Workshop on ODP for Enterprise Computing (WODPEC 2005)

P. Linington, A. Tanaka, S. Tyndale-Biscoe, and A. Vallecillo (Eds.)

Enschede, The Netherlands, September 19, 2005
http://www.lcc.uma.es/~av/wodpec2005/

Proceedings

In conjunction with:

The 9th International IEEE Enterprise Distributed Object Computing Conference
19-25 September 2005, Enschede, The Netherlands
http://www.edocconference.org

Editors

Peter F. Linington
Computing Laboratory
University of Kent
Canterbury
Kent, CT2 7NF (UK)
P.F.Linington@kent.ac.uk
http://www.cs.kent.ac.uk/people/staff/pfl/

Akira Tanaka
Open Standards and Architecture
Hitachi, Ltd.
Software Division
5030 Totsuka-cho, Totsuka-ku, Yokohama, 244-8555 Japan
akira.tanaka.pr@hitachi.com

Sandy Tyndale-Biscoe
Open-IT Limited
Cedarcroft, Sunnyway, Bosham,
Chichester, W. Sussex, PO18 8HQ (UK)
sandy@Open-IT.co.uk
http://www.open-it.co.uk

Antonio Vallecillo
University of Málaga
ETSI Informática
Campus Teatinos
29071 Málaga (Spain)
av@lcc.uma.es
http://www.lcc.uma.es/~av

ISBN 84-689-3693-6.

Impreso en España – Printed in Spain
September 2005

Table of Contents

List of Authors

Program Committee

Preface

The present volume contains the papers selected for presentation at the EDOC 2005 Workshop on ODP for Enterprise Computing (WODPEC 2005), which aims to provide a venue where researchers and practitioners on the use of the RM-ODP in the realm of Enterprise Distributed Computing can meet, disseminate and exchange ideas and problems, identify key issues related to these topics, and explore together possible solutions and future work.

As stated in the Call for Papers (www.lcc.uma.es/~av/wodpec2005/wodpec2005-CFP.pdf), the Workshop focuses this year on problems and ideas related to the use of UML for ODP system specifications, and its relationship with other architectural practices and approaches (e.g., MDA, SOA, CBA, EDA), in the realm of Enterprise Distributed Computing.

All papers submitted to WODPEC 2005 were formally peer reviewed by at least two referees, and 9 papers were finally accepted for presentation at the workshop. These contributions cover a wide range of topics related to the RM-ODP, from general issues of the Reference Model, to modelling approaches for specific ODP viewpoints.

We would like to thank the EDOC 2005 organization for giving us the opportunity to organize the workshop, especially to the General Chairs, Marten J. van Sinderen, Maarten W.A. Steen and Marc M. Lankhorst, and to the Workshop Chair, Bryan Wood, for their assistance and support. Many thanks to all those that submitted papers, and particularly to the contributing authors. Our gratitude also goes to the paper reviewers and the members of the WODPEC 2005 Program Committee, who helped in choosing and improving the selected papers. Finally, we want to acknowledge the Department of Lenguajes y Ciencias de la Computación of the University of Málaga for supporting the production and distribution of this volume, and the Spanish CICYT research project TIC2002-04309-C02 for funding some of its costs.

Enschede, The Netherlands, September 2005

Peter Linington, Akira Tanaka, Sandy Tyndale-Biscoe and Antonio Vallecillo
WODPEC 2005 Organizers

A new viewpoint for change management in RM-ODP systems

Nesrine Yahiaoui[1, 2], Bruno Traverson[1], Nicole Levy[2]

[1] *EDF R&D 1 avenue du Général de Gaulle F-92140 Clamart France*
[2] *UVSQ PRiSM 45 avenue des Etats-Unis F-78035 Versailles France*

nesrine.yahiaoui@prism.uvsq.fr, bruno.traverson@edf.fr, nicole.levy@prism.uvsq.fr

Abstract

RM-ODP (Reference Model - Open Distributed Processing) is a standardized modelling framework to describe a distributed application, according to different viewpoints. The standard has been published for about nearly ten years but in a quite abstract form and in leaving some issues pending (how to describe the viewpoints, management of consistency between views). Several academic and/or industrial projects, such as ODAC (LIP6) and DASIBAO (EDF R&D), have proposed various approaches to solve these problems.

Our proposal supplements these past or in progress works so that the evolution of distributed application is better taken into account. Indeed, because of the "one-shot" approach generally adopted, when a view evolves, it is necessary to rebuild manually consistency with the other views. In addition, the methodologies generally suggested impose a "top-down" approach which is not adapted when existing systems may evolve according to a viewpoint.

We define a modelling framework that maintains consistency using explicit correspondence links which are established between the various views of the distributed application. These links memorize the relationships between the views, thus guaranteeing traceability.

Keyswords. Multi-view system, ODP, UML, change management.

1. Introduction

In the real world, a three dimensional object may be mapped according to several viewpoints (front face, back face, left face, right face). The result of the projection of an object according to a viewpoint is called a view of the object. For example, if we project a cube on its six faces, we observe that each view is a square. The property of a cubic object is that the six squares that we obtained are identical. If we modify, for example, the size of one square, we must modify also the other views (squares), in order to preserve the constraint of the cube. The various views are not independent, since the global constraint must be enforced in the various views.

The concept of viewpoint also exists in the computer science world. A system may be described according to several viewpoints. A viewpoint allows to break up the system and to focus on a particular aspect of the system. A viewpoint introduces specific concepts which take into account the aspect considered by this viewpoint.

Thus, RM-ODP standard enables to describe systems according to five viewpoints: enterprise (objective, business rules), information (data), computational (functional decomposition), engineering (communication and deployment) and technology (hardware and software infrastructure). If we apply these five viewpoints to the same system, the obtained views must be consistent, i.e. the specification of a view should not conflict with the specification of another view. When a modification occurs in a view, it may involve a modification in other views in order to preserve the consistency of the system.

In our study, we are interested in the evolution of multi-view systems that are described according to RM-ODP viewpoints. The evolution means that each view may be modified for various reasons, for example: change of quality of service, environment, and objective. When a modification occurs in a view, several changes may be performed in other views, in order to maintain the consistency of the system.

In this article, we propose a modelling framework that maintains consistency using correspondence links, which are established between the various views of the system. These links memorize the relationships between the views, thus guaranteeing traceability.

This article is structured in five parts: The first part presents the RM-ODP standard that we use to describe multi-view systems. The second part describes some projects using and supplementing the RM-ODP standard. The third part introduces our modelling framework and insists on the interest to make explicit the correspondence links in a multi-view system. The fourth part gives some examples using our modelling framework for change management. Finally, we conclude in the fifth part.

2. RM-ODP reference model

RM-ODP reference model (Reference Model - Open Distributed Processing) is a standardized modelling framework to describe a distributed application according to five viewpoints [7] [8] [9] [13]. Each viewpoint gathers a set of concepts that take into account the concern associated with the viewpoint. Here are the five viewpoints of the standard:

1. ***Enterprise.*** It introduces the concepts necessary to represent a system in the context of an enterprise on which it operates. It is interested to the objective and the policies of a system. A system is then represented by a community which is a configuration of enterprise objects formed to achieve a goal.

2. ***Information.*** It is focused on the semantics of information and the treatment carried out on information. A system is then described by information objects, relationships and behavior. The description is expressed through the use of three diagrams named invariant, static and dynamic.

3. ***Computational.*** It allows a functional decomposition of the system. The various functions are fulfilled by objects that interact thanks to their interfaces. The basic concepts define the type of the interfaces which the computational objects support, the way in which the interfaces can be bound, and the forms of interaction which can take place.

4. ***Engineering.*** It is focused on deployment and communication mechanisms used in the system. It defines communication concepts (channel, stub, skeleton) and deployment concepts (cluster, capsule,...).

5. ***Technology***. It describes the implementation of a system in term of configuration of technical objects representing the hardware and software components used in the implementation. The goal of such a description is to provide additional information for the implementation and the test, by selecting standard solutions for the components and the communication mechanisms.

3. RM-ODP projects and standardization

The standard has been published for about nearly ten years but in a quite abstract form and in leaving many issues pending (how to describe the viewpoints, management of consistency between views). That explains why it is still little used in industrial applications. Nevertheless, several academic and/or industrial projects were carried out to solve these problems. They generally choose UML (Unified Modelling Language) [14] as modelling language and define one or more ways of guaranteeing consistency between views.

ODAC (Open Distributed Application Construction) is a project of the LIP6 laboratory. It defines a method for constructing systems according to an ODP semantic. It prescribes an order in the use of viewpoints [12]. This method proposes three UML profiles corresponding to enterprise, information and computational viewpoints. First, the designer describes the enterprise view, then the information view and after that the computational view, by complying to the correspondence rules specified by the method. These rules describe complementary and consistent views. The set of these three views are named behavioural specification because they describe the behaviour of the system independently of any platform. ODAC does not provide UML profiles for engineering and technology viewpoints, but it proposes an operational specification which results in the transformation of behavioural specification according to a target environment reflecting the real environment for execution, this environment being described by a UML profile. Two UML profiles exist for mobile agent environments [5] and for ANTS active networks [1].

DASIBAO (French acronym that means method for building Information System Architecture based on ODP) is a project from the EDF R&D (Electricity of

France Research and Development). It defines a method which is close to ODAC, because it constructs an ODP system by following a list of sequential steps [2]. Contrary to ODAC, it enables to build the five ODP views. First of all, the designer begins with the enterprise view, then the information view. The two obtained views are used to build the computational view, after that, she constructs the engineering view and finally the technical view. The transition from a view to another follows correspondence rules. The first three viewpoints are well described by UML profiles. However, the method proposes a graphical notation for the two last viewpoints, and a choice of UML profiles is not yet carried out.

In addition, OMG (Object Group Management) published in 2004 a set of UML profiles called EDOC (Enterprise Distributed Object Computing) [4]. In particular, ECA specification (Enterprise Collaboration Architecture) proposes profiles for the enterprise, information and computational viewpoints. The profiles corresponding to the engineering and technique viewpoints in other specifications and the rules of projection of the concepts of ECA are defined. Currently, the use of UML as a notation to describe the ODP viewpoints is in the process of being standardized by the ISO (International Organization of Standardization) [6].

4. Our modelling framework

The various views of an ODP system must be consistent between themselves. Consistency may be considered like a relationship between the views of the system in order to preserve the properties which are expressed in a different way according to the considered view.

In the methods which allow to build ODP systems according to a "top-down" approach [2] [12], the consistency of the system is verified at the time of the construction of the various views. The rules of transformation are applied to one or more views in order to construct one or more consistent views. Once the views are established, no information is saved on the relationships which exist between the any views, i.e. an element in a view does not have knowledge on the element or the elements from which it is build. However, even if these approaches build consistent systems, this consistency is lost as soon as one of the views is modified.

Our approach aims at providing a modelling framework, so that the designer can explicitly describe not only the views of his or her system (corresponding to the ODP viewpoints) but also the correspondences which exist between these views. More precisely, we want to allow the designer to connect the elements in the views which specify the same property of the system.

To illustrate our framework, we have chosen three viewpoints, the enterprise, the computational and the engineering viewpoints. We express for each viewpoint a meta-model that describes the appropriate concepts and the relationships that exist between them, the meta-models are inspired by [6]. In this article, we will not present the three meta-models but only the concepts that enable to connect the three views.

4.1 Correspondence rules

RM-ODP standard describes each concept defined in a viewpoint independently to other concept defined in other viewpoint, and does not stress on the concepts that allow to express the relationships which may exist between different viewpoints. We propose to extend ODP viewpoints to introduce a new concept named ***correspondence rule***. Correspondence rule describes a constraint applied to concepts from two different viewpoints. A set of rules is thus already defined in an abstract way by the standard. These correspondence rules are between two viewpoints. Here are some correspondence rules:

1. Between enterprise and computational viewpoints, for instance, an enterprise object corresponds to computational object or a configuration of computational objects.
2. Between enterprise and engineering viewpoints, for instance, a policy may correspond to and determines transparency requirements that engineering objects support.
3. Between computational and engineering viewpoints, for instance, a computational binding, primitive or via a binding object, corresponds to a local or distributed binding (via a channel).

We propose to introduce these rules into the viewpoints in order to be able to connect ODP concepts defined in different viewpoints. The rules may be described as OCL constraints (Object Constraint Language) [11], they constrain the concepts having to be related and their multiplicity.

4.2 Correspondence links

In order to connect the views, we suggest to introduce a new viewpoint named link viewpoint. The link viewpoint has concepts that permit to connect the views two by two, in our case the enterprise view with the computational view, the computational view with engineering view. It is also possible to connect enterprise view with the engineering view but it represents a redundancy which we prefer to avoid, because these are implicitly linked by a transitivity relationship [Fig 1]:

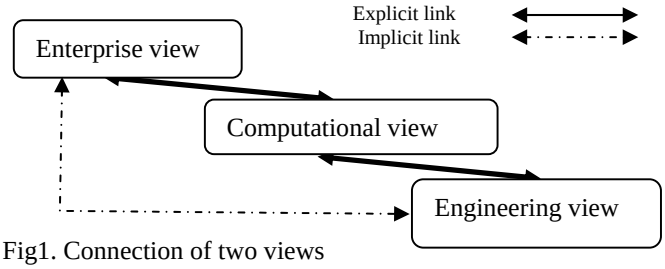

Fig1. Connection of two views

The relationship between two views of the application is called a correspondence link. A correspondence link is established between elements belonging to two different views to indicate that there is a certain relationship between them. The link must comply to the correspondence rules stated between the two considered viewpoints. The link may be governed by one or more correspondence rules.

The link must be bidirectional, i.e. it must be possible to navigate it from any of the two views. The bidirectionnality enables us in particular to let no constraint on the way of navigating the system views. As we will see in the following part, each view may be modified. The interest of a bidirectional link is to find the correspondences with the other views, no matter which view is modified. Moreover, the links established between two ODP views has 1-N cardinality; indeed, it may be possible to connect an element in a view to one or more elements in another view. It is not a priori necessary to define links of N-M cardinality within the framework of our study because we did not find any use of this type of link in the ODP standard.

We propose the following link meta-model which specify the concepts that describe a correspondence between two views [Fig 2].

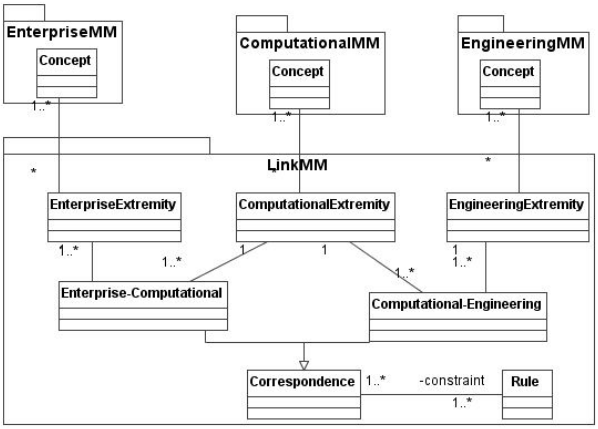

Fig 2. Link meta-model

The correspondence relationships are instances of the correspondence class of the link meta-model (***LinkMM***). The **correspondence** relationship is restricted by **rule**s, these rules are those described previously.

The correspondence may be either a correspondence between enterprise and computational views (**enterprise –computational**), or between computational and engineering views (**computational-engineering**). Each extremity correspondence (**EnterpriseExtremity, ComputationalExtremity, EnginieeringExtremity**) identifies elements belonging to the suitable view and having to be connected. Thus an instance of the correspondence class connects elements in a view with elements in another view.

The figure [Fig 3] illustrates an example of correspondence relationship. ***R1*** and ***R2*** are two correspondence links, ***R1*** connects the ***EntC1*** object of the enterprise view with the ***ComC1, ComC2*** and ***ComC3*** objects of the computational view, under the constraint of the ***rule1*** which establishes that EntC1 is an enterprise object and ComCi (i=1,2,3) are computational objects. ***R2*** connects the ***ComC3*** object of the computational view with the ***EngC1*** and ***EngC2*** objects of the engineering view, under the constraint of the ***rule2*** which establishes that ComC3 is computational object and that EngCi (i=1,2) are basic engineering objects.

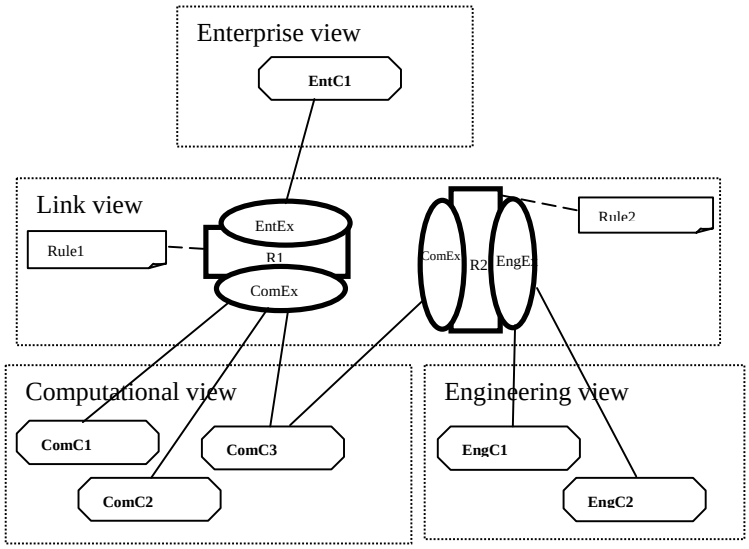

Fig3. Representation of correspondence link

5. Using an framework for change management

The specification of a system is not static and may evolve in order to answer to new user requirements or new constraints of the technical environment. The designer of the system takes into account these changes in the concerned views by adding, removing and/or modifying elements in the view.

In a RM-ODP vision, the system is described in different but dependent views; any action on a view may involve the same action or other actions in one or more other views. If the views are not explicitly linked, the designer does not have any means of knowing the elements in the other views which have a relationship with the modified element. Thus, change management is based only on his or her expertise and knowledge of the system.

Our modelling framework can facilitate the change management in a multi-view system thanks to the introduction of the link viewpoint. Contrary to the already existing UML profiles which allow to design views, according to viewpoints, that seem completely independent between them, we propose to extend ODP system descriptions, by adding a link viewpoint which permits to the designer to connect the views two by two. Moreover, our modelling framework guides the designer in his or her various activities. These activities fall in the three following categories.

1. **Description** of ODP system views.
The designer does not follow a particular method to build his multi-view system. The system may be even described by a team of designers; each designer is responsible for the building of a view by modelling it according to the adequate UML profile. According to the considered view, the modelling framework prevents the designer to use other concepts not relevant for this view.

2. **Establishment of correspondence link** between the ODP system views. Once the modelling of the view completed, the obtained views are completely independent. The designer binds the views by using UML profile representing the link viewpoint. The correspondence relationships are thus established between elements in two different views. In our case, enterprise view with computational view, and computational view with engineering view. These links save and keep traces of the correspondence which may exist between the elements. As the correspondence links are constrained by rules, the modelling framework checks that these links verify the correspondence rules stated between the concerned views.

3. **Change management** in the various views. The designer may perform changes in the views by modifying, for example, one of their elements. The modelling framework indicates the elements which have a correspondence link with this element. Any action on an element generates an event which is delivered to a change manager. The event contains information about the element which emitted it (its identity and in which view it is). The change manager seeks whether there is correspondence links, in the link view, which connect this element. Once the correspondence links found, it is possible to know the other elements which are connected to the modified element. The change management can take one of the following forms:

a. **Manual.** The change manager retrieves only the correspondence links referring to the modified element. It informs the designer by a message that points out the modified element, the action carried out and the elements having to be ckecked; the designer will act according to his or her knowledge of the system on the other views.

b. **Semi-automatic**. The change manager can associate to the change a set of modification strategies which act in the other views in order to modify them. If a modification was not planned, the change manager informs the designer. A new strategy concerning this case could then be added

later to automate its management. If, for example, an action of deletion was performed in an element of the view, the associated strategy may specify that the elements having a correspondence link with this element must be removed from the other views, if there is not other correspondence links with other elements belonging to the modified view.

c. **Automatic.** The change manager is autonomous and can act in the other views whatever the modification performed. This kind of manager is able to build new scenarios which were not planned at the time of its design.

6. Conclusion and perspectives

First of all, we have reminded the principles of the RM-ODP viewpoints and the various past and in progress projects and standardization efforts around it. Then, we presented our modelling framework that integrates, in particular, the management of correspondence links between various views of an application through the introduction of the link viewpoint. We showed its interest for the various activities of the designers of a multi-view system, in particular the activities relative to its evolution.

Currently, we have implemented the enterprise, computational, engineering and link meta-models. The first three meta-models were not presented in this article due to lack of space. This implementation is in the form of a "plug-in" in Eclipse [3].

In addition, we proposed a framework for change management in only one view. It has been validated by a prototype developed in the form of a "plug-in" [15] in MagicDraw [10]. A use case resulting from the context of the intelligent network for electric systems enabled us in parallel to better apprehend the requirements in term of business evolution.

The concept of correspondence link goes beyond the scope of ODP systems and may be generalized for any system which can be modelled according to several viewpoints such as, for example, PIM (Platform Independent Model) and PSM (Platform Specific Model) models with regards to the MDA (Model Driven Architecture) approach.

REFERENCES

[1] S.Bouzitouna, M.Elias, P.Spathis, M-P.Gervais et K. Thai, Création de services actifs dans ANTS, 4ème Colloque francophone sur la gestion de réseau et de service (GRES'01), Maroc. 2001

[2] A.Picault, P.Bedu, J.Le Delliou, J.Perrin, B Traverson. Specifying Information System Architectures with DASIBAO - A Standard Based Method. Proceedings of the 6th International Conference on Enterprise Information Systems. ICEIS 2004.

[3] http://www.eclipse.org/

[4] OMG. UML Profile for Enterprise Distributed Object Computing (EDOC). OMG Document. 2004. http://www.omg.org/technology/documents/formal/edoc.htm

[5] M-P. Gervais and F. Muscutariu. A UML Profile for MASIF Compliant Mobile Agent Platform. OMG's Second Workshop on UML for Enterprise Applications: Model Driven Solutions for the Enterprise USA. 2001.

[6] ISO. Information Technology – Open Distributed Processing – Use of UML for ODP system specifications. Committee Draft. 2005.

[7] ISO/IEC 10746-2.Information technology - Open Distributed Processing – Reference Model: Foundations. 1996.

[8] ISO/IEC 10746-3.Information technology - Open Distributed Processing – Reference Model: Architecture. 1996.

[9] ISO/IEC 10746-1.Information technology - Open Distributed Processing – Reference Model: Overview. 1998

[10] http://www.magicdraw.com/

[11] OMG. UML 2 OCL Final Adopted Specification. http://www.omg.org/cgi-bin/doc?ptc/2003-10-14.pdf

[12] M-P.Gervais, ODAC: An Agent-Oriented Methodology Based on ODP, Journal of Autonomous Agents and Multi-Agent Systems, Kluwer Academic Publishers, Vol. 7, n¬°3, pp199-228, 2003

[13] J-R.Putman. Architecting with RM-ODP. Prentice Hall PTR, Published October 2000.

[14] OMG. Unified Modeling Language Specification. http://www.omg.org/docs/formal/03-03-01.pdf

[15] N.Yahiaoui. Prototype using evolution contracts for reconfiguration. Research report EDF R&D. 2005

An RM-ODP Based Ontology and a CAD Tool for Modeling Hierarchical Systems in Enterprise Architecture

Lam-Son Lê, Alain Wegmann
Ecole Polytechnique Fédérale de Lausanne (EPFL)
School of Computer and Communication Sciences
CH-1015 Lausanne, Switzerland
{LamSon.Le, Alain.Wegmann}@epfl.ch

Abstract

Enterprise Architecture (EA) requires modeling enterprises across multiple levels (from markets down to IT systems) – i.e. modeling hierarchical systems. Our goal is to build a Computer Aided Design (CAD) tool for EA. To be able to build this CAD tool, we need an ontology that can be used to describe hierarchical systems. The Reference Model of Open Distributed Processing (RM-ODP) was originally defined for describing IT systems and their environment. RM-ODP can also be suited for general, hierarchical, system modeling and, hence, can be used to model enterprises. In this paper, we first give an overview of our CAD tool and we present then how Part 2 and Part3 of RM-ODP were integrated to define a computer-interpretable ontology that is used in the CAD tool. This ontology is formalized using the Alloy declarative language. Last, we illustrate how the CAD tool can render Unified Modeling Language (UML) diagrams by showing selected aspects of the hierarchical systems.

Keywords: RM-ODP, Enterprise Architecture, Ontology, Formalization, CAD tool.

1. Introduction

The goal of enterprise architecture [1] is to align the business systems and the IT systems in order to improve an enterprise's competitiveness. Enterprise architecture (EA) deals with systems perceived as hierarchical. They typically span from business entities (market, company, department…) down to IT entities (e.g. applications, applets, servlets, J2EE beans, COM…). Our goal is the development of an EA design method called SEAM (Systemic Enterprise Architecture Methodology) [1] and of the corresponding tools. SEAM is based on Reference Model of Open Distributed Processing (RM-ODP) [2] and is using a UML-like notation. In this paper, we present how the SEAM modeling ontology was developed by combining RM-ODP Part 2 (i.e. Foundations) and Part 3 (i.e. Architecture) and by formalizing the result of this combination in Alloy [3]. The originality of this paper is in the application of RM-ODP in the context of EA, in the combination of Part 2 and Part 3 and in the realization of a CAD tool. The snapshots used in the paper are made with our tool that implements the SEAM ontology. At the end of the paper, we present how UML-like diagrams can be generated from models developed with this ontology.

The "Reference Model - Open Distributed Processing" (RM-ODP) [2] is an ISO/ITU standard approved in 1996. It provides the definitions and relations between concepts useful to describe object-oriented distributed systems. It positions itself as a "meta-standard" for object-oriented modeling standards. The Object Management Group community adopted in 1998 this standard as a base for describing CORBA systems. RM-ODP has four parts: Part 1 is non-normative and introduces the standard. Part 2 defines the foundations. Part 3 describes the viewpoints necessary to design an IT system. Part 4 is a formalization of the RM-ODP concepts. RM-ODP defines the concepts in a narrative way.

