to a UML component, which represents autonomous system units, that encapsulate state and behavior and interacts with their environment in terms of provided and required interfaces. In UML, components are classifiers. A UML classifier can have a set of features, that characterize its instances.

ODP *computational objects* will then be mapped to UML component instances.

Computational interfaces. *Computational objects* interact with their *environment* at *interfaces*. These are instantiated from *computational interface templates*, which comprise the *interface signature* (*signal*, *operation* or *stream* as appropriate), a *behavior* specification and an *environment contract* specification.

There are no exact terms in UML 2.0 to provide one-to-one mappings for these ODP concepts. However, the semantics provided by other modeling elements can be used with slight customizations.

If we consider *computational interfaces* as interaction points at which *computational objects* interact, we find that this concept corresponds to the UML concept of interaction point, i.e., a port at the instance level.

In ODP, a *computational interface template* comprises an *interface signature*, which is defined as the set of *action templates* associated with the interactions of an *interface*. Each of these *action templates* comprises the *name* for the *interaction*, the number, names and types of the parameters

and an indication of *causality* with respect to the *object* that instantiates the *template*.

Then, an ODP *computational interface signature* will be mapped to a set of UML interfaces, each of which is defined as a kind of classifier that represents a declaration of a set of coherent public features and obligations. This means that each interface can be considered as the specification of a contract that must be fulfilled by any instance of a classifier that realizes the interface (e.g., the UML component instance that represents the *computational object*, through its corresponding interaction point).

Different stereotypes will be used to distinguish the interfaces that represent the different kinds of *computational interface signatures*.

4.2 Interactions

In ODP, the basic one-way communication mechanism from an *initiating object* to a *responding object* is the *signal*, which represents a single basic *interaction* between them. *Operations* and *flows* are also *interactions*, although they can be handled in terms of *signals*, as previously mentioned in Section 2.1.

An ODP *signal* will be mapped to a UML message, which is the specification of the conveyance of information from one instance to another. In UML, a message can specify either the raising of a UML signal or the call of an UML operation.

In ODP, in order to specify a *signal* we need to provide its *signature* and its *behavior*.

An *interaction signature* will be represented by an UML reception, which consists of a declaration stating that the interface classifier is prepared to react to the receipt of a signal. In ODP, each *interface signature* comprises a set of *interaction signatures* that conform to the *interface type*. This means that we need to define the proper set of ODP *interactions* as public features of the appropriate UML interface classifier.

The behavior of *interactions* refers to the communication process between *computational objects*, which will be expressed in UML with behavioral diagrams [3]: (a) Interaction models describe how messages are passed between objects and cause invocations of other behaviors; (b) Activity models focus on the sequence, input/outputs and conditions for invoking other behaviors; and (c) Finally, state machine models show how events (e.g., signal events) cause changes to the object state and invoke other behaviors.

Which of them to choose is a matter of the system perspective that the modeler needs to specify, since each of these models is focused on a different aspect of the system dynamics. For instance, timing diagrams could be also useful to represent the *interactions* among *computational objects* when some timed simple constraints need to be observed or applied.

4.3 Environment contracts

Environment contracts place constraints on the *behavior* of *computational objects*, and usually include QoS, usage, and management aspects. The ODP Reference Model does not prescribe how an *environment contract* must be specified; it just defines this concept and its basic contents.

Each system modeler might like to specify their own constraints in the way that best suits their particular application, and therefore the UML elements (and their semantics) required to model different *environment contracts* can change from one application to another. Thus, instead of incorporating these kind of concepts into our UML Profile, we have decided to use separate profiles for representing QoS and other extra-functional aspects of *environment contracts*. The possibility offered by UML 2.0 to apply multiple profiles to a package—as long as they do not have conflicting constraints—will allow the specifier use the QoS profile(s) of his preference.

4.4 Computational specifications

As described in 2.2, a computational specification describes the functional decomposition of an ODP system, in distribution transparent terms. In UML, the computational specification will be represented by a set of diagrams that model both structural and behavioral aspects of the system. These diagrams will use the elements provided by the applied profiles (using their specified semantics).