To be able to build a Computer Aided Design (CAD)[1] tool based on RM-ODP particularly for modeling hierarchical systems in EA, we need to have a formal description of the terms defined in RM-ODP. This work was done for RM-ODP Part 2, using the specification language Alloy [3] and the results were published in [4] and [5]. However, even if the terms in Part 2 are formalized in Alloy, this is not sufficient to build a CAD tool. In EA, the concept of hierarchical system modeling is crucial (hierarchical models allow making simpler representations of complicated systems). To be able to build a CAD tool for EA, it is important to specify how the terms defined in Part 2 can be used to represent hierarchical systems. This is why we need to combine

[1] We call a visual modeling tool for hierarchical systems in EA a CAD tool because its scope is mainly towards marketing and business process modeling rather than software modeling (even if software modeling is possible).

them with Part 3 that defines the modeling viewpoints. Once the terms in Part 2 and Part 3 have been combined and formalized, it is possible to build a CAD tool based on this formalization. Then, the tool can generate UML-like diagrams by filtering the elements to be shown.

Section 2 discusses the integration of RM-ODP Part 2 and Part 3. It illustrates our tool capabilities. Section 3 gives the formalization of our ontology in Alloy. Section 4 presents how UML-like diagrams can be generated from the model defined using our ontology. Section 5 outlines the related work.

2. Model Elements Defined by Integrating Viewpoints and Modeling Concepts of RM-ODP

In this section, we explain which viewpoints are necessary to make EA models (Section 2.1) and which model elements are defined in the different viewpoints (Section 2.2). Section 2.1 is based on RM-ODP Part 3 (i.e. Architecture) and Section 2.2 is based on RM-ODP Part 2 (i.e. Foundations).

2.1. Viewpoints and Model Elements in EA

EA has requirements that are different from regular IT system analysis and design. In EA, the development teams deal with hierarchical systems that spans from business down to IT. For example, in an EA project implementing a new way to manage price updates in a nation-wide department store required the development of an enterprise with ten hierarchical levels (from market down to Java programming classes). In a regular IT system specification, fewer levels are considered and are usually not represented in a systematic way. In EA, the teams work with an enterprise model that represents all relevant aspects of the company. Our goal is to define a way to build such hierarchical enterprise models that represent all levels in a systematic way. Thanks to this, the different specialists, who are in charge of the different levels, can share a common way to reason about the enterprise model (because all levels represent systems) while having level-specific heuristics to guide their design decision (an organization with people is essentially not designed in a same way as a software is).

Our goal is model “general systems” (regardless of the fact that they are IT related or business related). The main characteristic of the systems is that they can be considered as wholes or as composite. Systems represented as whole exhibit emergent properties. Systems represented as composite expose their construction. In RM-ODP Part 3, the information viewpoints are defined to describe systems considered as atomic (or as wholes); computational viewpoints are defined to describe non-atomic systems (or composite systems). In an information viewpoint, there is an information specification describing the system properties in terms of information objects. In a computational viewpoint, there is a computational specification describing the system’s construction in terms of component computational objects. So, in SEAM, we consider the computational specification and the information specification as very important descriptions of the systems.

As we model hierarchical system, we believe that it is important to make explicit which system is described by these specifications, so we always consider that we have information specifications of computational objects and computational specifications of computational objects. Let’s illustrate this by an example. We model a group of companies collaborating to serve a customer. We call this group of companies a supplier value network. The supplier value network can be considered as a computational object. The information specification of this computational object describes the customer perception of this supplier value network. Figure 1 is a snapshot of our CAD tool [6] that shows the information specification of the supplier value network considered as a computational object. “SellTxn” and “Sell” are the main information object and the main localized action of the supplier, respectively.

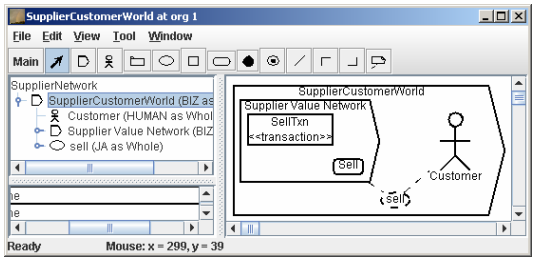

Figure 1. Information specification of the computational object “supplier value network”

The computational specification of this computational object describes which company collaborates to serve the customer is illustrated in Figure 2. It is a snapshot showing 3 companies collaborating within the supplier. Note that, in the computational specification of the supplier, each company is described in terms of its information specification.

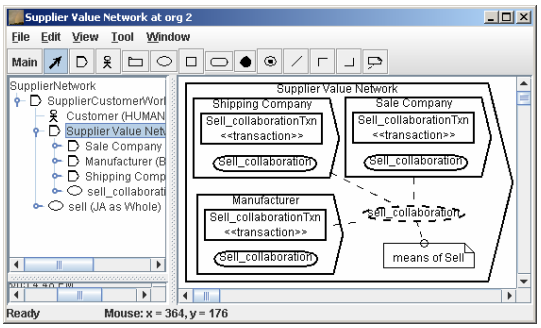

Figure 2. Computational specification of the computational object "supplier value network"

In summary our EA models are composed of a series of levels, called "organizational levels", in which computational objects are represented. These computational objects can be either specified by their information specifications or by their computational specifications.

RM-ODP defines five viewpoints to represent systems; enterprise, information, computational, engineering and technology viewpoints. We have discussed our use of the information and computational viewpoints which are central in SEAM. We analyze now the other viewpoints.

In SEAM we do not use the enterprise viewpoints. When designing an IT system, it is important to represent the environment in which the IT system exists. In SEAM, this is done through modeling all the organizational levels which are above the IT system: e.g. department level, company level or market level. In an enterprise viewpoint, all these levels are "collapsed" in the enterprise viewpoint. In SEAM, we keep all these levels separate. Note that some of the concepts defined in the enterprise language are still used in our modeling approach. The only real difference with the enterprise language is that we keep the organizational levels separate [7].

The engineering viewpoints and the technology viewpoints can be found in each organizational level. They are used to define the behavior templates that will have to be implemented in each level. For example, in an EA model, in the organizational level that represent people, the engineering viewpoint and the technology viewpoint are used to generate job descriptions. In the organizational level that describes Java programs, the engineering and technology viewpoints are used to generate the Java source code. The detailed discussion of the engineering viewpoints and of the technology viewpoints are outside the scope of this paper. Related information can be found in [8].

In RM-ODP Part 3, it is stated that each viewpoint is composed of model elements that are defined in RM-ODP Part 2.

An information specification (of a computational object) is defined as made of information objects. In SEAM we consider that these information objects are modified by actions that represent the computational object's behavior. In SEAM, we call these actions localized actions – a term borrowed from Catalysis [9]. For instance, in the example of a supplier value network in Figure 1, the supplier value network information specification represents the products and the customer information as information objects and the exchanges with the customer as a localized action.

A computational specification (of a computational object) is defined as made of computational objects that participate to actions. In SEAM, we call these actions, joint actions; a term also borrowed from Catalysis. For example, in the supplier value network, the computational specification represents the companies (i.e. computational objects described by their information specifications) participating to joint actions.

Information specifications can be described at different levels of details. This is useful to represent the goal of the systems (typically represented as localized actions or joint actions seen as a whole) or the way the goal is achieved (typically represented as localized actions or joint actions seen as a composite). For instance, the supplier value network has a "Sell" localized action in Figure 2 that represents, as a whole, its capability to sell books. This action can be broken down into "Get Order", "Deliver Book" and "Get Payment" as shown in Figure 3. This composite localized action represents a means to achieve the goal. In SEAM, we call "functional levels" this capability to describe information specifications at different levels of details.

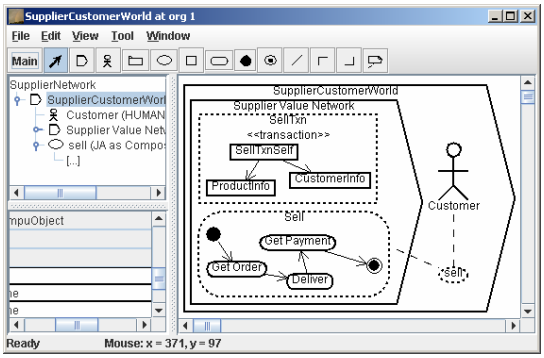

Figure 3. Information specification of computational object "supplier value network" seen at a lower functional level than that in Figure 2

In summary, to define an EA model, it is necessary to have organizational levels that describe the construction of the hierarchical systems (e.g. value networks made of companies, companies made of departments, departments made of IT systems and people,

IT system made of Java programs…). In each organizational level, we have computational objects that represent systems and joint actions between them. These computational objects can be described with their information specifications (defined as localized actions and information specifications) or computational specifications (defined as computational objects and joint actions). The information specification of a computational object describes the object as a black-box. Its computational specification describes its construction. The difference between both is substantial as both descriptions correspond to different levels of reality. For example, the computational specification of an enterprise might include departments and the concept of department would likely not appear at all in its information specification. So, the difference between both is not a simple behavioral refinement. The information specifications can be defined at different functional levels. The definition of these concepts is published in [10]. A concrete and real application of our approach is detailed in [11]. The problem now consists in understanding how these terms can be defined from RM-ODP Part 2.

2.2. Definition of Model Elements

According to RM-ODP Part 2, an entity is any concrete or abstract thing of interest in the universe of discourse. An entity is represented in the model as a model element. Each model element can be characterized by a basic modeling concept (BMC) and by a specification concept (SC) [12]. Examples of basic modeling elements are: object, action, state. Examples of specification concepts are: composite object, type, and instance.

The main concept is the computational object. A computational object is a kind of object as defined in RM-ODP Part 2. As stated in Part 2, an object has states and actions.

- The state of the computational object is described by information objects that have states. The behavior is described by actions that modify information objects. We call this kind of action a localized action. The localized actions are defined in terms of pre-conditions and post-conditions that change the state of the information objects (the dynamic schema defined in Part 3). Additional specification can be captured such as static schemas and invariant schemas that represent the states of the information objects at a given time or at all time. This description corresponds to the information specification of a computational object.
- The computational objects interact with other computational objects. So, we have another kind of actions that represent the collaboration of a computational object with the other computational objects in its environment. We call this action a joint action. The joint actions change the state of the information objects of the computational objects participating in the collaboration.

In summary, to build an EA model, we have the following basic modeling concepts: computational objects, information objects, localized actions, joint actions and states. RM-ODP defines all the concepts even if it does not name them explicitly (for example, the joint action is not defined but actions can involve more than one object). Our contribution to RM-ODP was to name some of these already existing concepts explicitly.

As we should be able to describe different functional levels, all information objects, joint actions and localized actions can be represented as whole or as composite. Of course, to be able to represent organizational levels, the computational objects can be represented as whole or as composite as well. This means that we need to have computational objects as whole or as composite, information objects as wholes or as composite, joint actions as wholes or as composite, localized actions as whole or as composite.

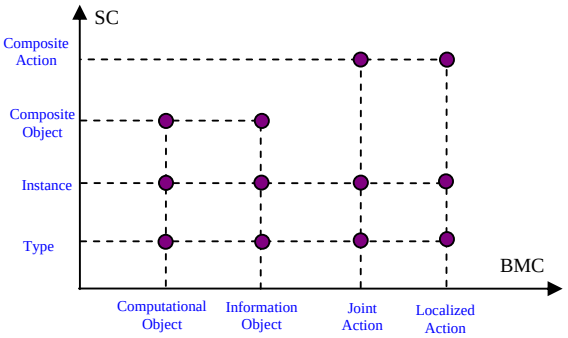

Figure 4. The 16 modeling elements can be defined by combining necessary BMCs and SCs

RM-ODP defines model elements by combining a basic modeling concept (such as object or action) and a specification concept (such as type or instance). To generate our ontology, instead of combining generic objects and actions with type and instance, we combine computational objects, information objects, localized action and joint actions with type and instance (as illustrated in Figure 4). In addition, we need to add if the model element is considered as a whole or as a composite. In summary, our model elements are defined by the following statements:

- <x> type as a whole
- <x> type as composite

- <x> instance as a whole
- <x> instance as composite

Where <x> is computational object, information object, localized action or joint action.

3. Formalization

Although we consider that <x> seen as whole and <x> seen as composite are two separate model elements, we notice that these two model elements always go together. In fact, they represent a single entity in reality. In the EA model, depending on the context, <x> seen as whole or <x> seen as composite is used. To simplify the formalization, when we define <x>, we always consider both aspects (whole and composite).

The formalization is done in Alloy 2.0 [3]. Alloy is a lightweight modeling language based on relations and takes the syntax of the first-order logic. Since version 2.0, the language also has object-oriented constructs. The language is accompanied by a tool called Alloy Constraint Analyzer (ACA) that can be used for simulating and checking models written in Alloy.

The entire Alloy code is given in the appendix at the end of this paper. Note that the Alloy code formalizes the types (<x> type), so all terms in the Alloy could be prefixed with "Type". For clarity reasons, we decided not to do so.

Table 1 shows signature `CompuObject` (`sig` is a keyword in Alloy) that formalizes the computational object. Each computational object optionally mediates a joint action (field `ja`), refers to an information object (field `info`) and a localized action (field `la`), has a number of child computational objects (field `compu_children`) and optionally has a parent computational object (field `compu_parent`). We can interpret that the field `ja` and the field `compu_children` correspond to a computational object seen as composite, whereas the field `compu_parent`, the field `info` and the field `la` stands for that computational object but seen as whole. In other words, the computational object as whole and as composite are formalized in one Alloy signature: `CompuObject`.

Table 1. Alloy code for the computational object

```
sig CompuObject // definition of the computational object
{  // declaration of all fields of the computational object
   ja : option JointAction,
   info : option InfoObject,
   la : option LocalizedAction,
   compu_children : set CompuObject,
   compu_parent : option CompuObject
} { // invariants concerning the computational object
   #compu_parent>0 => this in compu_parent::compu_children
   all c : compu_children | c::compu_parent = this
   #ja > 0 => ja::compu_med = this
   all j : JointAction - ja | j::compu_med = this =>
           j in ja.^joint_children
   #info > 0 => info::compu_host = this
   all io : InfoObject - info | io::compu_host = this =>
          io in info.^info_children
   #la > 0 => la::compu_owner = this
   all l : LocalizedAction - la | l::compu_owner = this =>
          l in la.^localized_children
}
```

The second block between curly brackets, right after the declaration of all fields of signature `CompuObject`, lists all invariants that must be held for any computational objects. The first line of this block assures that any computational object must be a child computational object of its parent. The second line implies that all child computational objects have the same parent. The rest of this block concerns the way each computational object relates to information objects, joint actions and localized actions. In short, these invariants mandate the way model elements are related to one another in our ontology.

In EA models, having individual model elements is not enough. Indeed, we need to put them in relation. First, each model element has child elements which are of the same kind as the parent element. Second, to express static schema in the information specification, information objects need to be related to other information objects by associations. Third, to express the activity of each computational object, its localized actions need to be related by action transitions. Fourth, to correctly show the interaction between computational objects, links between computational objects and joint actions they participate in must be kept. As a summary, the following relations need to be addressed in the ontology

- parent-child relation
- association
- action transition
- collaboration link

One way to formalize these relations is to declare additional fields in the signatures that formalize model elements. This first approach applies to the child-parent relation. For example, in signature `CompuObject` we can put a field such as `compu_children : set CompuObject` that results in all child computational objects of the declared computational object. Another approach is to declare a particular signature dedicated for each kind of relation. This signature must have at least two fields referencing the source and the destination model element of the declared relation. In this approach, it is easy to put additional attributes such as role name, cardinality… to declared relations. The second approach applies to association, action transition and collaboration link. Their corresponding signatures in Alloy code are named `IO2IO`, `LA2LA`, `CO2JA` and `JA2CO` respectively. This naming convention is put in place to emphasize which kinds of model elements are connected by the formalized relations.

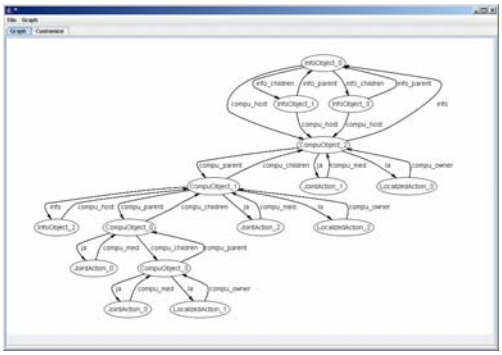

Figure 5. A simulation of the formalization written in Alloy

The entire formalization code is simulated using ACA at the size of 4 (i.e. at most 4 instances shall be created for each signature). Figure 5 shows one possible solution. The signature instances are represented as eclipses. For each instance, the fields are drawn as out-going arrows. The eclipse that an arrow points to represents the value of the field represented by that arrow. Note that to fully formalize the ontology, we also need to declare attributes in the signatures of model elements such as pre/post condition (of the localized action and the joint action), role name or cardinality (of the association and the collaboration link). These declarations will lead to defining special Alloy signatures to represent primitive data types (e.g. Integer, String…) and will make the simulation unnecessarily complicated. For this reason, we omit all primitive attributes and only declare the ones that represent child-parent relation.

Table 2 list some correspondences between the model of the supplier value network shown in Figure 1, 2, 3 and the Alloy solution shown in Figure 5.

Table 2. Comparison between a model done in our CAD tool and the solution found by ACA

Model Element	Model in CAD tool	Solution by ACA
Computational Object	Supplier Customer World	CompuObject_0
Computational Object	Supplier Value Network	CompuObject_1
Computational Object	Customer	CompuObject_3
Computational Object	Sale Company	CompuObject_2
Joint Action	Sell	JointAction_0
Information Object	SellTxn	InfoObject_2
Localized Action	Sell	LocalizedAction_2
Joint Action	sell_collaboration	JointAction_2

The design of the CAD tool follows Model-View-Controller approach. Each modeling concept declared in Alloy corresponds to a design class in the part "Model" and is rendered by, nevertheless, two classes in the part "View": one class for <x> as whole and the other for <x> as composite.

4. Example of UML Representation

The information specification of the Supplier Value Network can be filtered to show only the class diagram (Figure 6 a) or the activity diagram (Figure 6 b). The modeler can easily do so while selecting the pictogram of the Supplier Value Network in the diagram illustrated in Figure 3.

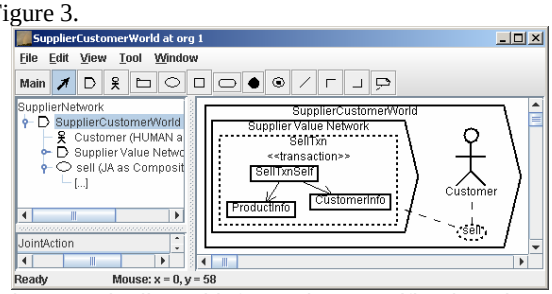

Figure 6 a) Filtered information specification shows a class diagram of "supplier value network" in its context.

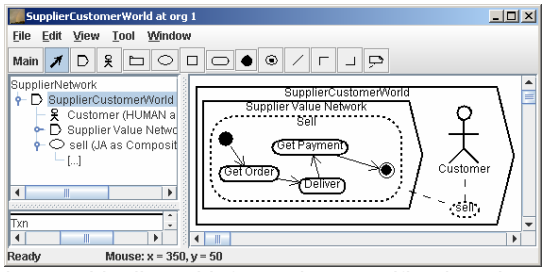

Figure 6 b) Filtered information specification shows an activity diagram of "supplier value network" in its context.

The value of this representation lies in the fact that the modeler can ultimately reason about each system by using individual UML diagrams. Future versions of our tool will allow her to edit these diagrams in separate windows, making it more UML-compatible.

5. Related Work

RM-ODP based modeling has increasingly attracted researchers in the field of EA and formal methods. [13] proposes a tool environment for viewpoint-oriented EA. This framework allows the modeler to define various viewpoints (that are not necessarily limited to RM-ODP viewpoints) of an EA project. [14] aims at relating RM-ODP enterprise viewpoint and computational viewpoint

to make consistency checking feasible. [15] explores the possibility of formally writing computational viewpoint specifications in Maude, an object-oriented executable rewriting logic language. [16] proposes a declarative language (with specialized constructs for security) to specify enterprise viewpoints. [17] describes a formal interpretation to viewpoint consistency and proposes some strategies for checking viewpoint consistency in general. [18] focuses on the consistency between the computational viewpoint and the engineering viewpoint. Our work addresses hierarchical systems in EA and tries to integrate the information viewpoint and the computational viewpoint of RM-ODP to make a computer-interpretable tool-supported ontology with quite precise semantics of model elements. We use Alloy, a lightweight modeling language based on set theory with object-oriented constructs, to formalize our ontology.

One of the big concerns about RM-ODP viewpoints is the transition between them. [19] defines information actions and proposes the articulation between them and messages in computational view. [20] establishes one-to-one mapping from an information action to a stateless object. In our ontology, information objects representing the transactions are similar to stateless objects [20]. We believe that, for a certain computational object, its composite view is quite dependent from its whole view. The decomposition made on a composite computational object is just a choice of the modeler, although it can be inspired by the whole view. Our tool manages the connection between the two views. However, it leaves the decomposition to the modeler as her own decision.

Conclusions

In this paper, we an ontology that is suited to modeling hierarchical systems in EA. We base our work on RM-ODP. To define the necessary model elements and their relations in the ontology, we integrate the computational viewpoint and the information viewpoint of RM-ODP. The resulting ontology is formalized in Alloy and simulated using Alloy Constraint Analyzer. The computer-interpretable version of the ontology is implemented in a CAD tool. This tool allows the modeler to build models for hierarchical systems where every model element can be seen either as whole or as composite. This tool can also render UML diagrams according to some filtering option to reflect particular aspects of the system of interest.

Our future work investigates the possibility to translate the formalization written in Alloy to partial Java code (mainly in the “Model” part of the entire Model-View-Controller design of the tool) that implements our CAD tool so that we can achieve some automation between the ontology specification and the ontology implementation. We also consider developing advanced features that allows the modeler to separately edit UML diagrams of the system of interest.

References

[1] A. Wegmann, "On the Systemic Enterprise Architecture Methodology (SEAM)", in the proceedings of ICEIS 2003, Angers, France, 2003.

[2] OMG, "ISO/IEC 10746-1, 2, 3, 4 | ITU-T Recommendation, X.901, X.902, X.903, X.904, Reference Model of Open Distributed Processing," 1995-1996.

[3] MIT, The Alloy Constraint Analyzer, http://alloy.mit.edu/

[4] A. Naumenko, Wegmann, A., Genilloud, G., Frank, W.F., "Proposal for a formal foundation of RM-ODP concepts", in the proceedings of ICEIS/WOODPECKER 2001, Setúbal, Portugal, 2001.

[5] A. Naumenko, Wegmann, A., "A Metamodel for the Unified Modeling Language", in the proceedings of <<UML>> 2002, Dresden, Germany, 2002.

[6] L. S. Lê, Wegmann, A., "SeamCAD 1.x: User's Guide," School of Computer and Communication Sciences, EPFL, Lausanne IC/2004/98, November 2004 2004.

[7] S. Tyndale-Biscoe, "RM-ODP enterprise language ITU-T Rec. X.911: ISO/IEC 15414", 2005

[8] A. Wegmann, Preiss, O., "MDA in Enterprise Architecture? The Living SystemTherory to the Rescue…", in the proceedings of 7th IEEE EDOC, Brisbane, Australia, 2003.

[9] D. Francis D'souza, Cameron Wills, A., *Object, Components and Frameworks with UML, The Catalysis Approach*: Addison-Wesley, 1999.

[10] L.-S. Le, A. Wegmann, "Definition of an Object-Oriented Modeling Language for Enterprise Architecture", in the proceedings of 38th HICSS, Hawaii, USA, 2005

[11] A. Wegmann, G. Regev, B. Loison, " Business and IT Alignment with SEAM" in the proceedings of the REBNITA Workshop, RE05, Paris, 2005

[12] A. Wegmann, Naumenko, A., "Conceptual Modeling of Complex Systems using an RM-ODP based Ontology", in the proceedings of 5th IEEE EDOC, Seattle, USA, 2001.

[13] M. W. A. Steen, Akehurst, D.H., ter Doest, H.W.L., Lankhorst, M.M., "Supporting Viewpoint-Oriented Enterprise Architecture", in the proceedings of 8th IEEE EDOC, California, USA, 2004.

[14] R. Dijkman, Quartel, D., Pires, L., Sinderen, M., "A Rigorous Approach to Relate Enterprise and Computational Viewpoints", in the proceedings of 8th IEEE EDOC, California, USA, 2004.

[15] R. Romero, Vallecillo, A., "Formalizing ODP Computational Viewpoint Specifications in Maude", in the proceedings of 8th IEEE EDOC, California, USA, 2004.

[16] E. Lupu, Sloman, M., Dulay, N., Damianou, N., " Ponder: realising enterprise viewpoint concepts", in the proceedings of 4th IEEE EDOC, Makuhari, Japan, 2000.

[17] H. Bowman, Steen, M., Boiten, E., Derrick, J., "A Formal Framework for Viewpoint Consistency", *Formal Methods in System Design*, 2002.

[18] E. Boiten, Bowman, H., Derrick, J., Linington, P., Steen, M., "Viewpoint consistency in ODP", *ELSEVIER*, 2000.

[19] J. Dustzadeh, Najm, J., "Consistent Semantics for ODP Information and Computational Models", in the proceedings of FORTE/PSTV'97, Osaka, Japan, 1997.

[20] C. Bernardeschi, Dustzadeh, J., Fantechi, A., Najm, J., Nimour, A., Olsen, F., "Transformation and Consistent Semantics for ODP Viewpoints", in the proceedings of FMOODS'97, Canterbery, UK, 1997.