A *configuration* of *computational objects* and their interacting *interfaces* will be modeled by component diagrams (at the instance level).

The *internal actions* of those *objects* will be represented by behavioral diagrams associated to the UML components that represent those *objects*.

4.5 Summary of the mappings

The fact that most ODP concepts can be represented by UML 2.0 concepts without changing their original semantics (maybe imposing some additional constraints on them, at most) enables the use of a UML Profile as the right kind of mechanism for our purposes [7]. Note that the profile mechanism does not allow for modifying existing metamodels. Rather, a profile is intended to provide a straightforward mechanism for adapting an existing metamodel with constructs that are specific to a particular domain.

As a summary, Table 1 shows the most important stereotypes defined in the UML Profile for the ODP Computational Viewpoint [18].

Table 1. Summary of the Computational Viewpoint Profile

ODP Concept	UML Base Element	Stereotype
Computational object template	Component	≪CV_CompObjectTemplate≫
Computational interface template	Port	≪CV_CompInterfaceTemplate≫
Signal interface signature	Interface(s)	≪CV_SignalInterfaceSignature≫
Operation interface signature	Interface(s)	≪CV_OperationInterfaceSignature≫
Stream interface signature	Interface(s)	≪CV_StreamInterfaceSignature≫
Announcement signature	Reception	≪CV_AnnouncementSignature≫
Interrogation signature	Reception	≪CV_InterrogationSignature≫
Termination signature	Reception	≪CV_TerminationSignature≫
Signal signature	Reception	≪CV_SignalSignature≫
Flow signature	Reception	≪CV_FlowSignature≫
Computational object	InstanceSpecification	≪CV_Object≫
Signal interface	Port (interaction point)	≪CV_SignalInterface≫
Operation interface	Port (interaction point)	≪CV_OperationInterface≫
Stream interface	Port (interaction point)	≪CV_StreamInterface≫
Signal	Message	≪CV_Signal≫
Flow	Interaction / Message	≪CV_Flow≫
Announcement	Message	≪CV_Announcement≫
Invocation	Message	≪CV_Invocation≫
Termination	Message	≪CV_Termination≫

5 A Case Study

We will illustrate the use of the UML Profile for the ODP Computational Viewpoint by modeling the computerized system that supports the operations of a Templeman Library at the University of Kent at Canterbury, in particular those operations related to the borrowing process of the Library items.

The system should keep track of the items of the University Library, its borrowers, and their outstanding loans. The library system will be used by the library staff (librarian and assistants) to help them record loans, returns, etc. The borrowers will not interact directly with the library system.

The basic rules that govern the borrowing process of that Library are as follows:

1. Borrowing rights are given to all academic staff, and to postgraduate and undergraduate students of the University.
2. Library books and periodicals can be borrowed.
3. The librarian may temporarily withhold the circulation of Library items, or dispose them when they are no longer apt for loan.
4. There are prescribed periods of loan and limits on the number of items allowed on loan to a borrower at any one time.
5. Items borrowed must be returned by the due day and time which is specified when the item is borrowed.
6. Borrowers who fail to return an item when it is due will become liable to a charge at the rates prescribed until the book or periodical is returned to the Library, and may have borrowing rights suspended.
7. Borrowers returning items must hand them in to an assistant at the Main Loan Desk. Any charges due on overdue items must be paid at this time.
8. Failure to pay charges may result in suspension by the Librarian of borrowing facilities.

Despite they leave many details of the system unspecified, these *textual regulations* will be the starting point for the ODP specifications below.

5.1 Computational objects and interfaces

In order to represent the computational specification for the Templeman Library, we need to identify the computational elements that participate in the borrowing process. Each of these elements (i.e., computational objects and interfaces) are instantiated from their corresponding computational templates. In UML, we represent the system structure using a component diagram, that describes the computational object templates and the computational interfaces at which these objects interact.