Appendix: Ontology Formalization in Alloy

```
module seamcad

sig CompuObject{
   ja : option JointAction,
   info : option InfoObject,
   la : option LocalizedAction,
   compu_children : set CompuObject,
   compu_parent : option CompuObject
} {
   #compu_parent > 0 => this in compu_parent::compu_children
   all c : compu_children | c::compu_parent = this
   #ja > 0 => ja::compu_med = this
   all j : JointAction - ja | j::compu_med = this => j in ja.^joint_children
   #info > 0 => info::compu_host = this
   all io : InfoObject - info | io::compu_host = this => io in info.^info_children
   #la > 0 => la::compu_owner = this
   all l : LocalizedAction - la | l::compu_owner = this => l in la.^localized_children }

fact compu_acyclic {
   all c : CompuObject | (c !in c.^compu_parent && c !in c.^compu_children) }

sig InfoObject{
   compu_host : CompuObject,
   info_children : set InfoObject,
   info_parent : option InfoObject
} {
   all c : info_children | c::info_parent = this && c::compu_host = this::compu_host
   #info_parent > 0 => this in info_parent::info_children }
fact info_acyclic {
   all c : InfoObject | (c !in c.^info_parent && c !in c.^info_children) }

sig JointAction {
   compu_med : CompuObject,
   joint_children : set JointAction,
   joint_parent : option JointAction
} {
   all c : joint_children | c::joint_parent = this && c::compu_med = this::compu_med
   #joint_parent > 0 => this in joint_parent::joint_children }

fact joint_acyclic {
   all c : JointAction | (c !in c.joint_parent && c !in c.^joint_children) }

sig LocalizedAction {
   compu_owner : CompuObject,
   localized_children : set LocalizedAction,
   localized_parent : option LocalizedAction
} {
   all c : localized_children | c::localized_parent = this && c::compu_owner = this::compu_owner
   #localized_parent > 0 => this in localized_parent::localized_children }

fact localized_acyclic {
   all c : LocalizedAction | (c !in c.^localized_parent && c !in c.^localized_children) }

sig IO2IO { // association
   source : InfoObject,
   destination : InfoObject
} {
   source::compu_host = destination::compu_host }

sig LA2LA { // action transition
   source : LocalizedAction,
   destination : LocalizedAction
} {
   source::localized_parent = destination::localized_parent
   source::compu_owner = destination::compu_owner }

sig CO2JA { // collaboration link
   source : CompuObject,
   destination : JointAction
} {
   source::compu_parent = destination::compu_med }

sig JA2CO { // collaboration link
   source : JointAction,
   destination : CompuObject
} {
   source::compu_med = destination::compu_parent }

sig IO2CO { // trace dependency
   source : InfoObject,
   destination : CompuObject }

fact oneRoot {
   sole co : CompuObject | #co.compu_parent = 0 }

fun hasLifecycle() {
   some co : CompuObject | one co.ja && no co.la && no co.info && #co.compu_children = 2 }

run hasLifecycle for 4
```

On Interactions in the RM-ODP

Guy Genilloud, Gonzalo Génova
Departamento de Informática, Universidad Carlos III de Madrid,
avda. Universidad, 30 – 28911 Leganés, Madrid, Spain
guy.genilloud@ie.inf.uc3m.es, ggenova@inf.uc3m.es

Abstract

The RM-ODP definition of *interaction* is subtle and difficult to understand. On closer inspection, we find that it is in fact invalid. We then attempt to establish the author's original intentions for this concept, and to provide an effective definition. Our investigations lead us to discover some fundamental flaws in the definitions of *interface* and of *behaviour of an object*.

1. Introduction

ISO has performed excellent work with the RM-ODP Foundations [1]. Unfortunately, this work has failed to influence the most popular object-oriented modelling techniques and languages (UML being a case in point), and methodologies. Their creators have either ignored the RM-ODP, or failed to understand it. To their credit, some definitions are excessively subtle or obscure, and the RM-ODP lacks explanations, and references to relevant literature.

In this paper, we analyse one of these subtle definitions: that of the concept of *interaction*. This definition is dependent on that of *environment (of an object)*. While both definitions may seem crisp and clear on a first read, they turn out to be meaningless since based on an apparent circularity. We then investigate whether new definitions, making use of the concept of role, would work. Having hopefully understood, through our analysis, the intentions of the RM-ODP authors, we provide an explanation for the concept of interaction. This analysis allows us to propose alternate definitions. Finally, armed with our new understanding that interactions have roles, we find fundamental flaws in two RM-ODP definitions, those of *interface* and *behaviour (of an object)*.

2. The Environment of an Object

The RM-ODP foundations provide this definition:

> ***8.2 Environment (of an object)****: the part of the model which is not part of that object.*
> *NOTE - In many specification languages, the environment can be considered to include at least one object which is able to participate without constraint in all possible interactions (see 8.3), representing the process of observation. [1]*

While its note is unlikely to be helpful to most readers, this definition seems at first to be straightforward. But now, consider the 4th note in the very next definition in the standard, that of action:

> ***8.3 Action****: Something which happens.*
> *Every action of interest for modelling purposes is associated with at least one object.*
> *The set of actions associated with an object is partitioned into internal actions and interactions. An* ***internal action*** *always takes place without the participation of the environment of the object. An* ***interaction*** *takes place with the participation of the environment of the object.*
>
> *NOTES*
>
> *1 - "Action" means "action occurrence". Depending on context, a specification may express that an action has occurred, is occurring or may occur.*
>
> *2 - The granularity of actions is a design choice. An action need not be instantaneous. Actions may overlap in time.*
>
> *3 - Interactions may be labelled in terms of cause and effect relationships between the participating objects. The concepts that support this are discussed in 13.3*
>
> *4 - An object may interact with itself, in which case it is considered to play at least two roles in the interaction, and may be considered, in this context, as being a part of its own environment.*
>
> *5 - Involvement of the environment represents observability. Thus, interactions are observable whereas internal actions are not observable, because of object encapsulation. [1]*

We have 2 contradictory statements: the environment of an object is "*the part of the model which is not part of that object*," and "*an object can be a part of its own environment*." We assume that it is the second statement, of minor status since in a note, which is incorrect. Peter Linington, one of the fathers of the RM-ODP, confirmed to us that the idea of an object being part of its own

environment is metaphorical (as suggested by the phrase "in this context"). Therefore, an object cannot actually be a part of its own environment.
This metaphorical note is unfortunate, not only because it can confuse readers of the standard, but also because it hides the fact that the concept of interaction is ill-defined. If taking place with the participation of the environment is not a discriminator of an interaction (vs. an internal action), then what is an interaction of an object? We will investigate this question in Section 3.
As for the notion of environment, we believe that it requires further explanations, as we discuss below.

2.1. Observability.

Studying the standard, it is clear that observability was the main concern, if not the only concern, for introducing the notion of environment ("*Involvement of the environment [in an action] represents observability.*" [1, Definition 8.3 Action, Note 5]). But why does the RM-ODP speak of observability, and not of observation?
A possible answer could be that the readers of a specification do not know whether an observer object in the environment actually does observe the action. That is, they do not have full knowledge of the environment of an object – there might be objects in the environment they know nothing about.
But what about specification techniques, such as classical UML [2], in which at most two objects may participate in an interaction? When an object interacts only with itself, no other object may observe this interaction. So why does ODP consider such an action as observable, when in fact it actually cannot be observed?

2.2. Asynchronous interactions.

In a recent paper [3], Peter F. Linington mentions another concern for the environment of an object. As he explains, the RM-ODP enables objects to interact either in a synchronous way (the objects are simultaneously in the interaction, as if they were shaking hands[1]), or in an asynchronous way (e.g., the exchange of a message consists of a sequence of synchronous interactions, initiated by the submission of the message, and terminated by its delivery). The notion of environment plays a key role in achieving this capability:

> *Interactions in ODP are synchronous[2], but are defined between an object and its environment. If we then constrain the way objects are composed so that each object binds directly to another object in its environment, the resulting communication is synchronous, but if objects bind to link ends in their environment, the resulting communication is asynchronous. [3]*

Such explanations would be interesting and very helpful to the many object-modellers who consider that objects only interact by exchanging messages (the authors of UML among them). Unfortunately, there is not even a hint about this idea in the RM-ODP Foundations. There is not even an explanation that communication links exist in a model (as binding objects), even though they are not specified explicitly (e.g., UML and ESTELLE [4]).

2.3. Are all communications observable?

The idea of asynchronous communication via (implicit) communication links suggests that all asynchronous communications are observable (if an object sends a message to itself, this composite action would be observed by communication links in its environment). However, was this the RM-ODP authors intention?. We are not so sure.
The notion of observation of actions is important with respect to behavioural compatibility[3] [1, definition 9.4]: if an object's behavioural specification is substituted for another, would some environment be able to notice the difference? However, since implicit (unspecified) communication links are unable to tell other objects any differences they may notice, should they count as observers? We think not.
The actions of submitting a message, or delivering it, would then never be (directly) observable (even though asynchronous communications may or may not be observable). Therefore, we would have some actions (e.g., sending a message) which involve participation of the environment, but which are nevertheless unobservable. Or perhaps, the definition of the environment of an object needs being revisited?
The RM-ODP definition of environment is therefore deceivingly simple, and perhaps even misleading. The following is a list of questions, which we think the standard should answer:

- Is the environment of an object the same thing as all the other objects in the model?
- Are all the other objects in a model necessarily known to a specifier?
- If an object sends a message to itself, using implicit communication links, is that necessarily an observable action?

[1] The notion of synchronicity discussed here has nothing to do with the notion of synchronicity in remote procedure calls.

[2] P. F. Linington probably meant to write that *atomic* interactions are synchronous. Otherwise, the communication of a message, a composite action, could not be considered to be an interaction…

[3] The RM-ODP should make this point clear, as soon as introduces the notion that some actions are observable.

3. Interactions

3.1. Which actions are interactions?

Intuitively, one would define an interaction as an action in which more than one object participates, and an internal action as an action in which a single object participates. However, this is not the idea that was retained for the RM-ODP. Instead, the participation of the environment was elected to be the discriminating criterion:

> *"An **internal action** always takes place without the participation of the environment of the object. An **interaction** takes place with the participation of the environment of the object." [1, Definition 8.3 Action]*.

This definition is unsatisfactory, for at least two reasons:

1) Environment is not a crisp and clear concept, as we have seen.
2) An object may interact with itself, without the participation of its environment (assuming that an object cannot be part of its own environment).

Regarding the second point, we suppose that the authors' intention was to classify some actions as interactions, independently of the actual involvement of the environment. So, for example, when a Java runtime object invokes one of its own methods, ODP would categorize this action as an interaction rather than as an internal action (because other objects may also invoke the same method). The fact that the RM-ODP considers interactions to be those actions that are observable [1, Definition 8.3 Action, Note 1], rather than those that are actually observed, comforts us in this interpretation.

Involvement of the environment is therefore not a necessary criterion for an action to be considered an interaction of an object[4].

3.2. Roles in interactions

As we have seen, the RM-ODP authors deliberately attempted to classify some unobserved actions as interactions (observable) rather than as internal actions (unobservable). However, they failed in their attempt of defining a classification.

When being presented with these contradictions, Peter Linington and Jean-Bernard Stefani, two of the most prominent authors of the RM-ODP, proposed to discriminate between interactions and internal actions by referring to the notion of role [5, 6]. However, they did not elaborate much on their respective answers. We will now examine if such a classification based on roles is indeed possible.

In fact, the 4th note of the RM-ODP definition of action already makes a reference to the concept of role: "*an object may interact with itself, in which case it is considered to play at least two roles in the interaction*").

This yields an interesting observation: an interaction has multiple *roles*. And indeed, it seems true that every interaction has multiple roles:

1) In classical object-orientation, there is only one type of interaction between objects: the exchange of a message between two objects. This is a composite interaction[5] with two roles, which we may call the sender and the receiver. An object may send a message to itself, in which case it is both the sender and the receiver of that composite interaction.
2) A rendezvous between two objects is an example of a synchronous interaction. Such an action serves the purpose of synchronising two activities, the rendezvous being a common action in both activities. There are two roles because two activities are required (otherwise, the rendezvous action would be pointless). Generally, two different objects will participate in a rendezvous, but it is also possible for a single object to play both roles in the rendezvous.

For every interaction type that we may conceive, it seems indeed possible and natural to consider that it has multiple roles. So, we may consider that an action with multiple roles is an interaction, and that an action with just one role, or with no role at all, is an internal action.

However, the question that we must answer is not, given an action, whether this action is an interaction of some object(s). Rather, the question is, given an object and one of its actions, whether this action is an interaction of that object, or one of its internal actions. Answering this second question is not straightforward, because as we will now see, an object can be a composite object.

3.3. Interactions of composite objects

In the RM-ODP, an object can be a composite object, expressed as the composition of more elementary objects.

> ***9.1 Composition:***
> ***a) (of objects)** A combination of two or more objects yielding a new object, at a different level of abstraction. The characteristics of the new object are determined by the objects being combined and by the way they are combined. The behaviour of a composite object is the corresponding composition of the behaviour of the component objects. [1]*

[4] In the RM-ODP Overview, Section 7.1.2, it is said explicitly "*Note that these interactions do not necessarily occur with other objects: an object can interact with itself.*"

[5] This interaction is composite since composed of sub-actions, among them submitting the message onto the implicit communications channel, and the message delivery.

Considering the behaviour obtained by pure composition (i.e., without applying any abstraction[6]), all the actions of the *component objects* are also actions of the composite. Obviously, any action internal to one of the components is an internal action of the composite. And any interaction of a component object with the environment of the composite is an interaction of the composite. But what about the interactions of component objects among themselves? Are they interactions of the composite with itself, or are they internal to it?

If we were to use the criterion that an interaction has roles, we would conclude that all the interactions among the components are interactions of the composite. But this conclusion would not reflect what happens in practice. For example, most systems have communications among their components that are not observable by external entities.

But note that definition 9.1 says explicitly that a composite object is defined not only by which objects compose it, but also by how they are combined. Some specification languages enable a specifier to declare, with a *hide* operator, that some interactions among the component objects will be internal actions of the composite (therefore hiding them to the environment of the composite; no actions are suppressed). A similar effect can be obtained by using devices such as interceptors that make it impossible for an external object to access some components of the composite object. In the ODP computational language [7], the behaviour of components might be such that some interface references are not communicated outside of the composite, making it impossible for external objects to have interactions at those interfaces: all interactions at such interfaces shall therefore be considered to be internal actions of the composite.

In [3], Peter Linington explains that the identification of interfaces is a design activity. That is, it is design decisions that allocate interactions to interfaces. The same idea applies with respect to deciding whether an interaction among components is an interaction of the composite, or one of its internal actions. Note that the design decision may not be left entirely to the specifier; it also depends on the specification language that he or she uses.

4. Interactions: an explanation

We have no doubts at all that the RM-ODP authors tried to define interactions and internal actions in a sensible and useful way. It is no accident that they did not define interaction in the straightforward way, that is, as an action in which two or more objects participate. But they failed to provide clear and unambiguous definitions. Even more annoyingly, we do not even know what were the authors' intentions.

What the authors should have done was to explain the definitions they wrote, and in particular the intentions behind the non-straightforward definitions. We will now try, given our analysis and understanding of the issue, to provide these missing explanations.

We suppose that the RM-ODP authors were very much concerned with the problem of writing behavioural specifications for objects. We also know that some ODP authors are not always coherent with the standard, as they entertain confusion between a term and the model element to which it refers. In particular, they incorrectly speak of "actions in a specification" (see for example [8]), which does not make sense since an action is "*something which happens.*" And of course, a specification is a static entity in which nothing may happen. An "action in a specification" may be explained as a term for an action in a model[7]. Or more precisely, it is a term for one or more actions, as a same term can be interpreted again and again, each time referring to a different action.

In [3], Peter F. Linington provides an example of this situation in the context of ODP. He observes that an object can theoretically participate in infinitely many actions. As he explains, "*most objects will have a behaviour that allows arbitrary repetition of smaller fragments of behaviour associated with some sort of thread or session.*" We would say instead that *the term for a fragment of behaviour* (itself composed of terms for actions) can be interpreted again and again to refer to different fragments of behaviour[8]. This property is essential for enabling an object to have an infinite behaviour, since a specification can only have a finite number of terms. More pragmatically, this property is also very important for writing specifications of reasonable length, as any programmer could testify.

Thus, a term of an action generally refers to many actions. Now, consider a term of an action that is related to actions in which it is *always* the case that only one object participates. ODP authors (we suppose) would consider all these actions to be *internal actions of this object*. Consider another term, which in general refers to an action in which this object plays just one role, but which may refer to an action in which this very same object plays all the roles. ODP authors (we suppose) would consider all these actions to be *interactions of this object.*

[6] We do not understand why the RM-ODP pretends that a composite object is necessarily at a more abstract level of abstraction than its component objects. Surely, composition enables abstraction, but does it always include it?

[7] As we announced in Section 1, we use the term "model" in its ODP meaning, and not in its UML meaning.

[8] Strictly speaking, Linington's sentence is incorrect, since a fragment of behaviour, being a collection of actions (occurrences), cannot be repeated.

Our explanation is compatible with [1, definition 8.3], in which it is said that "*An internal action always takes place …*" Since an action takes place only once, why did the authors of the RM-ODP wrote "always takes place"? We suppose that they were thinking about the term of the action.

4.1. What it means to be observable

In the RM-ODP foundations, the notion of observability (of actions) is important in the context of another notion, that of behavioural compatibility (between objects):

> ***9.4 Behavioural compatibility:*** *An object is behaviourally compatible with a second object with respect to a set of criteria (see notes) if the first object can replace the second object without the environment being able to notice the difference in the objects' behaviour on the basis of the set of criteria. [1]*

Since an ODP object is "*the model of an entity*" [1], it servers no purpose to consider two objects, and to wonder whether or not they are behaviourally compatible. The question of behavioural compatibility is in fact about object specifications, on whether one is compatible with the other[9]. More precisely, the problem is to know whether an object's environment (i.e., the environment of the object in some model) would be able to find out which of two possible specifications applies to this object. Nothing is known about the object's environment, besides some assumptions (derived from the set of criteria in the above definition). For the environment to observe anything about the object, it must participate in interactions with it.

Therefore, we are talking here about many actions in many models. What we really want to classify is not the actions of an object, but the terms of its actions in a specification. An action term is said to be "observable" if it refers to at least one action, in some model, which may be observed by the environment of the object. Unless an action term can be proven to always refer to unobserved actions of an object, it is to be considered an "observable" action term of the object specification[10].

Thus, our explanation is that ODP considers an action to be observable if it is referred to by an "observable" action term; otherwise it is non-observable. And the name "interaction" is a synonym for "observable action".

Thus, in a given model, some interactions of an object are observed while others are not. But they are all considered to be observable, in this particular ODP sense. This is the case even though the environment, in this given model, cannot in fact observe them. On the other hand, no internal actions of the object are ever observed, in any model, and they are therefore non-observable.

4.2. An alternative explanation

In a recent private communication, Peter Linington provided us with an interesting explanation of what it means for an object to interact with itself, and to be in such context a part of its own environment:

> *"The text about an object contributing to its own environment is in a note and is clearly not the defining occurrence. In fact the phrase "in this context" is a strong hint that the usage is metaphorical. The point being made here is that when an object interacts with itself, there is a degree of decoupling, so that the constraints on the interaction arise from the object playing two roles, and for each role the actions of the other are as if from the environment. This implies that static analysis of the behaviour of an object as a configuration component may indicates liveness that is not guaranteed if the two roles are closely coupled by the internal behaviour of the object; the situation goes from possible liveness to necessary deadlock."*

We do agree with the above explanation, but we are not sure that it is sufficient. We therefore choose to stick to our own explanations, until convinced otherwise.

5. Proposed new definitions

Having (hopefully) understood the original ODP authors' intentions, we are now in a position to propose some alternate definitions of the concepts of environment and interaction. But first of all, we would like to stress that what matters first and foremost is the message we want to communicate to the readers of the standard. It would be better to leave the RM-ODP Foundations with its current deficient definitions, but to add proper explanations, than to provide revised but obscure (or excessively subtle) definitions. And after all, the new definitions might prove be defective again, in which case the explanations would be sorely missing.

---

9 Whereas object is a basic modelling concept in the RM-ODP, behavioural compatibility is a specification concept.

10 It is in general an undecidable problem to determine which action terms of an object specification may refer to (in some interpretation) an observed action. For that reason, the principle of precaution applies: in case of doubt, an action term of an object specification is considered to be "observable." This matters, for example, when trying to determine whether an object specification is equivalent to another.

5.1. Environment, and communications

The following is our proposed revised definition for the environment of an object:

> **8.2 Environment (of an object)**: all the objects in the model which are not part of that object.
> NOTES
> 0- An object is never a part of its own environment.
> 1 - In many specification languages, the environment can be considered to include at least one object which is able to participate without constraint in all possible interactions (see 8.3), representing the process of observation.
> 2- In some specification languages, an object can be considered to include, as parts of itself, outbox and inbox objects, even though those are not explicitly specified. An internal outbox object enables a sender object to submit a communication (see 8.9) with its environment, without that action being observed directly (the communication as a whole is observed, but not its initiation). Likewise, an internal inbox object enables the delivery of a communication, without that action being observed by the environment of the receiver object.

Note 0 is provided here to stress the change in this new definition from what is said in the original RM-ODP Foundations standard.
Note 1 may need to be suppressed or amended, since it explains the process of observation in a narrow way, valid only in the case of some specification languages. But it has the merit to introduce a reference to the notions of interaction and observation, which are the very reasons behind the definition of environment. And they also help with Note 2. But whether or not this not remains, it is essential for the revised standard to explain the very reasons behind the concepts of environment and observability.
Note 2 may need to refer to complementary explanations regarding the exchange of asynchronous communications between objects. In particular that a communications channel consists of a configuration of objects, among which an outbox object (or inbox object) which is part of the object itself, and other binding objects which are parts of the object's environment. The standard should explain that inbox and outbox objects are internal to an object for reasons of evaluating whether two object specifications are behaviourally compatible or not. Since implicit communication channels are beyond the control of any specifier, it is not necessary, but preferable, to consider that submitting a communication or receiving it are internal actions of the sender and the receiver, respectively.

5.2. Interactions

We do not see a way to define interaction without referring to specifications and terms. We first need this definition:

> **Observable action term (of an object)**: An action term is **observable** in an object specification, if it refers to at least one action, in some model, which may be observed by the environment of the object. Otherwise, this action term is **non-observable**.
> NOTE: In case of doubt, an action term is observable: if an action term cannot be proven to be non-observable, it is to be considered as being observable.
>
> **Internal action (of an object)**: an action of an object which is referred to by a *non-observable action term* in the specification of that object.
> NOTE: Not a single internal action of an object is observed by its environment.
>
> **Interaction (of an object)**: an action of an object which is referred to by an *observable action term* in the specification of that object.
> NOTES:
> 1- An object may play all the roles of an interaction, in which case this action is unobserved (by this object's environment). This action is still categorized as an interaction of this object because it is referred to by an observable action term in the object's specification.
> 2- Some specification techniques are such that an action is referred to by several action terms. In this case, it is sufficient for one of these action terms to be deemed observable for the action to be an interaction.

Since observable in ODP has a non-obvious, counter-intuitive meaning, we propose to deprecate the use of the phrase "observable action".

6. Other issues in the RM-ODP

In our investigation, we have realized that interactions are characterized by having multiple roles. This fact was not taken into account when writing some of the definitions of the RM-ODP Foundations, which as a result are flawed.

6.1. Interface

The definition of interface needs revisiting:

> ***8.4 Interface**: An abstraction of the behaviour 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.
Each interaction of an object belongs to a unique interface. Thus the interfaces of an object form a partition of the interactions of that object.
NOTES
1 - An interface constitutes the part of an object behaviour that is obtained by considering only the interactions of that interface and by hiding all other interactions. Hiding interactions of other interfaces will generally introduce non-determinism as far as the interface being considered is concerned.
2 - The phrase "an interface between objects" is used to refer to the binding (see 13.4.2) between interfaces of the objects concerned.[1]

The second paragraph, which associates an interaction to a single interface, is wrong. It would be true in the ANSA architecture [9], in which a binding establishes a shared interface between several objects. The RM-ODP was largely inspired from ANSA, but its concept of interface is different: an ODP interface belongs to a single object, and an ODP binding binds two or more interfaces (see [1, definition 13.4.2] and [7, clause 7.2.3.2]). Therefore, each ODP interaction is necessarily associated with two or more interfaces.

It is also wrong to pretend that an interaction is necessarily associated with a single interface of a participant object. Applied in the context of the ODP computational language, such a statement would disallow an object to bind two of its own interfaces, client and server, and therefore to interact with itself. Peter Linington and Jean-Bernard Stefani confirmed that these two statements do not reflect the RM-ODP authors' intention [5, 6].

We propose to rewrite these two statements as follows: "Each interaction role played by an object belongs to a unique interface of that object. Thus the interfaces of an object form a partition of the interaction roles played by that object." The first note of the definition would need to be revised accordingly, since the roles at other interfaces must be hidden as well.

6.2. Behaviour

Interaction roles are not mentioned at all in the definition of behaviour in the RM-ODP, of which we reproduce here its first sentences:

8.6 Behaviour (of an object)*: A collection of actions with a set of constraints on when they may occur.*
The specification language in use determines the constraints which may be expressed. Constraints may include for example sequentiality, non-determinism, concurrency or real-time constraints. [1]

Knowing which role an object plays in an interaction is critical to knowing its behaviour. For example, asking a question is not quite the same thing as answering it. But the current definition of behaviour implies that it is sufficient to say that an object participates in an interrogation...

7. Summary and conclusions

In writing this paper, we spent a lot of time and efforts trying to understand just one concept, that of interaction. We indeed discovered that it was not really defined in the RM-ODP. In doing so, we had to go past some subtle statements and metaphors, and second-guess the author's original intentions behind them. Our work would have been much simpler, and more effective, had these intentions been documented in the standard itself, or in a companion document.