As shown in Figure 1, there are four different kinds of computational objects: (a) a manager (UserMgr) for each user (i.e., borrower); (b) the system that manages

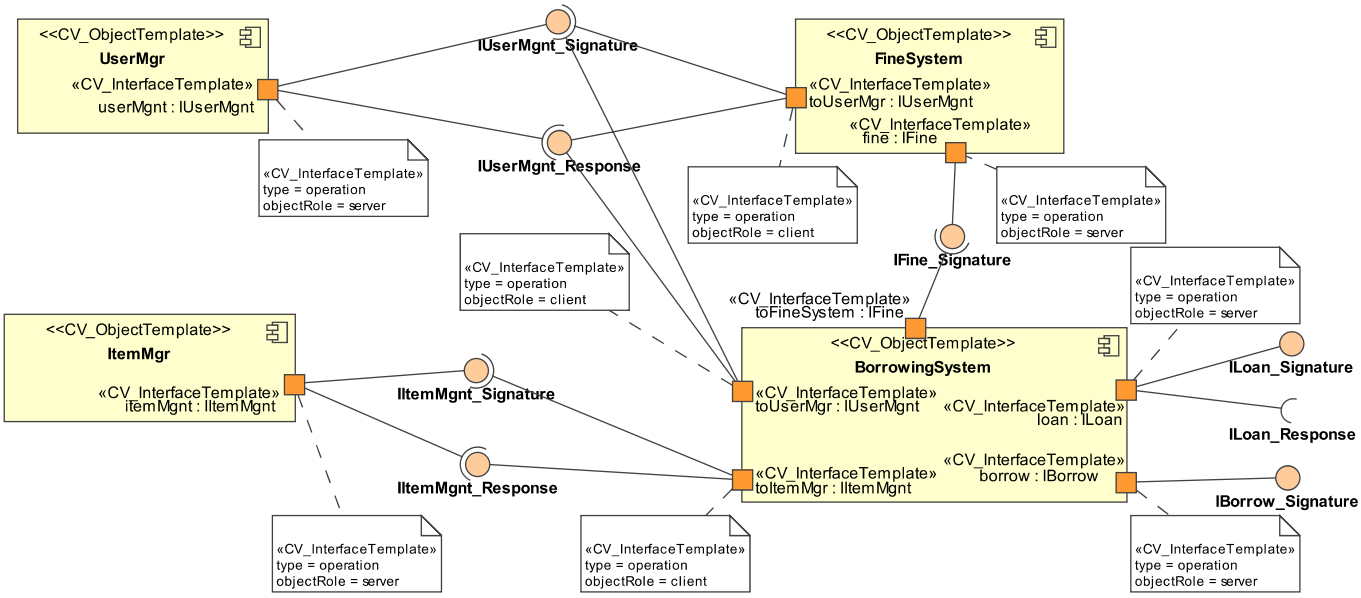

Figure 1. Component Diagram representing ODP Computational Templates

the fines applied to users who exceed the borrowing period (FineSystem); (c) the system that manages the library items (ItemMgr); and (d) the borrowing process coordinator (BorrowingSystem).

These objects interact with each other and with their environment at computational interfaces, which are instantiated from their corresponding interface templates. In this case, we use five computational interfaces, all of them operational interfaces. As shown in Figure 1, each interface is modeled by a UML port feature and its provided and required UML interfaces, whose receptions represent the individual interaction signatures. For readability reasons, we have shown interface signatures as *balls* and *sockets* in Figure 1. An extended notation for the signature of the IUserMgnt interface is shown in Figure 2, where UML receptions are explicitly depicted.

In this example, only operation computational interfaces have been defined. Therefore, just two causalities are possible: *client* or *server*. This implies that the tag objectRole can be omitted because the causality is implicitly represented by the kind of dependency existing between the UML port and the UML interface—e.g., an usage dependency (required interface) represents that the computational interface will interact as a *client*. There are also cases in which the system designer might prefer to adopt an operational object-oriented approach, which represents the exchange of information between objects in terms of operation interactions between computational objects. In this case, modeling these interactions as UML operations might probably be simpler, as shown in Figure 3.

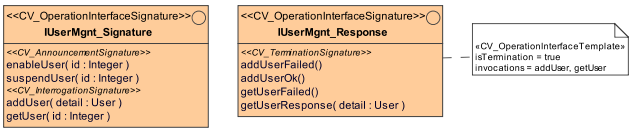

Figure 2. Interface signature for IUserMgnt

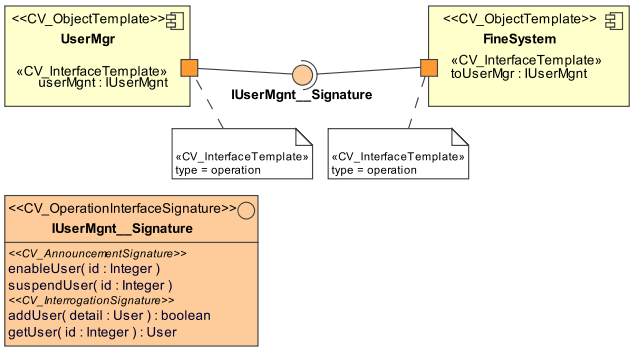

Figure 3. Component diagram following an operational OO approach

5.2 Behavioral specification

We need to specify different behavior aspects of the computational elements. In fact, activity, communication, interaction and sequence diagrams might be useful to represent

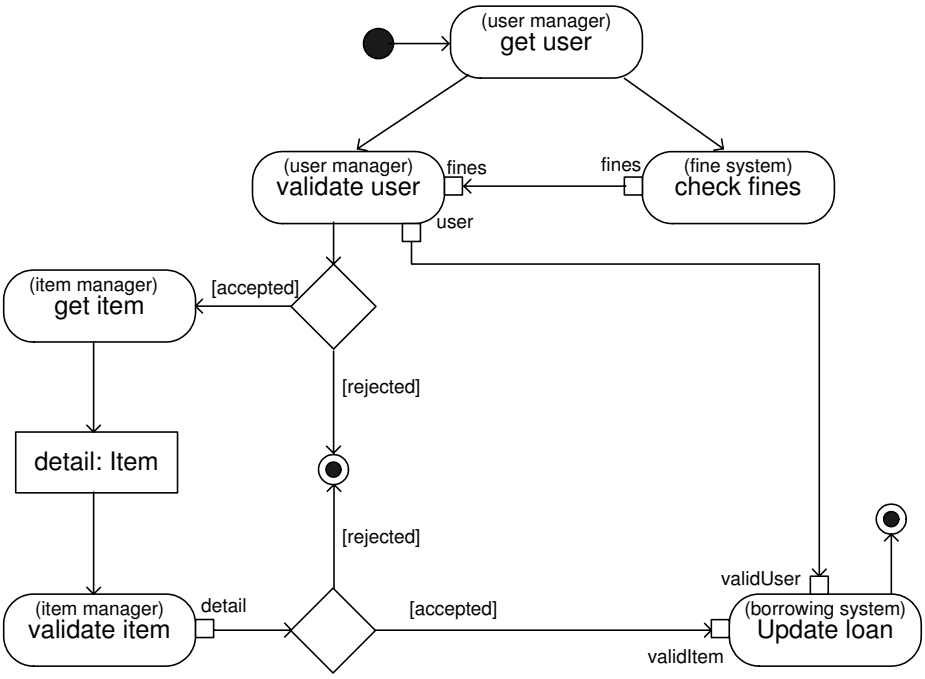

Figure 4. Activity diagram for the Borrowing **Process**

both the internal actions of the computational objects, and the interactions that occur between them. In case we want to specify how object interactions are performed, activities can be useful because they are an abstraction of the many ways that messages are exchanged between objects [3]. This makes activities useful at the stage of development where the primary concern is dependency between tasks, rather than interaction protocols. The activity diagram for the *borrowing process* is shown in Figure 4.

Alternatively, when messages and interaction protocols are the focus of development, UML interaction diagrams are more appropriate, as shown in Figure 5.