Our investigations lead us to discover some fundamental flaws in the definitions of *Interface* and of *Behaviour of an object.* These flaws have remained unnoticed (or at least unreported) for more than ten years. However, when one is armed with the understanding that interactions have roles, these flaws are fairly obvious.

While performing our research, we came to better appreciate the considerable difficulty of writing the RM-ODP Foundations: the definitions must be notation independent (even syntax independent), grounded in semantics, very general and widely applicable, and yet precise. Compared with other works of the same nature, ISO has performed excellent work. Unfortunately for the object and software engineering communities, the RM-ODP Foundations have not had yet the impact that they deserve. One reason is that the standard lacks proper explanations and references. We invite ISO experts to provide these missing explanations, rather than just fix the flawed definitions. After all, the new definitions might prove be defective again, in which case the explanations would be sorely missing.

8. References

[1] ISO/IEC and ITU-T, "Open Distributed Processing - Basic Reference Model - Part 2: Foundations," in *Standard 10746-2, Recommendation X.902*, 1995.

[2] OMG, "OMG Unified Modeling Language Specification v. 1.5," OMG, OMG Standard March 2003.

[3] P. F. Linington, "What Foundations does the RM-ODP Need?," presented at Workshop on ODP for Enterprise Computing (WODPEC 2004), Monterey, California, 2004.

[4] ISO/IEC, "Information processing systems - Open systems interconnection - Estelle : a formal description technique based on an extended state transition model," in *Standard 9074*, 1989.
[5] P. F. Linington, "Personal email communication, Aug. 31," 2000.
[6] J.-B. Stefani, "Personal email communication, Sept. 22," 2000.
[7] ISO/IEC and ITU-T, "Open Distributed Processing - Basic Reference Model - Part 3: Architecture," in *Standard 10746-3, Recommendation X.903*, 1995.
[8] P. F. Linington and W. F. Frank, "Specification and Implementation in ODP," presented at 1st Workshop on Open Distributed Processing: Enterprise, Computation, Knowledge, Engineering and Realisation (WOODPECKER'2001), Setubal, Portugal, 2001.
[9] A. J. Herbert, "An ANSA Overview," *IEEE Network*, pp. 18--23, 1994.

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

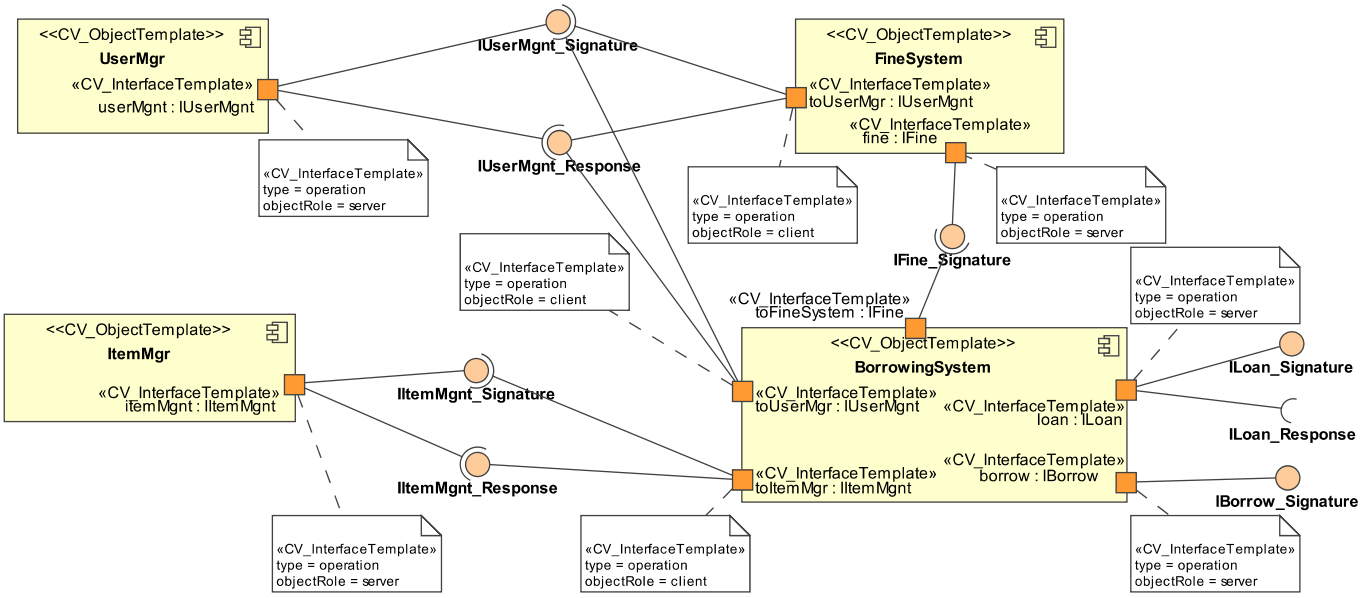

Figure 1. Component Diagram representing ODP Computational Templates

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.

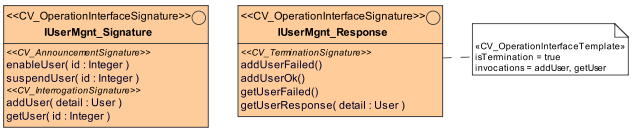

Figure 2. Interface signature for IUserMgnt

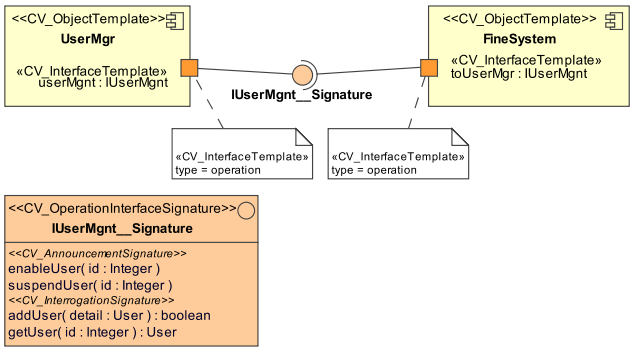

Figure 3. Component diagram following an operational OO approach

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

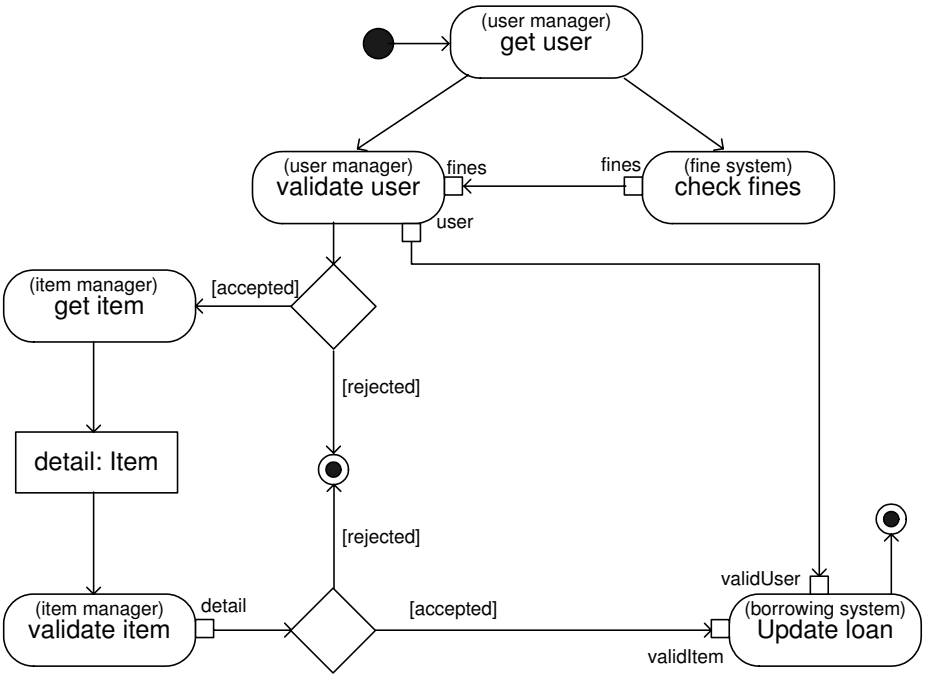

Figure 4. Activity diagram for the Borrowing **Process**

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.

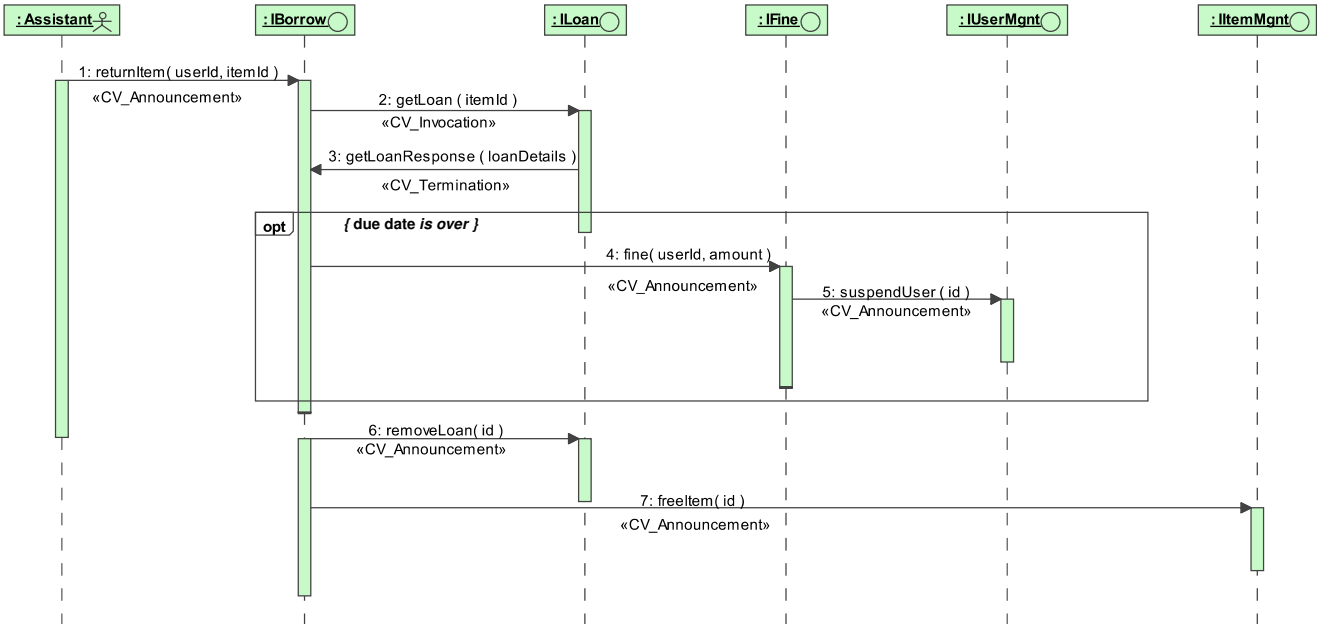

Figure 5. Interaction diagram for the Return Item **Process (excerpt)**

[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.

UML 2 Models for ODP Engineering/Technology Viewpoints – An Experiment

Daisuke Hashimoto
Technologic Arts Inc.
hashimoto@tech-arts.co.jp

Hiroshi Miyazaki
Fujitsu, Ltd.
miyazaki.hir-02@jp.fujitsu.com

Akira Tanaka
Hitachi, Ltd.
tanakaak@itg.hitachi.co.jp

Abstract

The advance of UML® 2.0 standardization work by OMG™ provides a good opportunity for ODP community to leverage UML 2.0 to show the value of ODP. In this paper, we examine issues in applying UML 2.0 to ODP Engineering and Technology Viewpoint Languages, and show how we may be able to use UML to represent those ODP Viewpoint specifications.

1. Introduction

With the wide acceptance of Unified Modeling Language [2] in the industry, there is a growing interest in applying UML to represent ODP [1] viewpoint specifications. This direction is beneficial to both sides, since ODP modelers will eventually be able to get ODP modeling tool based on UML tools, and UML modelers will get a robust way of organizing their UML models. This comes from ISO/IEC and ITU-T's joint project called "Use of UML for ODP system specifications," and also comes from various "Enterprise Architecture [6]" practices where rows in two-dimensional matrix are usually quite similar to ODP viewpoints. The combination of UML and ODP with OMG's MDA initiative [3][4] will create a good foundation for systems' lifecycle management. The content of this paper is a work-in-progress level, based on INTAP's technical report [5], and is a result of elaborating current Committee Draft of "Use of UML for ODP system specifications" standard by our group. In this paper, we identify and examine issues in modeling Engineering and Technology Viewpoints with UML 2.0, and propose one possible UML 2 model diagrams for those viewpoint specifications.

2. Engineering Viewpoint – Issues and discussion for UML modeling

2.1 Target Architectural Diagrams

First we need to identify kinds of architectural diagrams to be described in UML. Some candidates are found in RM-ODP Part 3, Clause 8 Figure 2 to 6. The following figures (R2 to R8) are extracts from RM-ODP Part 3 standard document. In order to avoid confusion, we will add "R" to refer to those diagrams.

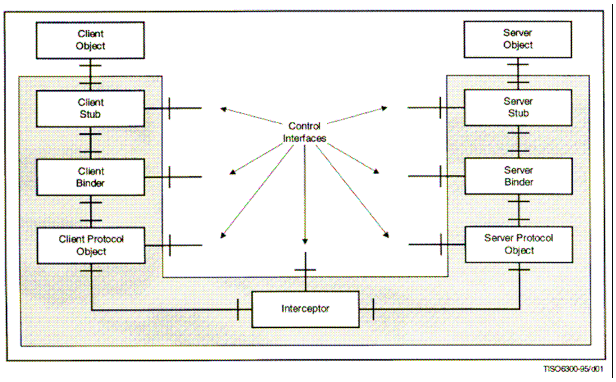

Fig. R2 An example of a basic client / server channel

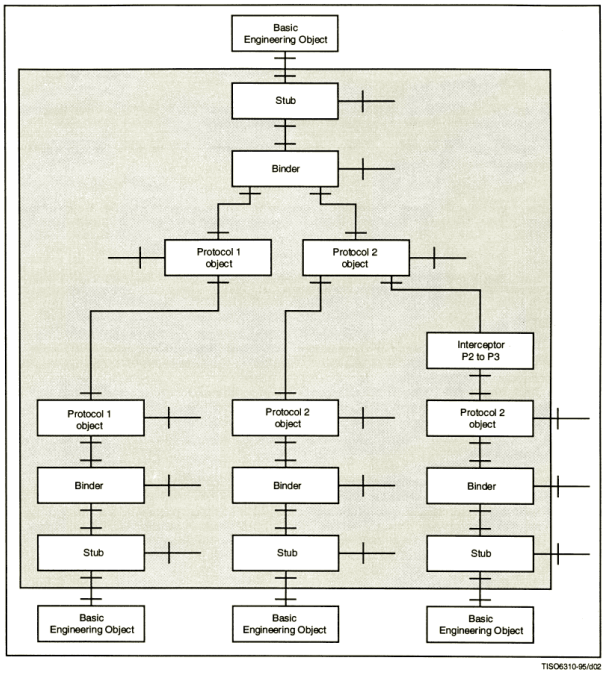

Fig. R3 An example of a multi-endpoint channel

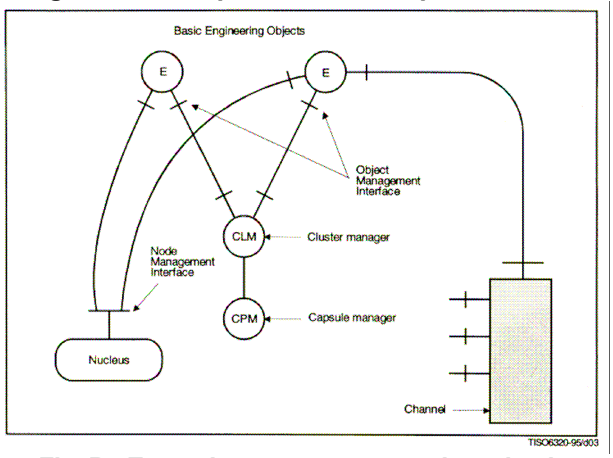

Fig. R4 Example structure supporting a basic engineering object

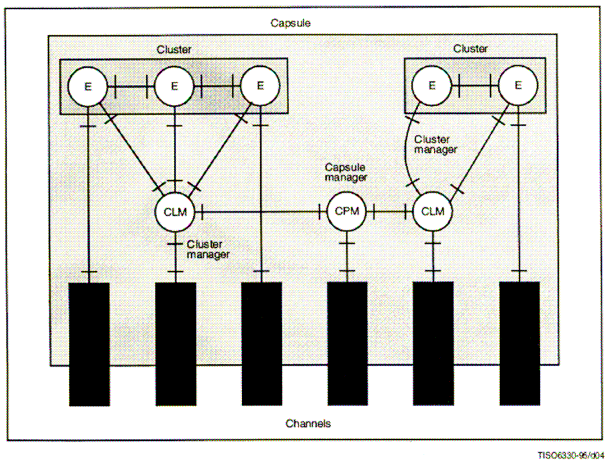

Fig. R5 Example structure of a capsule

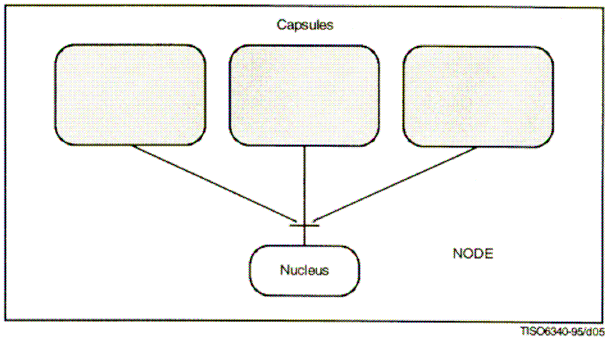

Fig. R6 Example structure of a node

From those example figures, we can identify essential target diagrams, which are Channel structure diagram (covering Figure R2 and R3) and Capsule structure diagram (covering Figure R4, R5, and R6).

In addition, to cover all the Engineering concepts, we will need UML diagrams describing domains and templates.

2.2 Representation of Engineering Concepts

Engineering Object:

The issue is the choice of UML elements representing Engineering Object, which includes at least Basic Engineering Object (we will refer it as BEO in this paper), CapsuleManager, ClusterManager, Stub, Binder, ProtocolObject, and Interceptor. It could be modeled with UML Object (InstanceSpecification of Class), UML Class, UML InstanceSpecification of Component, or UML Component. In either case, UML 2's Structured Classifier will help us describe the internal structure of Channel and Capsule.

Containers:

The next are containers such as Node, Nucleus, Capsule, and Cluster. Those elements are defined as "a configurations of (basic) engineering objects …" in RM-ODP Part 3. It may be possible to use UML Node to represent ODP Node. However, if we take this approach, only available UML diagram will be Deployment Diagram, and UML 2 only allows certain types of modeling elements (e.g. Node, Artifact) to be placed within a Node. Note that it was possible to take this approach with UML 1.4, since it was legal to place Component within a Node. The UML diagram we need has to have a capability to show the internal and logical structure of ODP Node, similar discussion regarding Class or Component in Engineering Object may also apply.

Nucleus:

Nucleus may either be treated as Engineering Object or container. It could be Engineering object, since it provides Nucleus Services to all types of Engineering Objects. It could also be considered as a container, in which Capsule and Containers may reside within.

Channel:

Channel may either be treated as a container or a structured Object or Class etc. The issue associated with Channel is that both Channel and Capsule shares the same Engineering Objects (Stub, Binder, and ProtocolObject) and we do not have overlapping diagrams even in UML 2.0.

Template:

<X> Template is defined in RM-ODP Part 2 as "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." The templates in Engineering Viewpoint Language are Cluster template, Checkpoint, Cluster checkpoint. However, the definitions of those Engineering templates seem more like "snapshot" than the Part 2 definition implies. The choice of UML model element for template has impact on the choice of UML model element for Engineering Object.

2.3 Interactions between Engineering Objects

In Computational Viewpoint, RM-ODP Part 3 provides various concepts to deal with interactions with interface and signature. In Engineering Viewpoint, however, we do not have specific concepts for this purpose, and that makes it difficult to model interactions between Engineering Objects. One possibility is to consider "recursive application of viewpoints." If we can apply Computational Viewpoint Language to interactions between BEOs, we will get a capability to specify this. In that case, the modeling element will represent Engineering Object with Computational aspects. The issue becomes how to do this in UML. For instance, do we want to allow UML Class or Component or InstanceSpecification stereotyped as "«NV_BEO» «CV_Object»?"

In addition, if we are to allow "recursive application of viewpoints" as described here, we would need to consider the alignment of base classes for corresponding viewpoint profile.

2.4 Engineering Functions

In Engineering Viewpoint, there are functionalities defined as a part of its language. Those are Checkpointing, Deactivation, Cloning, Recovery, Reactivation, and Migration. Those are different from other concepts, since they are representing functions possibly including behaviors. One possibility is to apply "recursive application of viewpoints" again, and make use of Enterprise Viewpoint's Objective and Process etc. concepts to model those functions as «EV_Objective» with behaviors as «EV_Process» expressed with Activity Diagram.

2.5 ODP functions

In RM-ODP Part 3, many ODP functions are described. When defining ODP viewpoint specifications, those common functions may need to be explicitly included. An example of ODP function is ODP Trader, where the ODP Trader standard defines Enterprise, Information, and Computational Viewpoint specifications of itself. The issues are how we can identify,

reference, and include those functions in Engineering Viewpoint specifications.

2.6 Domains

Domain concept in RM-ODP is defined as follows. <X> Domain is "A set of objects, each of which is related by a characterizing relationship <X> to a controlling object. Every domain has a controlling object associated with it." It is a set of objects, rather than a configuration of objects. The issue is which UML element is suitable for representing this "a set of objects" concept.

Also, since the same Engineering Object may be a member of several different kinds of domains at the same time (e.g. NamingDomain_1, SecurityDomain_2, and PolicyDomain_3 etc.), UML element representing domain should allow sharing of objects. If we choose Class, parts may be shared. If we choose Component, it may become issues. If we choose Package, we will need to use «import»/«access» etc. to have access to those elements contained in other Packages.

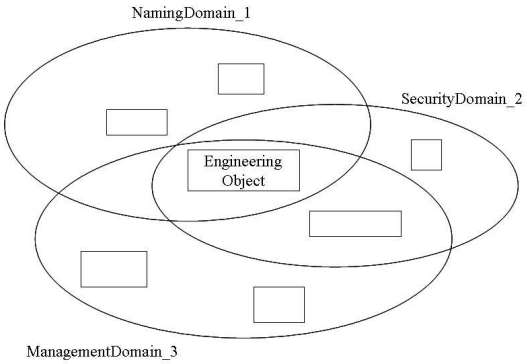

Figure 1 Engineering objects and domains

2.7 Selective Transparencies

Computational Viewpoint specifications are defined in distribution transparent manner, and the degree of distribution transparency may be specified in a "transparency schema" associated with "a specification that uses specific ODP functions and engineering structures to provide the required form of masking." This may be considered as a mapping specification from Computational Viewpoint to Engineering Viewpoint. And, this implies that given one Computational Viewpoint specification, there will be a variety of Engineering Viewpoint specifications, each corresponding to different transparency schema. Although this issue may be mainly related with Computational Viewpoint, it has an impact on Engineering Viewpoint specification and Engineering Viewpoint's correspondence to Computational Viewpoint specification.

2.8 Architectural Styles (or Engineering Templates or Patterns)

Since the day RM-ODP Part 3 became International Standard, five or more years have passed, and now there are various types of commercial and open source middleware which in fact implements most of the Engineering Viewpoint concerns, e.g. CORBA ORBs, J2EE application servers. .NET, Web Services, SOA, and Message Oriented Middleware, etc. Even more, there are new types of middleware emerging for Grid and Utility Computing, Wireless Networking, and Collaboration Environment etc. Also, various best practices and architectural styles (such as MVC) were developed and in use. Now may be a good time to define architectural styles as Engineering Templates and publish them for use in mapping Computational Viewpoint specifications to Engineering Viewpoint specifications. The issue is how to define those architectural styles with UML.

3. Engineering Viewpoint – Possible UML 2 Models

3.1 Profile definition

The first step is to decide which UML element should be used to represent Engineering Viewpoint Language, especially Engineering Object. We also need to consider about related templates. The choice we made in this paper is use of UML Component for Engineering Objects. A Component is closer to real world software component than Class, and UML 2 also provides Component with structuring capability. The following is a Class diagram showing partial UML 2 Profile definition. The stereotypes have prefix NV_, which is a rule for defining stereotype names for Engineering Viewpoint in Use of UML for ODP system specifications standard. Most (not all) stereotypes extend metaclass Component.

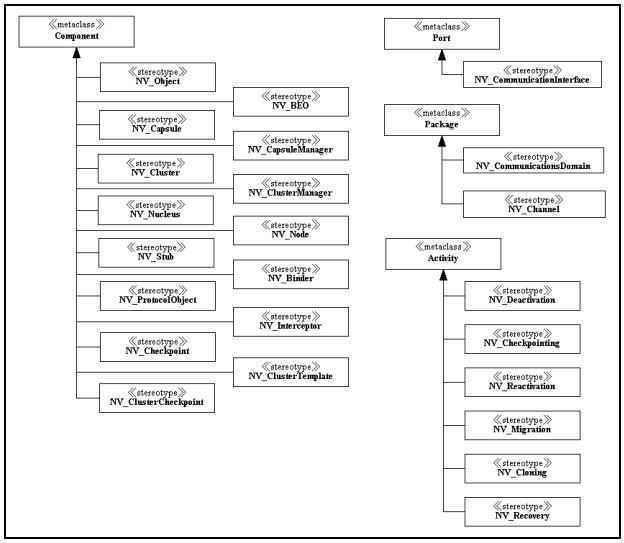

Figure 2 UML 2 Profile for Engineering Viewpoint Language

3.2 Node configuration

The computational to engineering correspondence, or model transformation, is not within the scope of this paper. However, to achieve this, we will at least need a transformation pattern and mechanism. Assuming that there is one, the output N-tier distributed system's node configuration may look like the following. In this case, there are a Node for user interaction, a Node for front-end, a Node responsible for business logic, and two Nodes for persistent data. In essence, BEOs derived form computational objects and engineering objects for providing specified transparency scheme will be deployed on those nodes.