6 Conclusions

In this paper we have shown with an example how the ODP computational specifications can be expressed in UML 2.0, using the Profile for the Computational viewpoint described in [17]. We find results to be encouraging, since the profile has proved to be expressive enough to describe the system functionality and processes, in a natural way. It is still to be proved whether ODP and UML modelers find it natural, too, but we hope this example can help these two kinds of audiences understand better the proposal.

There are some lines of work that we plan to address shortly. In particular, once we count with a graphical notation to model the ODP computational viewpoint, we perceive that its connection to formal notations and tools might bring along many real advantages. For instance, formal analysis of the system can be achieved from the UML environment (such as model checking or theorem proving), freeing the system analyst from most formal technicalities. In this sense, we are working in the provision of bridges between the UML 2.0 specification and the Maude language, so that the Maude formal toolkit can be used with the UML models produced for the ODP system.

In addition, the computational viewpoint is just one of the five ODP viewpoints. Defining and analyzing the correspondences between the different viewpoint specifications is also required. The aforementioned ITU-T Rec. X.906 — ISO/IEC 19793 standard is defining UML profiles for all viewpoints [9]. The example presented here tries to serve as input to this work, both to illustrate the use of the Computational Profile and to provide with examples that help tackling how to define and analyze viewpoint correspondences.

Acknowledgements This work has been supported by Spanish Research Project TIC2002- 04309-C02-02.

References

[1] D. H. Akehurst, J. Derrick, and A. G. Waters. Addressing computational viewpoint design. In *Proc. of EDOC 2003*, pages 147–159, Brisbane, Australia, Sept. 2003.

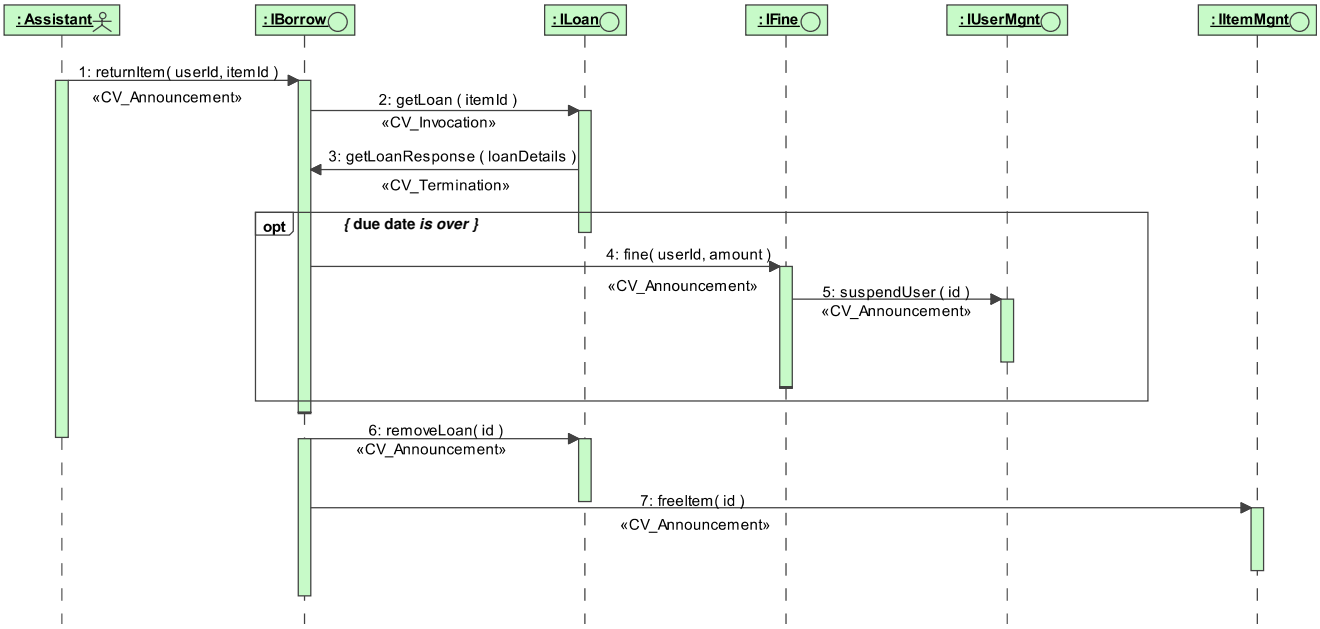

Figure 5. Interaction diagram for the Return Item **Process (excerpt)**

[2] C. Bernardeschi, J. Dustzadeh, A. Fantechi, E. Najm, A. Nimour, and F. Olsen. Transformations and consistent semantics for ODP viewpoints. In *Proc. of FMOODS'97*, pages 371–386, Canterbury, 1997. Chapman & Hall.