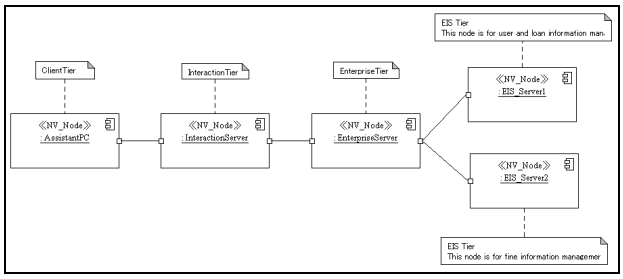

Figure 3 Example Node Configurations

3.3 Node Structure Diagrams

As discussed before, we need to be able to model structure of each Node. Sample structural UML Diagram is shown in Figure 4, which is a UML Component Diagram. We have a Node containing two Capsules, one of which has two Clusters containing two BEOs. There are BEOs (e.g. BEO_1AA) which have access to the services provided by Cluster Manager, which in turn has access to services provided by Capsule Managers. Other BEOs (e.g. BEO_1AB) have access to a Stub for communication, and the Stub interacts with a Binder, and the Binder interacts with a Protocol Object.

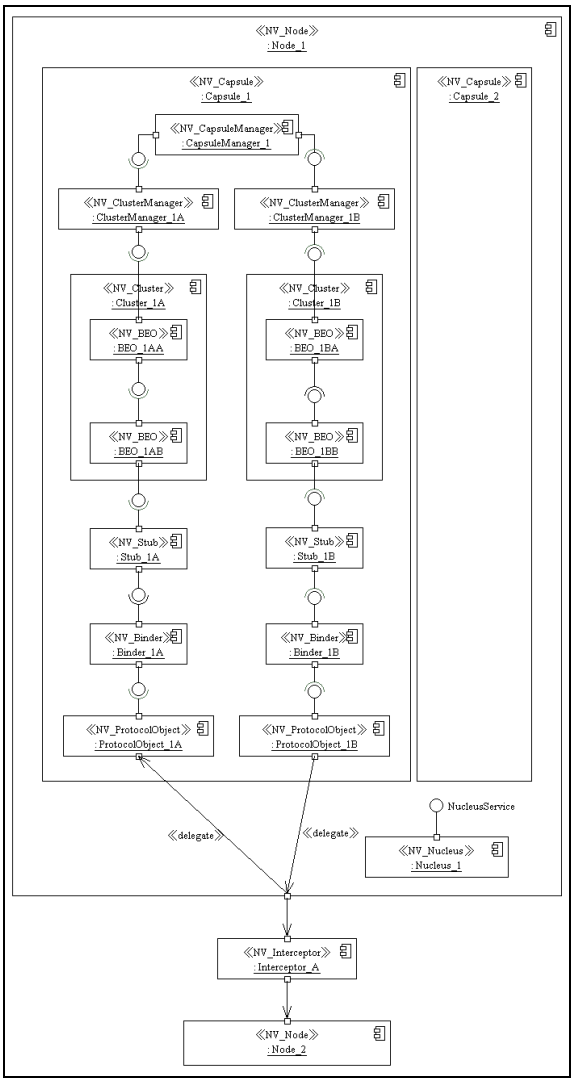

Figure 4 Example UML 2 model for Node, Capsule, Cluster, and BEO

Interesting observation is whether Capsule and Cluster should have interfaces or not, since Capsule is a unit of resource assignment, Cluster is a unit of activations/deactivations, and they have associated Managers. If they have interfaces, then what kind of interface should they be?

3.3 Channel Diagrams

A Channel may be represented with the following UML 2 diagram. The same Engineering Objects appear in Figure 4 and Figure 5 (i.e., Stub, Binder, Protocol Object and Interceptor). Figure 4 shows the structure, interfaces with ports, and services in the hierarchy. Figure 5 shows membership of the Channel Package. Those two diagrams complement with each other to provide different views to the same set of Engineering Objects that make up a part of Node and a Channel. In Figure 5, a channel is defined as a package containing engineering objects, which are the components necessary for enabling communication between Nodes. There are three BEOs involved (not shown): a BEO interacting with Stub_1, a BEO interacting with Stub_2, and a BEO interacting with Stub_3. This channel is defined to serve for those BEOs interacting with each other. The structural aspect, e.g. Stub_1 interacts with Binder_1, is described in structural diagram like Figure 4, and therefore this channel diagram just defines the member engineering objects.

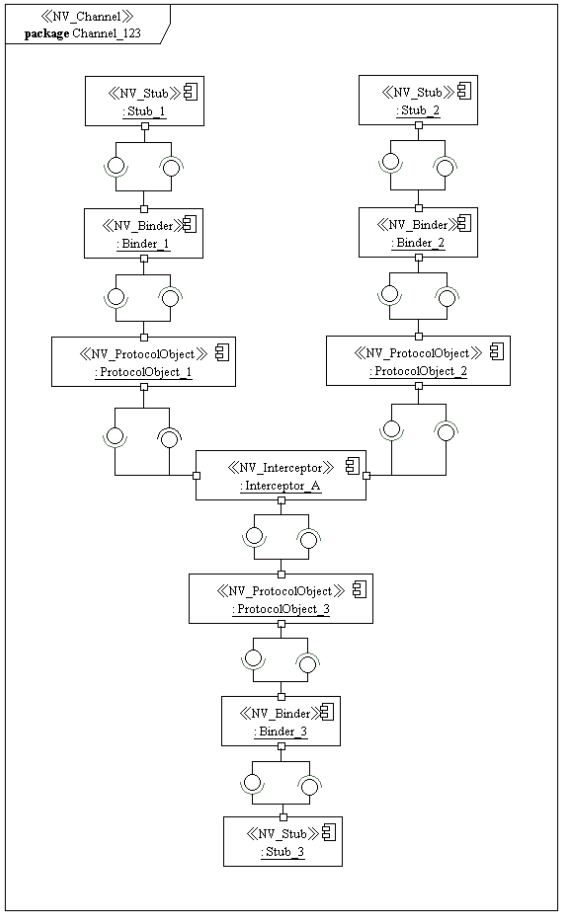

Figure 5 Example UML 2 model for Channel

Figure 4 and 5 together cover most of the figures (R2 to R8) from RM-ODP Part 3, except for control interfaces, which are just one kind of interfaces we can add to engineering objects.

From other perspective, a channel diagram is a package importing necessary member engineering objects from Node

structure packages for communication.

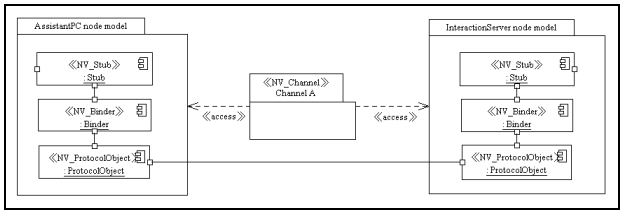

Figure 6 Example Channel

3.4 BEO configuration

Like in computational viewpoint, it is sometimes necessary to specify interactions between BEOs. One of the issues is that the diagram will become complex if BEO configuration is placed in Node Structure diagram (Figure 4). One possibility is to isolate Clusters from the Node Structure and use component diagram (in our case) to describe the interaction. We may need to use e.g. double stereotypes to represent computational aspects of engineering object, e.g. for interfaces, signatures, and interactions for engineering objects. In Figure 7, a Cluster contains two BEOs providing services through the Cluster's Port and Interface.

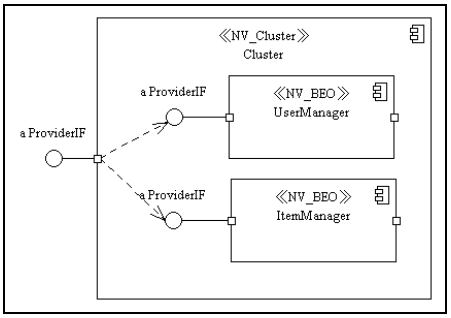

Figure 7 Example basic BEO Configuration

Figure 8 shows a case where two BEOs in a same Cluster have dependency, or interaction. If we need to specify this interaction, interface, and signatures, we may need to borrow stereotype elements from already defined Computational Profile.

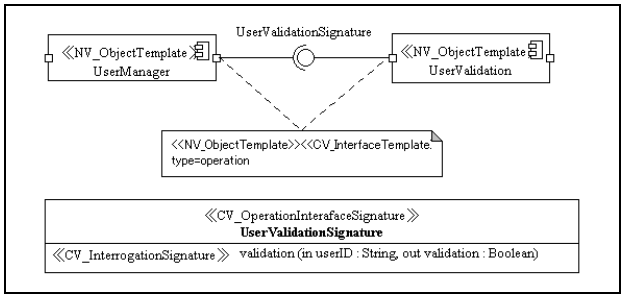

Figure 8 Example BEO configuration including computational aspects

3.4 Domains

Domain may be represented with the following UML 2 diagrams (Packages). One missing element is a Policy that Controlling Object is enforcing. Since Policy concept, at least the base class for it, should be aligned with other viewpoint (i.e. Enterprise Viewpoint), and that is not concluded yet, we did not include it in this diagram.

In Figure 9, CommunicationDomain_A is defined as a package containing a CommunicationControllingObject, ProtocolObject_1 to 3, and an Interceptor_A. Those objects share the same communication protocol, and are able to communication with each other. Note that not all ProtocolObjects included in this package may be instantiated at certain location in time.

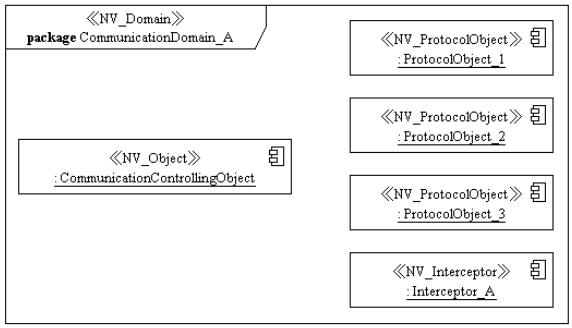

Figure 9 Example Communication Domain

Also from different perspective, a CommunicationDomain can be described as the package importing necessary member engineering objects.
The following diagram shows CommunicationDomain package containing only one engineering object (in this case it is a controlling object) accessing other member engineering objects from various Node structure packages.

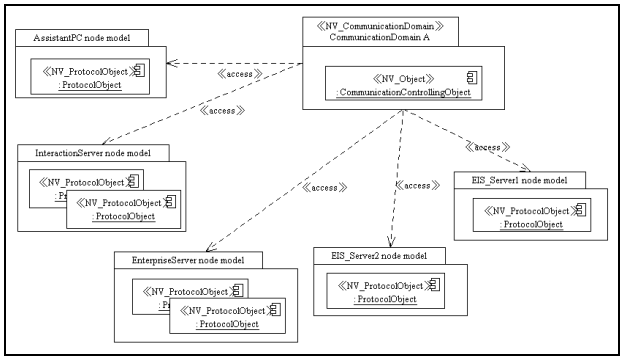

Figure 10 Example Communication Domain (2)

3.5 Correspondence

From Engineering Viewpoint to Computational Viewpoint

Our assumption is that each BEO has one to one relationship to corresponding Computational Object (from Engineering to Computational, not vice versa). This could be expressed as dependency from BEO sub-Package in Engineering Viewpoint Package to Computational Objects in Computational Viewpoint Package in UML. Note that the distribution support mechanism part of Engineering Viewpoint Language model elements may have one to one relationship with specified transparencies in transparency schema.

From Engineering Viewpoint to Technology Viewpoint

This correspondence is closely related to OMG's MDA initiative where Platform Independent Model is transformed to Platform Specific Model. In this case, Platform means real software and hardware platforms. It is our expectation that when

MDA standards get mature, they will become an enabling technology for specifying Viewpoint Correspondence.

Each Engineering Objects may be implemented with multiple Technology Objects, and multiple Engineering Objects may be implemented with one Technology Object. This could be expressed also as dependency between two Viewpoint Packages.

4. Technology Viewpoint – Issues and discussion for UML modeling

4.1 Hardware, Software and Network

In UML Deployment Diagram, main modeling elements are Node and Artifact. Typical representation of hardware elements such as CPU and memory is by the use of tagged values to Node, and we follow this approach. For operating systems and middleware, UML 2 introduced new metaclass ExecutionEnvironment to Node. We can add, if necessary, ODP semantics to this model element. Software implementing BEOs can be modeled as Artifact. CORBA Middleware and POSIX-compliant Operating System, for instance, may be modeled as or based on ExecutionEnvironment. Network like the Internet and LAN can be modeled as a Node, although hardware/software aspects of network can be treated differently.

The discussion may be regarding the extent for defining UML Profile for Technology Viewpoint, since UML 2 provides a similar set of modeling elements. Our approach is to define minimal extensions and to see what is missing based on users' experience.

4.2 Representation of Technology Concepts

Technology Object

Technology Objects, covering software, hardware and network, may be represented with UML Artifact or Node (including Node within Node).

Implementable standard may be considered as a part of specification Technology Objects implements. It could be modeled as UML Class or Component.

Implementation is defined as "a process of instantiation whose validity can be subject to test." A process may be represented with UML Activity Diagram. Since "Process" concept has been refined in Enterprise Language, we may need to refer to related modeling elements in Engineering Viewpoint.

IXIT (Implementation eXtra Information for Testing) may be represented with UML Comments as annotations to Technology Viewpoint specifications. However, there may be a case where an IXIT contains a lot of information and could not be described within a UML Comment, we may need to introduce a mechanism to refer external documents.

5. Technology Viewpoint – Possible UML 2 Models

5.1 Profile definition

The following is a diagram showing a partial UML 2 Profile definition. In order to represent hardware, software, and network as Technology Object, two base classes are used for Technology Object. The stereotypes have prefix TV_, which is a rule for defining stereotype names for Engineering Viewpoint in Use of UML for ODP system specifications standard.

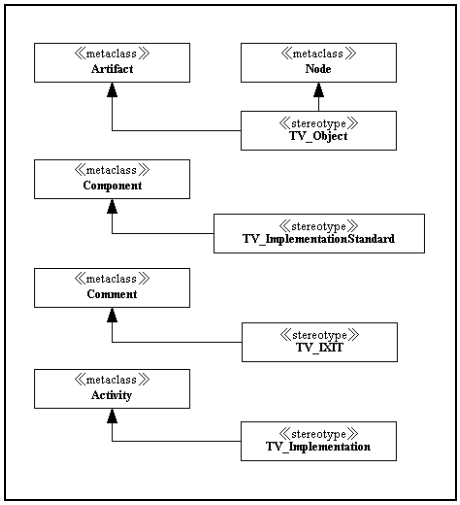

Figure 11 UML 2 Profile for Technology Viewpoint Language

5.2 Node Configuration

In technology viewpoint, we can show network components and hardware components as well as software components. This is the difference with Node Configuration of engineering viewpoint.

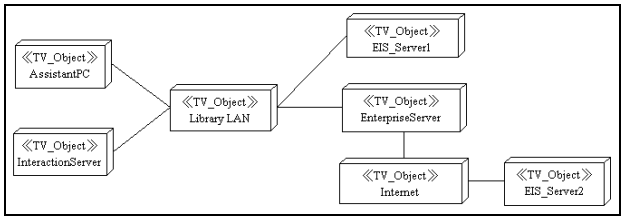

Figure 12 Example Node Configuration

5.3 Node Structure Diagram

The following is an example diagram showing physical structure of a node with hardware, software, and network elements. In this diagram, there is a Node which is connected with the Internet, and which hosts POSIX compliant operating system and J2EE compliant middleware, and two applications are introduced as Artifacts to run under those execution environments.

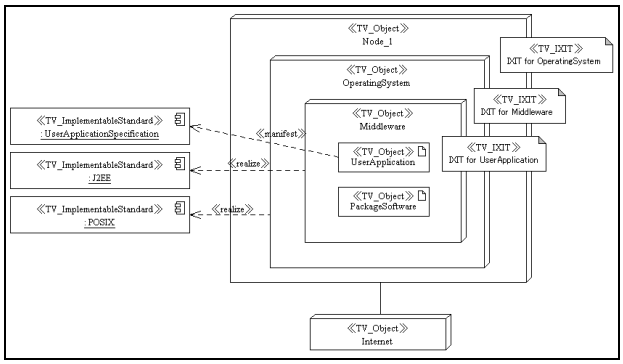

Figure 13 Example Node Structure

5.4 IXIT

IXIT stands for Implementation eXtra Information for Testing, and is one of technology viewpoint concepts. Since this is associated with technology objects, and since we do not have formal way of modeling "extra information," UML Comment is used.

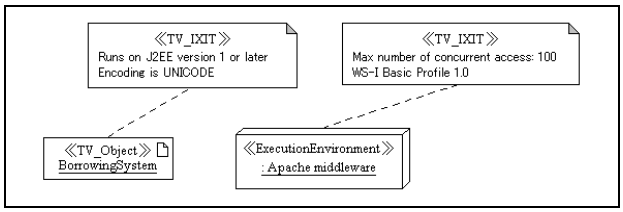

Figure 14 Example IXIT

5.5 Correspondence

From Technology Viewpoint to Engineering Viewpoint

Each Technology Object has one to one relationship to corresponding Engineering Object. This could be expressed as dependency from Technology Objects in Technology Viewpoint Package to Engineering Objects in Engineering Viewpoint Package.

6. Conclusions

There may be multiple ways to represent ODP viewpoint specifications with UML 2.0. Regarding approaches, we believe we have choices e.g. on class-based modeling vs. component-based modeling and on class/component based modeling vs. instance (object/component instance) based modeling. In this paper we took component-based modeling approach. And, if you compare the diagrams presented in this paper with the diagrams in RM-ODP Part 3 (i.e. Figure R2 to R8 in 2.1), we believe we have successfully demonstrated most of the Engineering Viewpoint Specifications with UML 2.0 and its Profiles. We also have demonstrated a possible UML Profile for Technology Viewpoint, which also works. However, we need to develop consistent UML Profiles for all the ODP viewpoints to allow describe different viewpoint's concern in certain viewpoint's specification. Another important point is that the UML mapping should be practical and the results should be accessible and usable. We hope that UML Profile for ODP to be standardized soon and the profile data be developed, published, and become openly available to interested parties, that is, UML modelers and ODP modelers.

Acknowledgements

The work described here was mostly done within INTAP Open Distributed Processing Committee, and includes some ideas and discussions from International and Japanese Committee for SC7/WG19. The authors would also like to thank Mr. David Frankel for his contribution in [5] to this UML and ODP work.

References

[1] RM-ODP, ITU-T Recommendation X.901 to X.904 and ISO/IEC 10746-1 to 4
http://isotc.iso.ch/livelink/livelink/fetch/2000/2489/Ittf_Home/ITTF.htm??Redirect=1

[2] UML 2.0 Superstructure, OMG,
http://www.omg.org/cgi-bin/doc?ptc/2004-10-02

[3] Model Driven Architecture, Document number: ormsc/01-07-01

[4] MDA Guide, Document number: omg/03-06-01

[5] Applying EDOC and MDA to the RM-ODP Engineering and Technology Viewpoints, INTAP Technical Report, David Frankel Consulting, http://www.net.intap.or.jp/e/odp/odp-techguide.pdf

[6] Federal Enterprise Architecture Reference Model, http://www.whitehouse.gov/omb/egov/a-2-EAModelsNEW2.html

Modeling the Engineering Viewpoint of ODP systems with MODERN

Hai-Quan NGUYEN
Université des Sciences et Technologies de Lille (USTL), 59655-Villeneuve d'Ascq Cedex,France
E-mail: Quan.NguyenHai@lifl.fr.

Abstract

Difficulties in software architecture design come from the lack of analysis tools to assist the architect in detecting system errors. Current approaches in architecture design such as Architectural Description Languages (ADLs) are often limited by the way they tackle quality issues separately and only in one viewpoint. None of them addresses analysis and verification of distributed architecture with a global approach from functional design (i.e. liveliness or safety properties) to its deployment in distributed environments (i.e. the environment provides containers and servers to host functional components), and their non-functional qualities (i.e. performance, reliability).

This paper presents our new language for Modeling the Engineering Viewpoint of ODP systems (MODERN) supporting the integrated analysis for different quality properties of RM-ODP systems. Those qualities range from structural and behavioral properties (such as the assembly correctness and the liveness properties) to the quality of services properties (ex. performance).

1. INTRODUCTION

RM-ODP (Reference Model for Open Distributed Processing) [1,2] is an International Organization for Standardization (ISO) 's standard for modeling distributed systems based on separation of concerns of stakeholders. Although RM-ODP provides rich concepts for modeling distributed systems, it does not provide tools and methods for analysis and verification of software quality properties.

In this reference model, ODP provides five viewpoints where each one represents one concern. The Enterprise viewpoint describes client needs and organization policy, the Information viewpoint describes system data, the Computational viewpoint describes business functionality, the Engineering viewpoint describes mechanism supporting distributed communication and Technology viewpoint describes technology utilization. However, although this reference model provides rich concepts for modeling distributed systems in accordance with the different viewpoints, it does not provide languages and tools for modeling and analyzing software qualities.

This work presents our new language MODERN to specify the technical architecture of ODP system, which is specified in the Engineering viewpoint. Actually, the Engineering viewpoint is one of the most important viewpoints in ODP system. In this viewpoint, concerns such as the distribution, the technical concern should be specified in this level. Quality assurance is important in this phrase, which helps to detect conception faults before the implementation choices.

The MODERN language is expected to provide a number of advantages by supporting the integrated analysis. Firstly, analysis tools can be integrated in the same analysis environment. Secondly, the specifications of different analysis concerns are consistent between them. Third, one unique language is used to address the different analysis concerns.

The paper will be organised as following: in the Section 2, we introduce the motivation of our work. After, we present the MODERN language, which support the integrated analysis capacity in the Section 3. Later, we will illustrate our approach with case study and present quickly our prototype in the Section 4. Afterward, we will compare our work to related work in the architectural modeling. Finally, we conclude and give some perspectives of our work.

2. Motivation & Problems

2.1 Why a new language for Engineering viewpoint?

The engineering viewpoint focuses on mechanisms and functions required to support distributed interactions between components in the system. This viewpoint is important because it addresses the problems of component interactions: how the infrastructure and communication mechanisms support distributed interaction.

In RM-ODP reference model, no languages or tools are defined. The architects are free to choose their own methods. In fact, we can use the languages like UML[9], or the architectural description languages (ADLs)[8] to specify the Engineering viewpoint. However, as we will see in the following, those languages fail to model the different aspects of the ODP's Engineering viewpoint. Moreover,

they do not provide sufficient capacity of analysis for the distributed system architecture.

The first choice is highly used in the industry whereas the second choice is studied much in the research field. The UML language, in their latest version 2.0, provides graphic notations for modeling systems. We can access to its dynamic diagrams such as interaction diagram or state chart diagram. However, analysis tools are not supported for UML diagrams. Moreover, concerns such as quality of service assurance are not considered when using this language.

The architectural description languages, in other hand, can provide effective analysis tool. Wright provides analysis tools such as type checking analysis [10], assembly correctness and behavioral verification. However, it does not address the quality of service analysis. Rapide provides simulation capacity but does not provide also this kind of analysis. AADL (The SAE Avionics Architecture Description Language) [31] is a powerful language that provides also a range of the quality analysis such as the dependability properties such as performance or security. However, no means is provided to take into account the new quality property analysis.

2.2 Why do we need integrated analysis ?

We define **integrated analysis** is the capacity of satisfying a well range of quality properties of the system architecture. It is composed of four analysis aspects: the structural aspect analysis, the behavior aspect analysis, the deployment aspect analysis and the quality of service aspect analysis of the component architecture.

The integrated analysis is required when we specify the Engineering viewpoint because it will help us to identify globally the different kinds of potential conception errors of the system. When dealing with distributed system, the architects interest not only in the functionality of the system but also to the mechanism of deployment of the system in the distributed environment.

The integrated analysis can bring many advantages for the architects. The integrated analysis approach has the following qualities:

- Integrated: number of quality properties can be analyzed in the same and unique environment.
- Consistence: specification of the architecture is consistence in accordance to Engineering concepts. Different concepts in the language are verified to assure the consistence between them.
- Unified: One unique language is used to support different analysis aspect of the architecture. The architects reduce their effort in learning different notions when dealing with analysis problems.

2.3 The integrated analysis problems

In our research, we have identified the following analysis problems needed take into account when specifying the Engineering viewpoint:

- Concerning the structural aspect: in this aspect, one of the concerns of the architect is how to assure to interaction between components?
- Concerning the behavioral aspect: we interest in verifying the temporal properties of the architecture. The assurance of those properties validates the functionality of the architecture.
- Concerning the deployment aspect: when deploying the component to the distributed environment, the difficult is to assure the execution of the components in the run-time.
- Concerning the quality of service aspect: the most challenge problem is the integration of new quality of service tool to analyze the technical architecture. The approach is required to be open with new kind of analysis.

MODERN, our modeling language for ODP's Engineering viewpoint, addresses those problems above by introducing not only the concepts supporting the specification of the distributed concerns, but also provide powerful integrated analysis concepts.

3. MODERN: the language for intergrated analysis of the engineering viewpoint

3.1 Overview of the language

The language provides basic concepts for specifying the Engineering viewpoint. Moreover, it provides extra-concepts that support the integrated analysis criterion.

The basic concept of our MODERN language is the component (class *Component Type*). It has the hierarchy structure and is composed of ports (class *Port*), which represent functionality of the component. In the distributed environment (class *Distributed Environment*), the component is deployed to an encapsulation unit called Capsule (class *Capsule*), which has component's life cycle management mechanism thanks to its manager (class *Capsule Manager*).

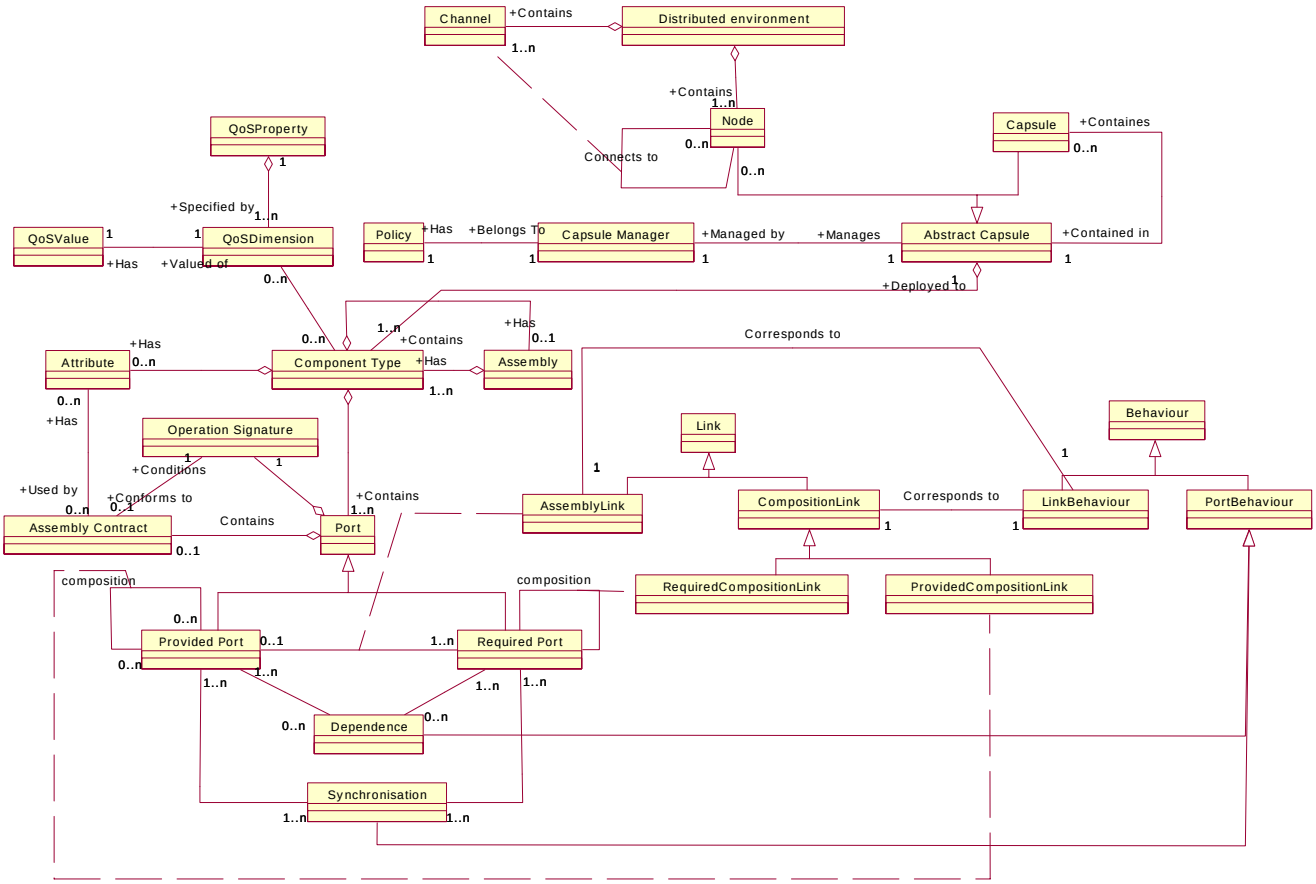

Figure 1: the MODERN meta-model

Besides the basics concepts, the MODERN language provides others concepts in order to archive the integrated analysis. Firstly, we use assembly contract notion helping semantics assembly. Secondly, we provide component's life cycle management to address the deployment analysis. Last but not least, the QoS property is used to integrate new non-functional quality which is the object for the quality analysis.

3.2 Basic concepts

3.2.1 Component

Our architecture is specified with components. The concept of component used in this architecture is based on the following definition from Szyperski[29]: "A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third parties."

The elements needed to describe our component model and the relations between them are taken into account in the MODERN's meta-model represented in Figure 1.

Our approach handles component as component type. Components may have attributes that represent their state. Moreover, they are described by provided and required interfaces. A component communicates with other components through its interfaces. In our model, an interface is represented by a port that is associated with a single service. A service is specified by its signature, which is composed of a name and of ingoing and outgoing parameters.

3.2.2 Components Assembly

The model does not include explicit connectors between components. Nonetheless, if an architect needs one, he can model it in a component. Communications between components or more precisely between their ports are specified by an assembly link. The semantics of a link corresponds to a synchronous call from the required port to the provided port. The choice regarding a port structure involves that a required port can only be bound to a provided port whereas a provided port can be bound to several required ports.

3.2.3 Components Composition

In order to build a complex architecture, we use composite components. They are differentiated from primitive components because they contain

subcomponents which may be primitive components and also composite components hence a recursive definition of a component. The ports of a composite component are called delegated ports. Indeed a call to a provided port of a composite component is forwarded to a provided port of one of its subcomponents. Moreover, a call from a required port of a composite component results from the forwarding of a call from a required port from one of its subcomponents.

3.2.4 Component Distribution

In the ODP's Engineering viewpoint, the most important concern is the mechanisms supporting the distribution. Our language, as the result, uses the following concepts to model the deployment of the distributed architecture.

Capsule

In our model, we consider component type entity of deployment. When component types are deployed, we use the capsule concept to encapsulate those components. We manage those components with deployment policy where we specify the management of life cycle's component. We apply the same politic for component in the same capsule.

Node

A node is one of the most important concepts in our meta-model. It is an abstract concept that represents an addressable machine in the reality. The node describes the place where the component types are deployed following the architect choice. The nodes interact between them via communication network. As we consider the node is a special case of the capsule, a capsule manager manages it. Moreover, it can contain other capsules.

Channel

A communication channel represents a communication way between two nodes in deployment environment. Between two nodes, there is only one channel.

This concept is derived from ODP specification. We consider specify the links but not show how to make possible the binding like in ODP's specification. As opposed to channel concept in ODP, ours does not take into account the binding because the communication and interaction problem have been considered in assembly model.

Deployment environment

The deployment environment composes of nodes and channels. It is where the application is deployed and execute. Normally, this environment is distributed.

3.3 MODERN's Integrated Analysis concepts

In our approach, we determine three categories of the integrated analysis. The first category of integrated analysis concept contains the functionality contract concept supporting structural and behavioral analysis. The second category contains component management policy supporting deployment analysis. The last category contains concepts for modeling the quality of service properties and means to integrate them to the components.

3.3.1 Functionality Contracts

The conditions of validity of a component assembly are improved by associating an assembly contract composed of a pre-condition and a post-condition to each port. These conditions focus on the attributes of the component and on the parameters of the signature. Thus in addition to the verification of the signature compatibility between two linked ports, there is an analysis that checks respectively the compatibility of the pre-condition and pos-condition of a port with the precondition and post-condition of the linked port. Furthermore, behavioral contracts are added to the components. These contracts describe the expected behavior of a component and are used to generate the behavior of the components assembly. An appropriate tool has been developed to check some properties on it.

3.3.2 Component Management Policy

Capsule Manager

The components deployed in the distributed environment are controlled by the capsule manager. In fact, the policy which manages the component's life cycle can modify the level of quality of the application. For example, the passivation of the instances of component during it does not have there a request of these customers can make the resources available for other components. Another example is the instantiation of new instance of component or the reactivation of the existing instances for requests these of the customers.

Then we find that it is necessary to specify the management of life cycle of the components deployed in the environment of reception. This specification enables us to check the policy of management of components which influences the level of quality of the application.

Component's Life Cycle Management Policy

For a component type deployed in a capsule, we specify following information for managing the component's life cycle:

- the maximum number of instances of the component type which the capsule manager can instantiate.
- the maximum number of simultaneous requests treated by the component.
- the policy specification of the component's life cycle. This specification must make it possible to specify activation, passivation, and the termination of the cycle of life of the instances of component.

3.3.3 Quality of Service Property

We define the concepts for analysis the quality properties. Three concepts have been proposed: the QoSProperty (QoSProperty), the analysis parameters (QoSDimension) and its values (QoSValue) associated with component descriptions.

QoS Property

The QoS property represent the quality property which one want to analyze.

QoS Dimension

In order to analyze a quality property, we need to have analysis parameters concerned with this property. For example, to analyze the performance property, we need to provide the arrival rate and the service rate of the service center. Other parameters include the number of server and the server capacity.

QoS Value

QoSValue represent the value in a given time of the QoS Dimension. For example, the dimension service rate has the value of 1000 messages/second.

4. Case study

In the following section, we illustrate how to use the MODERN language to model technical concerns in Engineering viewpoint. We will show how integrated analysis is supported by our MODERN language. The language supports **integrated analysis**. Those analyses are composed of the structural aspect analysis, the behavioral aspect analysis, the deployment and the quality of service analysis.

The most significant example verified by our verification is an existing application from the industrial case [30]. The following figure gives an idea of the complexity of the architecture. Required and provided ports are symbolized respectively by - and +. The example represents the use of a generic coupler of scientific code[30]. This coupler is used to study the interactions between codes of different domains in physics. It manages the exchange of values between the codes. The assembly model is described graphically in the figure 4. Two scientific code components Tree and Support communicate between them via the coupler. We need also to analyze globally the qualities properties of the system.

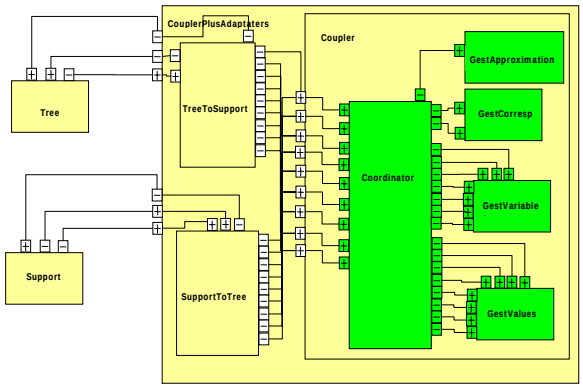

Figure 2: the Coupler application – the assembly view

The following specification using MODERN describes partly the component Tree. It has three ports: two provided port and one required port. In the specification of the Position port, we describe the assembly contract with the pre and post description. In the pre condition of the contract, we specify that the position of x must be superior of 0 and inferior of 100 and the position of y must be in the range of 0 and 100 also.

```
<Component_Type Component_Name="Tree">
   <Attribute attribute_name="att_position_x" type="float"/>
   <Attribute attribute_name="att_position_y" type="float"/>

…

 <Provided_Port port_name="Position">
                <Signature name="void Position(OUT float position_arbre_x,
OUT float position_arbre_y)"/>
                <Assembly_Contract>
       <Pre
expression="position_x<=100&&position_x>0&&position_y>0&&position_y<=100"/
>
                  <Post
expression="position_arbre_x==position_x&&position_arbre_y==position_y"/>
                </Assembly_Contract>
 </Provided_Port>
……
</Component_Type>
```

Figure 3: the Position provided port specification

The links between the components are specified in the figure 4. We show that the Tree component requires the Force port of the Coupler component for its execution whereas the Coupler requires the Speed and the Position port.

```
<Assembly_Link>
        <Required_Port Component_Name="Tree" Port_Name="Force"/>
        <Provided_Port Component_Name="Coupler" Port_Name="Force"/>
   </Assembly_Link>
<Assembly_Link>
        <Required_Port Component_Name="Coupler" Port_Name="Speed"/>
            <Provided_Port Component_Name="Tree" Port_Name="Speed"/>
</Assembly_Link>
<Assembly_Link>
  <Required_Port Component_Name="Coupler" Port_Name="Position"/>
    <Provided_Port Component_Name="Tree" Port_Name="Position"/>
</Assembly_Link>
```

Figure 4: the assembly links between the components

The figure 5 shows the graphic representation of the application in the distributed environment. The Tree component is deployed into the node NodeA, the Support component into the node NodeB and the Coupler into the node NodeC.

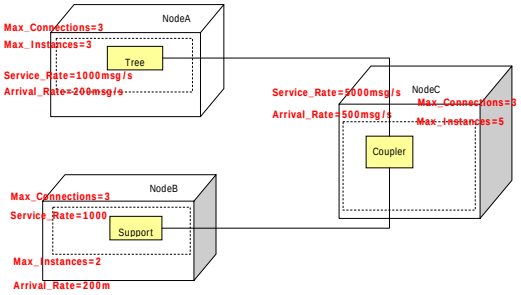

Figure 5: the Coupler application - the quality of service analysis view

As we interest in the performance quality analysis, the following section describes the analysis information for the performance quality. Using MODERN, we describe this quality by four parameters:

- The customer service time distribution (*Service*)
- The customer inter-arrival time distribution (*Arrival_Rate*)
- The number of independent component instances (*Max_Instances*)
- The queue capacity (*Max_Connections*)

The figure 6 describes the technical architecture containing three nodes A, B and C. In the node A, the component Tree is specified with the following performance parameters: the service rate equals to 500msg/second; the arrival rate equals to 100msg/second). The component Support is deployed in the node B with the following performance parameters: the service rate equals to 1000msg/second; the arrival rate equals to 200msg/seconde). The component Coupler is deployed in the C with the service rate equals to 5000msg/second and the arrival rate equals to 500msg/seconde).

```
<Node NodeName= "NodeA">
  <Capsule CapsuleName= "CapsuleTree">
     <Contains_Component Component_Name="Tree">
           <QoSProperty QoSProperty_Name="Performance">
          <QoSDimension Dimension_Name="Service_Rate" Value="500"/>
          <QoSDimension Dimension_Name="Arrival_Rate" Value="100"/>
        </QoSProperty>
     <Contains_Component/>
     <CapsuleManager Manager_Name="Tree_Manager">
           <Policy>
                <MaxInstances Component_Name="Tree" Value="3"/>
                <MaxConnections Component_Name="Tree" Value="5"/>
           </Policy>
     </CapsuleManager>
   </Capsule>
```

```
</Node>
<Node NodeName= "NodeB">
  <Capsule CapsuleName= "CapsuleSupport">
     <Contains_Component Component_Name="Support">
           <QoSProperty QoSProperty_Name="Performance">
          <QoSDimension Dimension_Name="Service_Rate" Value="1000"/>
          <QoSDimension Dimension_Name="Arrival_Rate" Value="200"/>
        </QoSProperty>
     <Contains_Component/>
     <CapsuleManager Manager_Name="Support_Manager">
           <Policy>
                <MaxInstances Component_Name="Support" Value="2"/>
                <MaxConnections Component_Name="Support" Value="2"/>
           </Policy>
     </CapsuleManager>
   </Capsule>
</Node>
<Node NodeName= "NodeC">
  <Capsule CapsuleName= "CapsuleCoupler">
     <Contains_Component Component_Name="Coupler">
           <QoSProperty QoSProperty_Name="Performance">
          <QoSDimension Dimension_Name="Service_Rate" Value="5000"/>
          <QoSDimension Dimension_Name="Arrival_Rate" Value="500"/>
        </QoSProperty>
     <Contains_Component/>
     <CapsuleManager Manager_Name="Coupler_Manager">
           <Policy>
                <MaxInstances Component_Name="Coupler" Value="3"/>
                <MaxConnections Component_Name="Coupler" Value="7"/>
           </Policy>
     </CapsuleManager>
   </Capsule>
</Node>
<Channel Channel_Name="ChannelAB"/>
<Channel Channel_Name="ChannelBC"/>
<Channel Channel_Name="ChannelAC"/>
<Distributed_Environement>
           <Connects_To   Node_Name="NodeA"   Node_Name="NodeB"   Channel=
"ChannelAB"/>
  <Connects_To Node_Name="NodeB" Node_Name="NodeC" Channel= "ChannelBC"/>
  <Connects_To Node_Name="NodeC" Node_Name="NodeA" Channel= "ChannelCA"/>

</Distributed_Environement>
```

Figure 6: Engineering specification

Behavioral analysis

In order to analyze on the behavior of a component model, we transform the behavioral elements into FSP (Finite State Process)[25] processes. We operate this translation first by generating a behavior formed of the behaviors of the composition and assembly links and the dependences and then by making a composition of it with the synchronizations.

The analysis uses a verification tool for concurrent systems, named LTSA (Labelled Transition System Analyser) [26], which supports FSP and a LTL (Linear Temporal Logic) checker to check safety and liveness properties such as deadlocks or absence of reachability.

Deployment analysis

The deployment analysis contains two tests: the validity of the deployment and the component's life cycle management analysis. The first analysis ensures that the components, which are assembled on the assembly, can communicate between them after the deployment process using mechanism provided by the environment. The second analysis verifies the execution correctness of the component in the run-time.

We define this validity a correspondence between technical architecture to be deployed and the distributed environment. As we explain above, we must ensure the correspondence between components to be deployed and the distributed environment.

Quality of service Analysis

In order to analyze the performance, we transfer the description of performance quality to queuing network model. The following section describes the results obtained from the analysis phase. In fact, the result is obtained by using the MCQueue tool.

5. Related work

In the following section, we discuss approaches in modeling the architecture of systems and its analysis methods.

5.1 ODP's related work

There are some efforts to define the language of specification from the Enterprise viewpoint with the UML [3]. In this proposal, it describes the concepts in the viewpoint of Enterprise like the policy, the roles, the community. In other work [4], OCL is used to describe the obligations of objects of company. Romero et al.[5] proposes the use of Maude, a logical language of achievable rewriting (executable rewriting logic), to specify the Computation viewpoint. The rewriting logic is the logic of change which can work with the state and calculations non-determinist. Maude is used to specify the concepts from the Computational viewpoint like the object, the interface, the behavior, the constrained ones of environment.[6] This work aims at formalizing the Computational viewpoint of by proposing the concept action template and the causality control.

In spite of the interesting proposals, we find that these approaches are limited to the languages for the Enterprise viewpoint of and the Computational viewpoint. Other viewpoint such as that Engineering is not treated for this moment.

5.2 Architecture Description Languages

Specifying software architecture requires an architectural language to define all static and dynamic aspects of architecture. These languages define the formalism in terms of component, connector, and configuration.

This clear separation between components, connectors and configuration is a basic concept shared by all ADLs. However, despite a large popularity among the research community, no ADLs have yet reached common practice in companies. However, several ADLs have been introduced like ACME, ArchJava, Aesop, C2, Darwin, Meta-H, Olan, Rapide, SADL, UniCon, Weaves, Wright, or xADL. Medvidovic and Taylor [7] provide a general framework to classify and compare them.

One of the advantages of ADLs is the way the associate with formal techniques. Traditionally, those methods are considered difficult to apply in company environment. They do not provide enough notions and tools to associate with the architecture notion. Besides, there is a lack of methodology and software tools. To overcome this drawback, some ADLs language use formal method as tool for analyzing properties. Wright use CSP for analyzing behavioral properties such as deadlock. Darwin use pi-calculus and later use FSP to analyze the liveness and safety property.

5.3 Quality of service analysis

In [11], Issarny and al, interest in dependability analysis from architectural viewpoint. Their work is based on work in ABAS[12] for dependability system. This work is closed to our problem. However, as there is no separation between functional and distribution concern, it is more difficult to determine where the faults come from. The functional architecture uses the connector which differentiates from our approach which considers it is only a specific component. Moreover, the approach does not provide iterating steps to construct technical architecture as ours.

Other quality analysis method such as queuing network for performance analysis [13,14,15,16,17] or block diagram[18,19] for reliability analysis provide efficient means to analyze qualities. Those approaches need to be integrated into architectural level to facilitate the design and analysis of those non-functional qualities.Some efforts have been made in the field such as the integration of performance technique with the ADL [20,21]. However, those approaches treat only one aspect and only one non-functional quality.

6. Conclusion

In this paper, we present our new language MODERN to specify the Engineering viewpoint of RM-ODP systems. The language provides a high effective manner to satisfy the quality properties of distributed software architecture.

The language has a number of original points:

- It provides rich concepts for specifying Engineering viewpoints of distributed system. Those concepts, which conform to RM-ODP standard, help to model the different concerns such as the functionality and the deployment of the system.
- It supports the global approach for analyzing quality properties. Those qualities contain the functionality properties (structural and behavioral properties of components assembly), the deployment correctness and the quality of service property.

- It is provided with a modeling and analyses environment. Number of properties can be validated thanks to our integrated environment.

In our further work, we are studying a new methodology for automatic transformation from the Computational viewpoint. We are working on the new transformation language that uses the architectural figure concept [22, 23, 24] to generate the Engineering viewpoint from a computational viewpoint specification.

ACKNOWLEDGEMENTS

This research has been originally supported by Electricité de France. The author thank to Professor Laurence Duchien of LIFL, Mr Philippe Bedu, Mr Jean Perrin of EDF, Professor. Duong Vu of Eurocontrol and Professor. Thuc Nguyen Dinh of HCM university for the previous discussions and the comments.

REFERENCES

[1] ISO/IEC. Open Distributed Processing Reference Model - parts 1,2,3,4, 1995. ISO 10746-1,2,3,4 or ITU-T X.901,2,3,4.
[2] J-R. Putman, Architecting with RM-ODP, Prentice-Hal, 2001.
[3] X. Blanc, M.-P. Gervais, R. Le-Delliou, Using the UML Language to Express the ODP Enterprise Concepts, LIP6 1999/024: Technical Research, LIP6, 1999.
[4] J.Putman, Model for Fault Tolerance and Policy from RM-ODP Expressed in UML/OCL, Third IEEE International Symposium on Object-Oriented Real-Time Distributed Computing, 2000.
[5] R.Romero, A.Vallecillo, Formalizing ODP Behavioural Viewpoint Specification in Maude, EDOC, 2004.
[6] R.Romero, A.Vallecillo, Action Templates and Causalities in the ODP Behavioural Viewpoint, Workshop on ODP for Enterprise Computing (WODPEC 2004), 2004.
[7] C. J. Kaufman, Rocky Mountain Research Lab., Boulder, CO, private communication, May 1995.
[8] N.Medvidovic, R.N.Taylor, A Classification and Comparison Framework for Software Architecture Description Languages, IEEE Transactions on Software Engineering, January 2000.
[9] [OMG/UML] OMG, OMG Unified Modeling Language Specification, version 2.0, http://www.omg.org/cgi-bin/doc?ptc/2004-10-02 , 2004.
[10] R. Allen, A Formal Approach to Software Architecture, PhD thesis, School of Computer Science Carnegie Mellon University,Pittsburgh, Mai 1997.
[11] V.Issarny, C.Kloukinas, and A. Zarras, Systematic aid for developing middleware Architectures, Communication of ACM, Vol 45, Num 6, June 2002.
[12] M. H. Klein, R. Kazman, L. Bass, J. Carriere, M. Barbacci, H. Lipson, Attribute-based architecture styles, Proceedings of the First Working IFIP Conference on Software Architecture (WICSA1), 1999, 225-243.
[13] K.S. Trivedi, Probability and statistics with reliability, queueuing, and computer science applications, John Wiley & Sons, 2001.
[14] H.G. Perros, Queuing networks with blocking, Oxford University Press, 1994.
[15] K. Kant, Introduction to computer system performance evaluation, McGrawHill, 1992.
[16] L. Kleinrock, Queuing systems, Wiley, 1975.
[17] S.S. Lavenberg, Computer performance modelling handbook, Academic Press, 1983.
[18] G. Myers, Software reliability - principles and practices, John Wiley and Sons, 1976.
[19] NASA, Reliability block diagrams and reliability modelling, Technical report, NASA Glenn Research Center, May 1995.
[20] H.Garavel,H.Hermanns, On combining functional verificaition and performance evaluation using CADP, INRIA research rapport, 2002.
[21] S.Balsamo,M.Bernardo,M. Simeoni, Performance evaluation at the software architecture level, SFM (School on Formal Methods), LNCS 2804, 2003.
[22] H.Q Nguyen, P.Bedu, L.Duchien, J.Perrin, H.M.Tran, Architectural Figure – a Reuse Pattern for Analysis and Transformation of ODP Based-Systems, IEEE International Conference on Information Reuse and Integration (IEEE IRI-2005), Las Vegas, Nevada, USA.
[23] H.M.Tran, H.Q Nguyen, P.Bedu, L.Duchien, J.Perrin, Figures de Transformation pour des Architectures Logicielles, LMO, 2005.
[24] H.Q Nguyen, L.Duchien, P.Bedu , J.Perrin, Achieving technical architecture with architectural figures, Proceedings of the IASTED International Conference on Software Engineering and Applications, Cambridge, USA, 2002.
[25] J. Maggee and J. Kramer. Concurrency - State Models and Java Program. John Wiley & Sons, 1999.
[26] J. Maggee, J. Kramer, and D. Giannakopoulou. Behaviour analysis of software architectures. In Proceedings of the 1st Working IFIP Conference on Software Architecture (WICSA1), 1999.
[27] F. Bueno, D. Cabeza, M. Carro, M. Hermenegildo, P. Lopez, and G. Puebla. The ciao prolog system: A next generation logic programming environment. Technical Report 3/97.1, CLIP, April 2004.
[28] G. Leavens and Y. Cheon. Design by contract with jml. Draft paper, March 2004.
[29] C. Szyperski, Component Software - Beyond Object-Oriented Programming, Addison-Wesley, 2002, ISBN 0-201-74572-0.
[30] C. Caremoli and J.-Y. Berthou. CALCIUM V2: Guide d'utilisation.
[31] Feiler, P.; Lewis, B.; & Vestal, S. "The SAE Avionics Architecture Description Language (AADL) Standard: A Basis for Model-Based Architecture-Driven Embedded Systems Engineering. " RTAS 2003 Workshop on Model-Driven Embedded Systems, May 2003.
[32] Java API for XML Processing (JAXP), http://java.sun.com/xml/jaxp
[33] MCQueue tool, http://staff.feweb.vu.nl/tijms/

Adopting the Practice of Enterprise Analysis in a Mid-Sized Company

Mary Burns Furr
Adaptis, Incorporated, 1100 Olive Way, Seattle, Washington 98101, USA
mbfurr@acm.org

Abstract

This case study describes initial problems faced by Adaptis, a mid-sized company, when it adopted the practice of enterprise analysis. The first problem was to distinguish enterprise analysis from the practice of domain or business analysis. The second problem was to create straw enterprise models for collaborative modeling that were well-formed and, to the extent practicable, based upon stable industry standards. The study reviews approaches for solving these problems and how they were applied. The results show that a mid-sized company can successfully adopt the value-adding practice of enterprise analysis.