[3] C. Bock. UML 2 activity and action models part 2: Actions. *Journal of Object Technology*, 2(5):41–56, 2003.

[4] H. Bowman, J. Derrick, P. Linington, and M. W. Steen. FDTs for ODP. *Computer Standards & Interfaces*, 17:457–479, Sept. 1995.

[5] F. Durán, M. Roldán, and A. Vallecillo. Using maude to write and execute ODP Information Viewpoint specifications. *Computer Standards & Interfaces*, 2005.

[6] F. Durán and A. Vallecillo. Formalizing ODP Enterprise specifications in Maude. *Computer Standards & Interfaces*, 25(2):83–102, June 2003.

[7] L. Fuentes and A. Vallecillo. An introduction to UML profiles. *UPGRADE, The European Journal for the Informatics Professional*, 5(2):5–13, Apr. 2004.

[8] ISO/IEC. *RM-ODP. Reference Model for Open Distributed Processing*. Geneva, Switzerland, 1997. International Standard ISO/IEC 10746-1 to 10746-4, ITU-T Recommendations X.901 to X.904.

[9] ISO/IEC. *Use of UML for ODP System Specification*. Geneva, Switzerland, (to appear in 2006). International Standard ISO/IEC 19793, ITU-T Recommendation X.906.

[10] D. R. Johnson and H. Kilov. Can a flat notation be used to specify an OO system: using Z to describe RM-ODP constructs. In *Proc. of FMOODS'96*, pages 407–418, Paris, Mar. 1996. Chapman & Hall.

[11] D. R. Johnson and H. Kilov. An approach to a Z toolkit for the Reference Model of Open Distributed Processing. *Computer Standards & Interfaces*, 21(5):393–402, Dec. 1999.

[12] P. Linington. RM-ODP: The architecture. In K. Milosevic and L. Armstrong, editors, *Open Distributed Processing II*, pages 15–33. Chapman & Hall, Feb. 1995.

[13] OMG. *A UML Profile for Enterprise Distributed Object Computing V1.0*. Object Management Group, Aug. 2001. OMG document `ad/2001-08-19`.

[14] OMG. *OCL 2.0*, Oct. 2003. Final Adopted Specification ptc/03-10-04.

[15] OMG. *Unified Modeling Language Specification (version 2.0): Infrastructure*, 2003. ptc/03-12-01.

[16] OMG. *Unified Modeling Language Specification (version 2.0): Superstructure*, 2003. Draft Adopted Specification ptc/03-08-02.

[17] J. R. Romero and A. Vallecillo. Modeling the ODP Computational Viewpoint with UML 2.0. In *Proc. of EDOC 2005*, Enschede, Netherlands, Sept. 2005. IEEE CS Press.

[18] J. R. Romero and A. Vallecillo. UML 2.0 Profile for the ODP Computational Viewpoint. Technical Report TR-05-03, Universidad de Málaga, Mar. 2005. Available from `http://www.lcc.uma.es/~jrromero`

[19] J. R. Romero and A. Vallecillo. Formalizing ODP computational specifications in Maude. In *Proc. EDOC 2004*, pages 212–233, Monterey, California, Sept. 2004. IEEE CS Press.

[20] R. Sinnot and K. J. Turner. Specifying ODP computational objects in Z. In *Proc. of FMOODS'96*, pages 375–390, Canterbury, 1997. Chapman & Hall.

[21] M. W. Steen and J. Derrick. ODP Enterprise Viewpoint Specification. *Computer Standards & Interfaces*, 22(2):165–189, Sept. 2000.