1. Introduction

Adaptis is a business process outsourcer (BPO) of administrative services for healthcare insurance plans. The range of services Adaptis supports includes claims processing, eligibility and benefit management, medical management, reimbursement and financial services, customer service, and decision support. The company, founded in 1996, now has over 150 employees with business operations in multiple locations, including offshore.

The company's success is based on personalized service, compliance with laws and regulations, meeting quality of service and service level agreement targets, and effective use of people, knowledge, and technology.

To successfully deliver outsourced services in a distributed processing environment, Adaptis must understand both its customers' business domains and the business domains that are central to Adaptis' own core competencies: operations management and support, professional services, account management, database services, networking, security, system administration, product management, and system development and integration.

In 2004 Adaptis created an enterprise analyst position to facilitate building intangible corporate assets that foster competitive edge. The first-year goals for the enterprise analyst were to prepare the organization to use enterprise analysis effectively and to create straw models that would jumpstart collaborative modeling of the Adaptis enterprise.

This case study describes Adaptis' first-year experiences with the practice of enterprise analysis. It is structured as follows: section 2 describes initial problems faced in adopting enterprise analysis, section 3 describes sources investigated for problem-solving guidance, section 4 presents the problem-solving approaches applied and the results, section 5 presents future and related work, and section 6 presents conclusions.

2. Problems Faced

2.1. Defining the practice of enterprise analysis

A need of stakeholders (parties impacted by the success or failure of the practice of enterprise analysis at Adaptis) was to understand the differences between enterprise analysis and other types of analysis, such as domain analysis or business analysis. The terms *enterprise*, *domain*, and *business* are often not defined when used in literature or conversation. Another need of stakeholders was to understand the differences between enterprise analysis and enterprise architecture.

2.2. Developing straw models

The optimum means for eliciting requirements from subject matter is a collaborative workshop. However, in a lean and efficient service organization

that solves complex problems within short timeframes, subject matter experts are scarce resources unable to devote extended periods of time to modeling workshops. So, while enterprise analysis was viewed positively by stakeholders, day-to-day operations were viewed as more urgent and important. The enterprise analysis approach had to manage within this real-life constraint. Straw models [1] were viewed as a way to engage stakeholders and discover which enterprise analysis techniques and assets would be of value.

A significant challenge in developing the straw models was finding practical advice, examples, and best practices that addressed the specific practice of enterprise analysis.

Another challenge was locating industry conceptual models and ontologies at an appropriate level of abstraction for enterprise modeling.

As understanding of the environment in which Adaptis interacts increased, it became apparent that a stable enterprise model would require a higher level of abstraction than anticipated. It was then necessary to explain to stakeholders why a higher level of abstraction would add value.

3. Sources Investigated for Guidance

3.1. Dictionaries

Interpretation and application of guidance found in the professional literature was difficult because the same terms were used by different authors and authorities to convey different concepts. Dictionaries were consulted when such a situation presented itself.

3.2. Reference Model for Open Distributed Processing (RM-ODP)

The RM-ODP, a framework for the specification of ODP systems, was explored as both a means to distinguish between the kinds of analysis performed within the organization and, as suggested by [2], a means to "bridge communication gaps between stakeholders."

The framework comprises five viewpoints: enterprise, information, computational, engineering and technology. The viewpoints specify different kinds of stakeholder interests (e.g., business requirements, information modeling, software design, system design, system installation and integration) [1]. Each viewpoint uses a viewpoint language to specify the system characteristics of interest to the owners or users of that viewpoint. For example, the concepts of *community*, *federation*, *purpose*, *role*, *resource*, *process*, *policy*, and *accountability* are the focus of the enterprise viewpoint, whereas the concepts of *things*, *actions*, and *relationships* (described by *invariant properties* and *schemas*) are central to the information viewpoint [3, 4]. The following points [1] were perceived to be of particular importance to the practice of enterprise analysis:

- A complete specification consists of all five viewpoints, but not all viewpoints in a particular specification may have properties with interesting semantics.
- Different viewpoints are not more general or detailed representations of the same concepts.
- Abstraction levels are used to express a viewpoint at different levels of detail.
- Concepts specific to a viewpoint, as defined by the RM-ODP, may be used in other viewpoint specifications when it is useful to do so.
- All viewpoints expressed in particular specification should be consistent.

3.3. Zachman Framework for Enterprise Architecture

The Zachman Framework [5] is, according to [6], "a classification schema used to organize an enterprise's artifacts, thinking, reasoning, and communicating among the participants of the enterprise." The Zachman Framework describes an enterprise architecture using two independent aspects [7]:

- *perspectives* that frame the view of a business, a situation, an opportunity, or a system
- *dimensions* that specify a perspective based on the basic interrogatives *what*, *how*, *where*, *who*, *when*, and *why*

The planner and business owner perspectives of the Zachman Framework were perceived to be the most useful for the practice of enterprise analysis. The contextual view of the planner considers the complete problem area relative to a single perspective. The conceptual view of the business owner considers the boundary and area of concern of a project or enterprise [6].

3.4. The Open Group Architecture Framework Version 8 (TOGAF 8)

TOGAF 8 is an industry standard architecture development method and resource base that has been developed by members of The Open Group Architecture Forum. A core part of TOGAF 8 is the Architecture Development Method (ADM), an approach for developing a description of an enterprise architecture that meets the business needs of the enterprise [8]. Another core part of TOGAF 8 is the Enterprise Continuum, a repository concept that facilitates use of assets such as models and patterns during the process of developing an enterprise architecture. A third core part of TOGAF 8, the Resource Base, is a set of resources such as guidelines and templates that can help implement the ADM [9].

3.5. The Object Management Group Model Driven Architecture (MDA)

MDA is a systems development approach sponsored by the Object Management Group (OMG). MDA leverages the power of models by separating their use from specific development methodologies and specific technologies so as to preserve "a company's core software assets in the constantly changing world of information technology" [8]. MDA also uses viewpoints, which are described as a technique for abstraction using a selected set of architectural concepts and structuring rules to focus on particular concerns within a system. Of particular relevance to the practice of enterprise analysis are the concepts of computation independent viewpoint and Computation Independent Model (CIM). A CIM [8]:

- focuses on the business environment in which a system will be used
- is commonly referred to as a domain or business model
- is oriented to the stakeholders of the system
- helps understand a problem and share vocabulary

Other points about the MDA that are relevant to the practice of enterprise analysis are: a CIM may consist of more than one model; a CIM typically communicates the RM-ODP enterprise or information viewpoint; and OMG envisions developing standard domain models for industries (e.g., healthcare) [8].

3.6. Ontologies

An ontology is a shared conceptualization of a particular domain that allows people to reason about concepts and to derive mappings for establishing semantically correct communication channels [10]. Reviews of the ontology literature were investigated to discover ontologies of enterprise concepts [10, 11, 12].

3.7. Industry-Specific Standards

Standards for specific industries were also investigated to discover enterprise-level concepts. Standards for clinical and administrative healthcare data are developed by Health Level 7 (HL7), a standards development organization accredited by the American National Standards Institute (ANSI) [13]. Healthcare insurance data standards for the exchange of information for healthcare administration are developed by the ANSI Accredited Standards Committee (ASC) X12 [14]. Other sources investigated are the works of the XML.org Insurance and Healthcare focus groups [15] and the OMG Healthcare and Insurance domain task forces [16, 17]. The North American Industry Classification System (NAICS) was investigated as a means to conceptualize the industries with which Adaptis interacts [18].

3.8. Off-the-Shelf Models

Two kinds of off-the-shelf models were investigated as a potential source of enterprise concepts: universal data models and process maps. According to [19], a universal data model is "a generic or template data model that can be used as a building block to jump-start development of the corporate data model, logical data model or data warehouse data model." Healthcare, insurance, and professional services universal data maps [20, 21] and SAP Business Maps for healthcare and insurance [22] were of particular interest for this effort. Another source investigated was the Process Classification Framework (PCF) developed by the American Productivity & Quality Center (APQC). The PCF is a high-level, industry-neutral enterprise model that allows organizations to view activities from a cross-industry process viewpoint [23].

4. Problem-Solving Approaches and Results

4.1. Approach to define the practice of enterprise analysis

To sort out how the practice of enterprise analysis should be integrated with other organizational activities, such as enterprise architecture, domain analysis, and business analysis, the following questions were addressed.

What is an enterprise? A definition of the term *enterprise* that reflects the environment in which a BPO operates is: A group of people organized for a particular purpose to produce a product or provide a service [6].

What is a domain? The term *domain* is often used without formal definition, which implicitly assumes readers have a common understanding of its meaning. In practice, it is difficult to reach a common understanding of the term when it is used in the context of modeling. The following dictionary definitions reflect the two primary uses of the term found in the literature investigated:

- a sphere of knowledge, influence, or activity [24]
- a knowledge domain that one is interested in or is communicating about [25]

The first dictionary definition connotes the concepts of realm and dominion. An example of this use is found in [26], where a domain is defined as a "set of objects, each of which is related by a characterizing relationship to a controlling object."

The second dictionary definition connotes a particular body of knowledge and associated semantics. An example of this use is found in [27], where a domain is defined as "an area of knowledge or activity characterized by a set of concepts and terminology understood by practitioners in that area."

Although a quick Internet search revealed many uses of the term, it was concluded that, within the context of modeling an enterprise or business (in contrast, for example, with modeling a data base or network), the term *domain* connotes a realm of influence or a specific body of knowledge.

What is a business? The term *business* is also often used without formal definition. The following are two definitions found in dictionaries:

- a usually commercial or mercantile activity engaged in as a means of livelihood [24]
- a commercial or industrial enterprise and the people who constitute it [25]

The second definition fails to distinguish a business from an enterprise in a meaningful way. For the purpose of establishing context for the practice of enterprise analysis, the first definition best distinguishes a business (i.e., a means of livelihood) from an enterprise (i.e., a group of people organized to achieve a purpose).

What is the difference between enterprise analysis, domain analysis, and business analysis? The following definitions clarified to stakeholders the differences between these terms:

- Enterprise models specify the purpose, processes, entities, and organization of one or more enterprises [28], and creating enterprise models is the practice of enterprise analysis.
- Domain models specify the important classes (i.e., concepts, including entities and processes) and vocabulary of a knowledge domain [27], and creating domain models is the practice of domain analysis.
- Business models specify the business and non-physical system requirements of a specific customer [27], and creating business models is the practice of business analysis.

The above clarification of analytical viewpoints and model content was helpful, because, in practice, a variety of positions (e.g., business architect, business analyst) routinely perform one or more of the above analyses depending upon the needs of a particular project. Clarification aligned efforts and prevented duplication of effort.

What remains problematical is defining level of abstractions to guide the universe of discourse [29] as customer business requirements are refined during a systems development project. The TOGAF 8 concept of Enterprise Continuum [9] and mappings between the RM-ODP, TOGAF 8, MDA, and Zachman Framework [30, 31] will be investigated to address this challenge.

How is enterprise analysis different than enterprise architecture? The following definitions clarified to stakeholders the differences between these terms:

- *Enterprise analysis* produces high-level, non-physical descriptions and models of an enterprise's motivations and strategy for survival; industry entities, events, processes, standards, and laws; industry business patterns, collaborations and use cases; and an enterprise's structure, including geographic sites and major communication nodes. The knowledge formalized by enterprise analysis aligns and informs marketing, product

management, architecture, business analysis, and system design activities.

- *Enterprise architecture* is the process of reasoning about the continual needs of integration, alignment, change, and responsiveness of the business to technology and to the marketplace through the development of models and diagrams [6]. Enterprise architecture builds the foundation needed to survive and adapt to present and future business challenges.

4.2. Approach to develop straw models

A BPO system may interact with systems from more than one line of business, more than one customer, more than one regulatory jurisdiction, and more than one industry. To facilitate flexibility and re-use, a BPO needs generic models that can be instantiated to specify the system's interactions within a particular environment. Creating these kinds of generic models requires mapping and harmonizing concepts and semantics across a wide spectrum of organizations. For example, each customer of a BPO has an enterprise viewpoint, each industry in which a BPO interacts has a generic enterprise viewpoint, and the BPO itself has an enterprise viewpoint.

The following describes the efforts taken to develop straw models, and in the process of doing so, achieve the understanding expressed in the preceding paragraph.

Getting started. The first step was to get gain perspective of the problem area. The Zachman Framework helped establish context and, when complexity began to cloud the mind, keep focused on the basic interrogatives of what, how, where, who, when, and why. Creating lists [6] of things to describe the planner perspective, though simple in practice, was a powerful tool.

Discovering stable conceptualizations. High-level, stable industry models helped discover and name enterprise concepts. Particularly useful were NAICS, universal data and process models, the PCF, SAP Business Maps, and ASC X12 standards for electronic data interchange. Although most industry ontologies and domain models were viewed as too immature for practical application during the timeframe of this case study, they will be investigated again as the straw models are verified and refined.

Finding a useful level of abstraction. The RM-ODP enterprise viewpoint provided insight on how to create well-formed, stable system specifications of the Adaptis enterprise and the multi-industry environment in which it exists. Compelling arguments about the power of abstraction to manage complexity were helpful in persuading stakeholders to view enterprise analysis as an investment rather than a cost [1, 32, 33, 34]. The following describes the pragmatic course charted to develop straw models:

- To model the federation of communities with which Adaptis interacts (including entities, policies, relationships, and collaborations), produce an RM-ODP enterprise viewpoint specification.
- To specify an internal viewpoint of the Adaptis enterprise, produce a set of lists to specify the planner perspective of the Zachman Framework.
- To specify the business owner viewpoint, produce UML models suggested by mappings of TOGAF 8 and MDA to the Zachman Framework where they add significant value to the enterprise architecture.
- To harmonize industry concepts and processes, produce domain models that provide significant value to development projects.

5. Future Problems to Be Addressed

5.1. Modeling business events

The term *business event* is a first-order concept in day-to-day conversation with business owners. Conceptualization of business events, decisions, and event response processes is necessary for effective specification of the enterprise, system services, and workflow management requirements [35, 36, 37, 38, 39, 40]. Use of the RM-ODP viewpoints to model and communicate with business owners about business events will be explored (e.g., in the same vein that events are specified for the planner perspective using the Zachman Framework).

5.2. Modeling product management concepts

Product management concepts such as *product* and *marketing feature* need to be expressed and integrated more effectively in specifications and models of the Adaptis enterprise. The literature addressing the use of domain analysis as a basis for product framework design will be investigated to address this issue. An approach for integrating the product management concepts with the Zachman Framework, found in [41], will also be investigated. Patterns for specifying

product-related concepts, such as those found in [42], may also be helpful.

5.3. Solving wicked problems

The concept of *wicked problems* was proposed by [43] when, in the context of social planning, the wicked nature of ill-defined design and planning problems was contrasted with the relatively tame problems of mathematics, chess, or puzzle solving [44]. A central concept in solving wicked problems is *issue*. The use of issue-based information systems, therefore, is proposed by [45] to track the issues and decisions made during the nonlinear process of solving a wicked problem [46].

The reality of wicked problems became apparent during an Adaptis project in which the possibility of leveraging the knowledge formalized in the enterprise straw models to improve access to the Adaptis knowledge base was explored. When business analysts were asked what of kinds information they need to perform their knowledge work more effectively, they indicated their primary need was information about the history of a problem and the context and state of related artifacts (e.g., issues raised, decisions made, what still needs to be done, cross-project interdependencies).

As a result of this needs assessment, it became apparent that, to avoid making an existing information overload problem worse when enterprise analysis specifications and models are added to the Adaptis knowledge base, the means to organize, relate, access, and maintain corporate knowledge must be investigated. The use of RM-ODP concepts to model this problem will be investigated. The TOGAF 8 concept of Enterprise Continuum may also be helpful. An ontology for group memory [47] will be reviewed as a basis for creating an Adaptis ontology and classification scheme for specifications, models, designs, test cases, and procedures. Protégé-2000 [48], an open source ontology editor and knowledge base framework, will be investigated as a tool to support this effort.

6. Conclusions

There was sufficient guidance available in the sources investigated to successfully initiate the practice of enterprise analysis at Adaptis. It is expected that evolution of industry frameworks, standards, and ontologies will necessitate refactoring the initial models, even at the relatively stable enterprise-level of discourse. The return on investment from enterprise analysis, however, is perceived to outweigh its cost and the risk of failure.

Preparing the organization for the integration of enterprise analysis has been a significant effort. Applying organizational change management best practices [49] was helpful in this regard, and management support was essential.

Acknowledgments

The support of Adaptis during this effort is deeply appreciated. In particular, I wish to thank Jan Goetz, Karen Storey, the Adaptis Product Development team, and the Adaptis Enterprise Architecture team.

References

[1] H. Kilov, *Business Models: A Guide for Business and IT*, Prentice Hall PTR, 2002.

[2] H. Kilov, "Using RM-ODP to Bridge Communication Gaps Between Stakeholders," *Proc. 1st Workshop on ODP and Enterprise Computing* (WODPEC 2004), IEEE Digital Library, 2004.

[3] *ITU-T Recommendation X.903|ISO/IEC 10746-3, Information Technology - Open Distributed Processing - Reference Model (RM-ODP): Architecture*, ISO/IEC, 1996.

[4] *ITU-T Recommendation X.911|ISO/IEC 15414, Information Technology, Open Distributed Processing – Reference Model (RM-ODP): Enterprise Language*, ISO/IEC, 2002.

[5] J. Zachman, "A Framework for Information," *IBM Systems Journal*, vol. 26, no. 3, 1987, pp. 276-292.

[6] O'Rourke, C., N. Fishman, and W. Selkow, *Enterprise Analysis Using the Zachman Framework*, Course Technology, 2003.

[7] Pereira, C. and P. Sousa, "A Method to Define an Enterprise Architecture Using the Zachman Framework", *Proc. Fourth Symposium on Applied Computing* (SAC 2004), ACM Press, 2004.

[8] *TOGAF ADM and MDA - Revision 1.1*, The Open Group and OMG, 2005; http://www.opengroup.org/cio/MDA-ADM/.

[9] *TOGAF Version 8.1*, The Open Group, 2003; http://www.opengroup.org/architecture/togaf8-doc/arch/.

[10] P. Bertolazzi, C. Krusich, and M. Missikoff, "An Approach to the Definition of A Core Enterprise Ontology: CEO," *Proc. International Workshop on Open Enterprise Solutions: Systems, Experiences, and Organizations* (OES-SEO 2001), Luiss Publications, 2001.

[11] V. Devedzic, "Ontologies: Borrowing from Software Patterns," *intelligence,* 1999, pp. 15-24.

[12] N. F. Noy and D. L. McGuinness, “Ontology Development 101: A Guide to Creating Your First Ontology,” *Knowledge Systems Laboratory*, 2001; http://www.ksl.stanford.edu/KSL_Abstracts/KSL-01-05.html.
[13] *Health Level 7*; http://www.hl7.org/.
[14] *ASC X12*; http://www.x12.org/x12org/index.cfm.
[15] *XML.org*; http://xml.org/.
[16] *OMG Healthcare Domain Task Force*; http://healthcare.omg.org/.
[17] *OMG Finance Domain Task Force*; http://fdtf.omg.org/.
[18] U. S. Office of Management and Budget, *North American Industry Classification System: United States*, Washington: Government Printing Office, 2002.
[19] L. Silverston, “Is Your Organization Too Unique to Use Universal Data Models?,” *Data Management Review*, September, 1998; http://www.dmreview.com/article_sub.cfm?articleId=425.
[20] L. Silverston, *The Data Model Resource Book: A Library of Universal Data Models for All Enterprises*, Vol. 1, Wiley, 2001.
[21] L. Silverston, *The Data Model Resource Book: A Library of Universal Data Models by Industry Types*, Vol. 2, Wiley, 2001.
[22] *SAP Business Maps: When You're Going Somewhere New, It Helps to Have a Good Map*, SAP; http://www.sap.com/businessmaps.
[23] *Process Classification Framework,* APQC; http://www.apqc.org/portal/apqc/site/content?docid=115313.
[24] *Merriam-Webster Online Dictionary*; http://www.merriam-webster.com.
[25] *UltraLingua English Dictionary*; http://www.ultralingua.net.
[26] *ITU-T Recommendation X.902|ISO/IEC 10746-2, Information Technology - Open Distributed Processing - Reference Model (RM-ODP): Foundations*, ISO/IEC, 1996.
[27] I. Jacobson, F. Booch and J. Rumbaugh, *The Unified Software Development Process*, Addison Wesley, 1999.
[28] C. Marshall, *Enterprise Modeling with UML: Designing Successful Software through Business Analysis*, Addition Wesley, 2000.
[29] T. Halpin, *Information Modeling and Relational Databases: From Conceptual Analysis to Logical Design*, Morgan Kaufmann, 2001.
[30] D. Frankel et al., “The Zachman Framework and the OMG’s Model Driven Architecture, *Business Process Trends*, September, 2003; http://www.omg.org/mda/mda_files/09-03-WP_Mapping_MDA_to_Zachman_Framework1.pdf.
[31] *Mapping the TOGAF ADM to the Zachman Framework*, The Open Group; http://www.opengroup.org/architecture/togaf8-doc/arch/p4/zf/zf_mapping.htm.
[32] M. Cline, and M. Girou, “Enduring Business Themes,” *Communications of the ACM*, May 2000, pp. 101-106.
[33] K. Bennett, and V. Rajlich, “Software Maintenance and Evolution: A Roadmap,” *Proc. 22nd International Conference on Software Engineering* (ICSE '00), ACM Press, 2000.
[34] M. Fayad, and A. Altman, “An Introduction to Software Stability, *Communications of the ACM*, September 2001, pp 95-98.
[35] B. von Halle, *Business Rules Applied: Building Better Systems Using the Business Rules Approach*, Wiley, 2002.
[36] J. Sowa, and J. Zachman, “Extending and Formalizing the Framework for Information Systems Architecture,” *IBM Systems Journal*, vol. 31, no. 3, 1992, pp. 590-616.
[37] D. Marca, and B. Perdue, “A Software Engineering Approach and Tool Set For Developing Internet Applications,” *Proc. 22nd International Conference on Software Engineering* (ICSE '00), ACM Press, 2000.
[38] M. Cilia, and A. Buchmann, “An Active Functionality Service for E-Business Applications,” *ACM SIGMOD Record*, vol. 31, 2002, pp. 24-30.
[39] K. Taveter, and G. Wagner, “Agent-Oriented Enterprise Modeling Based on Business Rules,” *Proc. 20th international Conference on Conceptual Modeling*, Springer-Verlag, 2001.
[40] B. Daum, and U. Merten, *System Architecture with XML*, Morgan Kaufmann Publishers, 2003.
[41] P. Ferdinandi, *A Requirements Pattern: Succeeding in the Internet Economy*, Addison-Wesley, 2002.
[42] J. Arlow, and I. Neustadt, *Enterprise Patterns and MDA: Building Better Software with Archetype Patterns and UML*, Addison-Wesly, 2003.
[43] Rittel, H., and M. Webber; "Dilemmas in a General Theory of Planning,” *Policy Sciences*, Elsevier Scientific Publishing Company, vol. 4, 1973, pp 155-169.
[44] *Wicked Problems*, Wickipedia; http://en.wikipedia.org/wiki/Wicked_problems.
[45] H. Rittel, and D. Noble, “Issue-Based Information Systems for Design,” *Working Paper 492*, The Institute of Urban and Regional Development, University of California, 1989.
[46] J. Conkin, “Wicked Problem Solving and Social Complexity,” *CogNexus Institute*, 2003; http://cognexus.org/wpf/wickedproblems.pdf.
[47] J. Vasconcelos, et al., “A Group Memory System for Corporate Knowledge Management: An Ontological Approach,” *Proc. 1st European Conference on Knowledge Management* (ECKM’2000) Bled School of Management, 2000.
[48] Stanford Medical Informatics, “What is protégé?,” *protégé*, *Stanford University School of Medicine*, 2005; http://protege.stanford.edu/overview/.
[49] LaMarsh J., Changing the Way We Change: Gaining Control of Major Operational Change, Addison-Wesley, 1995.

Generic model for services: health domain study

Zoran Milosevic

CRC for EDST,
University of Queensland,
Brisbane, QLD 4072, Australia.
zoran@dstc.edu.au

Abstract

This paper provides a broader view on services than current Service Oriented Architecture (SOA) approaches. We analyse a generic concept of service from economic, legal and business perspectives and discuss the implications for the health domain. This enables us to develop an ODP enterprise viewpoint of the service concept and its relationship with the ODP computational viewpoint, closely linked with the SOA concepts. The health domain provides a rich base for developing a comprehensive view on services. This is because of the inherent complexity of the domain with many actors, policies and dynamics involved in an increasingly multi-organisational and multi-jurisdictional context. The paper also provides a contribution in positioning services as part of an overall enterprise architecture for the health domain.

Keywords: *Service Model, RM-ODP, SOA, Health Domain*

1. Introduction

The recent popularity of Service Oriented Architecture (SOA) approaches can be explained by the fact that they promote modularity and enable developing and restructuring of ICT service units in business congruent structures. This is an attractive alternative to the monolithic applications of the past, that resulted from the structure clash between business systems and ICT systems. Such monolithic applications, offered by different vendors and deployed in a fragmented way, have resulted in IT systems that often provide duplicated and inconsistent views on data and functionality. This situation has resulted in many difficulties in managing IT systems, including change management in response to new requirements. This state of affairs is evident in many industry domains including health care.

The SOA allows decoupling of ICT applications into the logical units of business functionality, namely the components that can be configured and composed to create new or changed business function with minimal impact on the rest of the system. These components offer their functionality through their interfaces and are often referred to as *services* in the SOA speak. Web Services are one special case of SOA.

However, the SOA paradigm has also raised a number of questions regarding the meaning of the term ‘service’, in particular for the business stakeholders used to thinking in terms of economic or business perspectives. If the concept of service is to be used by both the IT and business domain experts, then there needs to be a clear separation of concerns pertinent to this concept.

This paper provides such a separation, by considering not only the computational and engineering aspects of service, as in the SOA approaches, but also their enterprise aspects. Both aspects can be regarded as a specialisation of a generic concept of service. The value of this broader scope is in the capability to provide a unified view of service, so that both business and IT stakeholders establish a common foundation for communicating among themselves. This also ensures architectural alignment between business and IT aspects of an overall enterprise architecture of a system in broader sense.

We discuss the ideas in the paper by considering the pervasiveness of the service concept in the health domain. This is a highly complex domain, involving many different actors that must effectively collaborate in a service delivery while increasingly relying on the IT capabilities to do so. The service concept in the health domain needs to cover activities of (and policies to apply to) health professionals, some of which are increasingly relying on the underlying IT services.

The next section provides a background discussion on the generic considerations for the concept of services,

illustrated with examples from health domain. Section 3 considers the concept of service from various architectural perspectives of an overall enterprise architecture. Section 4 proposes an ODP enterprise viewpoint on service. This is followed by an ODP computational viewpoint of service, which is then mapped onto the SOA and Event-Driven-Architecture (EDA) styles. Section 7 summarises our approach and outlines direction for our future work.

2. Service: generic aspects and health examples

This section outlines key aspects of the service concept, in particular from the economic, legal and business perspectives and reflects on their corresponding characteristics in the health domain.

In general, a service can be defined as 'something done to benefit others'. Service provision is then a process that creates benefits to consumers by facilitating a change in consumers, a change in their physical possessions, or a change in their intangible assets [1].

In the health domain, the benefits to consumers are the improvement of their medical condition, their health and their well-being, through the delivery of health-care services. These are provided by various specialist providers, such as doctors, nurses, and allied health professionals, and usually through coordinated interactions between these professionals, e.g. in the context of the "continuity of care" for individual consumers, or through care packages. The delivery of such a combined care package can be quite complex and often requires synchronised and timely service delivery from all service providers involved. Finally, in the health domain the benefits are increasingly covering a broader range of services for the population in general, including for example preventative care and community services in addition to the acute service delivery. This broader spectrum of service delivery for all citizens during their life time is often referred to as 'care continuum'.

2.1 Economic aspects

When considered from an *economic* perspective, service is an economic activity through which benefits to the consumer are delivered, in exchange for a payment of some form to those who provide services.

In the health domain, service provision also has an economic aspect because of the cost of service delivery, arising from the labour cost or from the utilisation of necessary resources such as medical equipment and medications. We note that there are various economic models for covering costs associated with service provision ranging from full public subsidies to fully private payment with most of these being a blend of public and private models. This depends on the national health system in question, which may include state jurisdictions.

2.2 Legal aspects

In the context of *legal* frameworks that govern service provision in most economies, including international trade, a promise to deliver service, or service offer, implies certain level of guarantees from the service providers. The guarantees can be in terms of what functionality is provided and also in terms of non-functional variables often referred to as Quality of Service (QoS). These guarantees can be substantiated by various mechanisms such as certification requirements and reputation characteristics of providers. The guarantees can even be taken as given, based on the direct trust in service providers from consumers' previous experience. The legal frameworks also provide rules and regulations for the measures to be taken in case these guarantees are not met or are violated.

In the case of health care service providers, part of the guarantees are derived from the rules that set prerequisite criteria for permitting delivery of health services, e.g. requirements for passing certification tests typically set by governments or medical boards. An example of such an organisation in Australia is the Australian College of Health Services Executives [9]. These bodies also set rules for the actions to be taken in cases where there is inadequate service delivery. There are various forms of penalties that apply to service providers ranging from financial penalties to the revocation of their licences or certifications for providing services.

2.3 Business aspects

In the world of *business*, services are usually described in terms of service offers, which when accepted by consumers, form the corresponding agreements or a legally binding contract. Contracts can be regarded as a special way of defining guarantees to consumers for the behaviour of service providers. A business contract is constrained by the legal framework of the jurisdiction in which it is made. The contract will specify what the service provider has agreed to deliver and what the service consumer has agreed to accept, i.e. the consideration to be given in exchange for the use of service.

In health, contracts also exist as a way of governing interactions between various actors such as between private and public providers, medical and non-medical providers and so on. They are also subject to standard business contracts law. However, the 'contracts' that apply to the interactions between health care providers are consumers have a somewhat different character. The main concerns here are the *policies* that specify the responsibilities of

health providers and which are motivated by the safety concerns. Policies can also state the rights that consumers have in case of inadequate service delivery by health professionals. Note that consumers also have certain responsibilities, for example to comply with the therapies prescribed. As opposed to business contract law coverage, the law that applies to the consumer and health providers is mostly common law.

Furthermore, in business, many standard agreement types have been developed over time, reflecting historical patterns of interactions. Examples are real-estate contracts, financial contracts, construction contracts and service level agreements (SLAs).

In the health domain, there are standard agreements too. Some involve commercial providers and others are more within government scope, such as agreements between non-government organisations and health service providers, e.g. Memorandums of Understandings.

2.4 Generic aspects: summary

In summary, in addition to the functional aspects to be provided by the service (i.e. 'something to be done'), there are other characteristics of services such as:

- the benefits or value delivered to consumers and the associated measures; in other words, these are outcomes, which in the health domain are an improved health for individual consumers and for the population in general
- policies that apply to those who deliver services and possibly to those who are recipients of services; for example in the health domain the former are quality of care, accountability, delegation and privacy while the latter are patients' responsibility to follow treatment guidelines. Note that a subset of policies come into effect in case of violation of the primary policies, e.g. the measures that need to be taken when there is an unsatisfactory treatment of patients
- costs associated with service delivery

3. Service concept and enterprise architecture

The term 'service' has also been used in the context of IT systems, where in the past it has been primarily used to describe functionality provided by a software component, application or the whole system. In some cases, the term is also linked with the cost aspects of service delivery. However, in IT systems services have rarely addressed their social, legal or economic aspects. These aspects are critical for the health domain and, from the point of view of an enterprise architecture, they belong to the business architecture aspects of an overall architecture.

Considering the importance of enterprise architecture for health systems, as evidenced by many new initiatives in the health domain, such as in the US Federal Enterprise Architecture [10], it is important to take into account a broader definition and model for health care service. This is needed to describe other characteristics of services, such as the policies that apply to health providers and their implication on the underlying IT applications that support coordinated health service delivery.

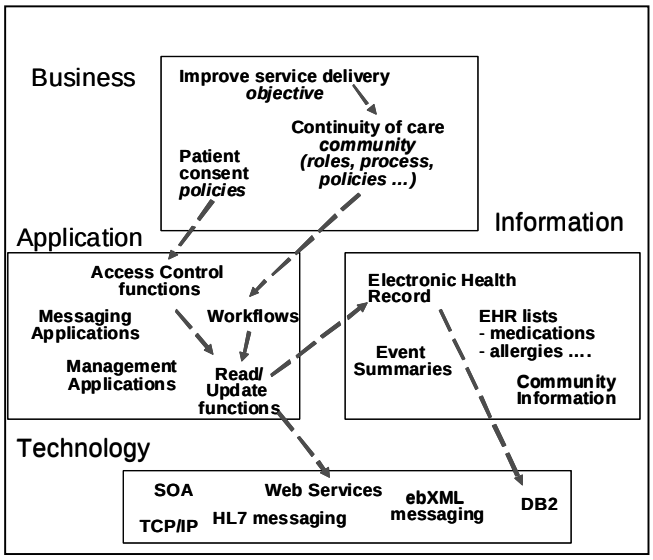

Figure 1: Different service aspects in an Enterprise Architecture

We illustrate this through a simplified example showing some of the modelling concepts in business, information, application and technology architecture of an overall enterprise architecture, Fig.1 Consider patient policy constraints regarding access to and sharing of information in their electronic health records (EHR). These are important aspects for the EHR service provision and they need to be expressed as part of a business architecture on an overall enterprise architecture. This business constraint needs to be propagated to the definition of the access control rules for the read/update functions for the EHR, in the application architecture, as shown with the arrows linking these modelling concepts. This functionality is then specified as part of the technology architecture, for example using Web Services technology. The example also shows links to the information architecture which specifies structure of EHR, and which uses a specific database technology, DB2.

Another example is the specification of various *roles* and *processes* in the continuity of care *community* (specified in the business architecture) that can be implemented using a

workflow engine which in turn may need to use read/update function to access the EHR system.

In the remainder of this paper, we propose a broader framework for defining services, by adopting the ISO ODP architectural framework. According to this framework, a system or a concept can be defined from various architecture viewpoints reflecting specific concerns of the relevant stakeholders. These viewpoints are enterprise, computational, information, engineering and technology. In this paper we consider only enterprise and computational viewpoints.

In the ODP standard the concept of service is mentioned in the context of foundational concepts and is defined as 'a particular abstraction of behaviour expressing guarantees offered by a service provider' [2]. To a significant extent, this statement reflects the fundamental aspects of service as per their economic, legal and business interpretation discussed above. We will use this definition as a starting point for further refinements. Specifically, we will refine this definition from each the ODP enterprise and computational viewpoints allowing us to support modelling of services from either the IT or business perspectives. Our intention is to provide a clear separation of concerns associated with service concept while ensuring that its fundamental properties are preserved.

Our aim is to ensure that service becomes a more explicit modelling concept in the ODP specification, in a similar way as the concept of object has the enterprise, information, computation and engineering viewpoints. This can be also seen as a contribution to current revisions within the ODP standards, with the aim to promote a more unified view on service.

We begin by considering service from the enterprise viewpoint in next section, followed by computational viewpoints, presented in following sections.

4. Enterprise viewpoint considerations

The ODP enterprise viewpoint is about the scope, policies to apply and objectives of the systems to be built. Thus the ODP enterprise specification needs to define the organisational, policy and legal constraints for the IT systems to be built. Accordingly, the concept of service in the enterprise specification deals with a broad set of issues that include human and social behaviour in the system. Part of this behaviour applies to the behaviour (i.e. functionality) of an underlying IT applications and systems that are used by the human actors. This was discussed in the context of health domain example in the previous section.

From the ODP enterprise viewpoint, and by refining the service as a foundation concept, stated above, service is an abstraction of *behaviour* of a *service provider* in terms of economic and legal activities through which the provider offers services with the corresponding *guarantees*. This definition includes a number of modelling concepts which we depict using the corresponding meta-classes in the meta-model in Fig. 2. These are the Service, the Provider and the Guarantee meta-classes and the relationships between the Provider and the Service meta-classes, and between the Service and the Guarantees meta-classes. The Service meta-class is an abstract meta-class, as there are many specialised types of services. In the health domain, these can be classified in many different ways [1]. Note that due to space limitation this is not a fully specified meta-model. Rather, it only includes key meta-classes and their relationship, while omitting other detail such as attributes of the meta-classes and full details of their relationships, e.g. their full

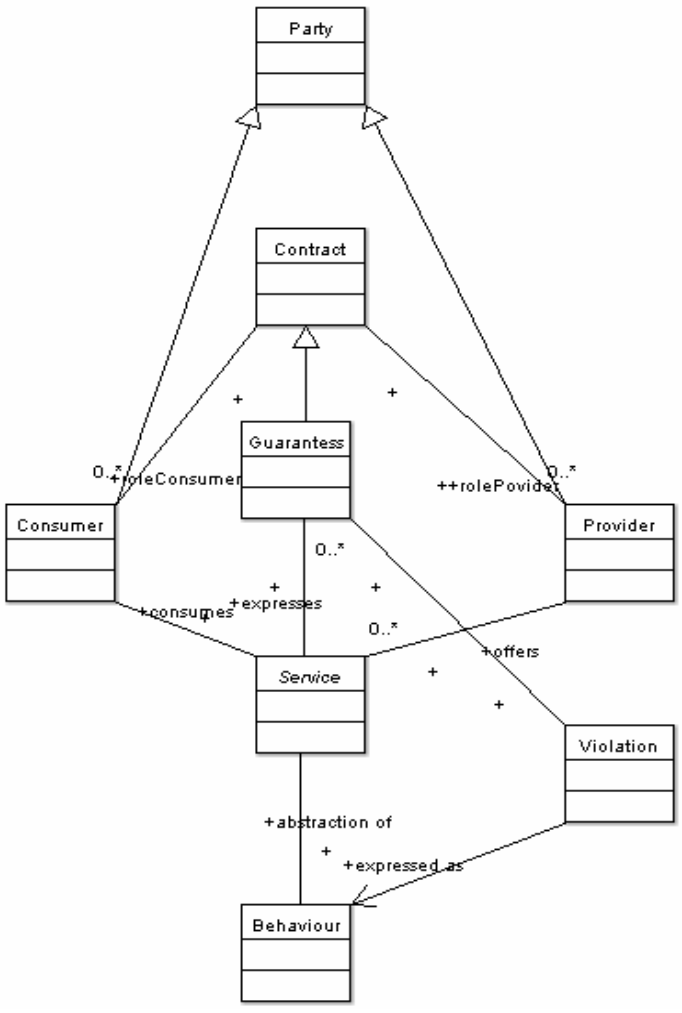

Figure 2: Service - ODP enterprise viewpoint

cardinality and navigability properties.

The service provides some value to potential *consumers* according to the guarantees associated with the service offer, and this is shown as the relationship between the Service and the Consumer meta-classes.

The guarantees that are offered by the service provider for the provision of service can be regarded as a partially filled business *contract* (shown as specialisation relationship between the Contract and the Guarantee meta-classes). Such a contract will be instantiated at the point when a consumer of service accepts the service offer of the provider.

Typically, the guarantees specify the capabilities and constraints of a service provider and can be in terms of functional or non-functional properties. It is also possible that, through a form of negotiation, the original guarantees are updated to better reflect needs of consumers. Notice that the Service Provider and Consumer meta-classes have their corresponding roles in the respective contract (shown through the relationships between the Contract and Consumer and Provider classes).

For a contract to be legally valid, and thus the guarantees to be accompanied by the corresponding reparation actions, each of the roles in the contract, i.e. the provider and the consumer must have the legal capacity properties. This is one of the elements of legal contract validity [4]. The ODP enterprise language includes several accountability concepts and we use the concept of *party*, defined as 'an enterprise object modelling natural person or any other entity to have some of the rights, powers and duties of a natural person' [3]. This is shown as two specialisation relationships between party and consumer and provider meta-classes.

Finally, we include a meta-class that describes violation measures that need to be applied in the case of failure to meet service guarantees. These measures could be defined in terms of the policies that take effect in response to violation events. One possible approach to formally specifying and implementing violations of guarantees stated in business contracts is given in [8].

5. ODP computational viewpoint considerations

From the ODP computational perspective a service is an abstraction of *behaviour* of a *server object* in terms of a function that it makes available to other objects. These guarantees include computational activities such as conveyance of information back to the *client object*. This transfer of information is initiated in response to the client's invocation, i.e. their request for a function to be performed by the server object.

It is important to note that the above definition encompasses both the synchronous (e.g. RPC) and asynchronous (e.g. message-oriented or publish-subscribe) styles of interactions. In the latter case, the server object is called producer and the client object is called consumer.

Therefore, while the concept of service in the ODP enterprise viewpoint required some additional modelling concepts in the ODP enterprise language, e.g. Guarantees, Provider and Consumer (as discussed in section 4), the existing ODP computational concepts are sufficient to model service in the computational viewpoint.

In the following we show how these ODP abstract computational concepts can be mapped onto the specific architecture styles such as SOA and event-driven architectures (EDA). These mappings, coupled with the relationships between the ODP enterprise and computational viewpoints (which will be briefly discussed first), enable us to provide more direct linkages between business and application architectures.

5.1 Relationships to enterprise specification

We describe several relationships between the ODP enterprise and computational modelling concepts pertinent to services only. These relationships need to be accompanied with the relationships between other enterprise and computational modelling concepts, some of which are described in [3].

The function that the (computational) server object makes available to others implements guarantees of the service provider stated in the enterprise viewpoint. The (computational) client object implements some of the functionality of consumers specified in the enterprise viewpoint.

The basic behaviour specified in terms of computational behaviour above needs to be augmented with the behaviour expressions specifying the policies that apply to the service provider and consumer enterprise objects. These additional behaviour specifications can be implemented using a special kind of computational objects, i.e. policy objects that implement obligation, permission or prohibition expressions. A detail description of policy objects and their behaviour is presented in our earlier work [6].

The use of policy specifications can be either to directly prevent an inadequate service delivery or as part of business activity monitoring. In the latter case certain reparation measures could be applied and these can be implemented through the appropriate set of computational objects. A more detailed description of various enforcement approaches are given in [7, 8].

5.2 SOA and mapping to ODP

The principles of SOA have been around for long time, including the early development of open distributed systems and the related standards such as ISO ODP [2,3] and OMG standards. The SOA principles became prominent in recent years with the emergence of the Web Service technologies and specifications. Perhaps the novel features are:

- the more explicit focus on the business logic to be supported, including more direct access to human users via Web-based interfaces
- the capability to link services through various choreography models and
- a more documented-oriented messages (typically defined in XML) in communication between objects.

The use of SOA in the health domain can be seen as an enabler towards replacing monolithic and hard-to-change health applications of the past with the components that more directly reflect the needs of various health professionals. This would enable clearer definition of services of each of the health providers and linking them into processes that reflect health-care specific activities. This applies to administrative and procurement activities, but also linking of various service providers in the context of 'continuity of care' delivery.

In the following we list key SOA principles, and provide a simplified mapping to the ODP computational architecture framework.

Loose coupling – meaning that services are defined solely in terms of the functionality they provide, without a strong (or with a small and well-known) dependency on other components; in ODP, this decoupling is achieved through the concept of object, which can expose its functionality through one or more interfaces;

Interoperability – meaning the ability of services implemented using different platforms and languages to communicate with each other; in ODP this is achieved through a protocol independent computational interface specification

Composition – meaning a capability for services to be assembled into applications in various ways including the ways which were not anticipated at the time of service definitions. This allows developing more complex business logic and adding new functionality as new business needs require. In ODP there are various mechanisms for composing objects, in particular the computational binding object which allows connecting computational interfaces of various objects. The binding object is an abstract mechanism for linking various objects and some specific styles of binding are orchestration and choreography. An approach based on the refinement of binding object targeting such binding styles are discussed in more detail in [7].

5.3 Event-driven architecture and ODP mapping

Event-driven architectures (EDA) are based on the capabilities of components to produce events and of other components to consume events. This is often referred to as publish-subscribe model of interactions. So, rather than through a usual synchronous communication style typical of SOA approaches, components interact by producing and consuming events. This assumes the existence of a message oriented middleware (referred to as message queue) which provides persistent storage for events, after they are produced and consumed by others. The capabilities of EDA, also enable the specification of applications in terms of events and event relationships, i.e. the event patterns, as proposed in [5]. A similar approach was also taken in [6], for the purpose of real-time monitoring of business activities associated with enterprise contract management.

The combined capabilities of SOA and EDA offer many options for developing health applications in an incremental manner, while leveraging the existing applications where possible. For some applications in which there is no need for real-time access to information, the SOA may suffice. For others, the choice may be based solely on the EDA principles or more likely it would be the combination of SOA and EDA solutions.

The use of EDAs and event-pattern specifications in the health domain could be exploited to support real-time checking of the adequacy of service delivery by health providers, or the implementation of medical treatment by the patients.

In terms of the mapping to the ODP computational model, the concept of event in the EDA corresponds to the ODP concept of *signal*. This is defined as an atomic shared action resulting in an one-way communication from an initiating object to responding object [11]. As operations in the ODP computational model can be defined in terms of signals, so the messages communicated between components in an EDA can be defined in terms of events.

6. Conclusions and Future Work

This paper provides an input towards a generic service modelling covering both the business and IT aspects of services. We use the ISO ODP standard as a framework for discussing the ideas. We presented an initial proposal for extending ODP standards to give more prominence to the

concept of service. We then discussed relationships between ODP enterprise and computational modelling concepts regarding services. When further consolidated, these relationships will allow generic mappings between different viewpoints on service.

Further, each of the abstract set of ODP viewpoints can be made more specific. On the technology side we have demonstrated how the ODP computational modelling concepts could be mapped to more concrete architectural styles such as SOA and EDA. These are still abstract models and the respective concrete models would be Web Services architectures and JMS. Our approach is illustrated in Fig. 3. The mappings 2, 3 and 4 show generic mappings from the enterprise language concepts to the underlying technology implementations. For example, the concept of policy in the enterprise language can be mapped onto the policy objects in the computational viewpoint. These can be implemented using both the SOA and EA solutions, such as WSDL and WS-eventing. Finally, the health specific concepts could be used to refine and extend relevant ODP concepts. This mapping (1), with the mappings 2, 3 and 4, thus provides a link from the enterprise language to the underlying implementation options, such as the specific SOA and EDA solutions.

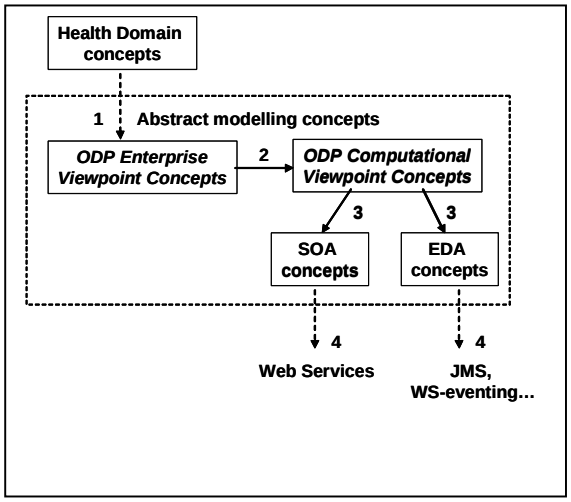

Figure 3: From health domain concepts to concrete technology solutions

The paper was motivated by a recent collaboration between DSTC and Queensland Health regarding the establishment of an enterprise architecture framework for this health organisation. However, the results pertinent to the health domain are not specific to Queensland Health and could be applied to many health jurisdictions.

In future we plan to continue this study by developing more comprehensive business language concepts for the health domain and test this on a number existing and future initiatives within Queensland and Australian health organisations. The concept of service is one, but important modelling concept in this language. We also plan to provide a more comprehensive mapping between various ODP viewpoints, by following the approach we used in [7].

We believe that the maturity of ODP standards (directly, or through other standards that were influenced by ODP), provide a sound basis for developing an enterprise architecture framework for health domain. The precise architecture of ODP, supported by the emerging and future tools, including model-driven engineering, is a promising candidate for developing a business-driven, and sustainable enterprise architecture for the existing and future health applications.

7. Acknowledgements

The work reported in this paper has been funded in part by the Co-operative Research Centre for Enterprise Distributed Systems Technology (DSTC) through the Australian Federal Government's CRC Programme (Department of Education, Science and Technology). The author would also like to thank Walter Robb, from Queensland Health for providing valuable comments to this paper and to Keith Duddy for his review of an earlier version of the paper.

8. References

[1] http://en.wikipedia.org/wiki/Service (accessed June 2005)

[2] ISO/IEC IS 10746-3, Open Distributed Processing Reference Model, Part 2, Foundations, ISO 1994

[3] ISO\IEC IS 15414, Open Distributed Processing-Enterprise Language, 2002.

[4] Z. Milosevic. *Enterprise Aspects of Open Distributed Systems*. PhD thesis, Computer Science Dept. The University of Queensland, October 1995.

[5] D. Luckham, *The Power of Events*, Addison-Wesley, 2002

[6] P. Linington, Z. Milosevic, J. Cole, S. Gibson, S. Kulkarni, S. Neal, A unified behavioural model and a contract for extended enterprise, Data Knowledge and Engineering Journal, Elsevier Science.

[7] A. Berry, Z. Milosevic, *Extending Choreography with Business Contract Constraints*, International Journal of Cooperative Information Systems, Vol. 14, Nos. 2 & 3 (2005), p. 131-179, World Scientific Publishing Company.

[8] G. Governatori, Z. Milosevic, Dealing with contract violations: formalism and a domain specific language, Proc.

the 9th IEEE EDOC conference, Holland, Sept. 2005, to appear.

[9] http://www.achse.org.au/

[10] http://www.whitehouse.gov/omb/egov/a-3-2-services-health.html

Services, contracts, policies and eCommunities – Relationship to ODP framework

Lea Kutvonen and Janne Metso

Department of Computer Science, University of Helsinki, Finland
E-mail: Lea.Kutvonen@cs.Helsinki.FI, Janne.Metso@cs.Helsinki.FI

Abstract

Agility for inter-enterprise collaborations requires development of interoperability and B2B middleware services. In the Pilarcos and web-Pilarcos projects, such middleware solutions have been researched and developed, in close relationship to the ODP reference model and complementary standards. Although it has been claimed that the ODP reference model has not reached its audience, most of the topical trends – service oriented computing (SOA, SOC), inter-enterprise business process management, virtual organizations management, and Web Services – reflect the same foundations. This paper discusses the issues around the concepts of services, eCommunities, and contracts as they are visible in the web-Pilarcos architecture. The contribution is directed two ways: for enhancing the concept related to service within the ODP framework, and for showing how the web-Pilarcos architecture applies the distinct concept of service type for gaining improved interoperability control over what is available for example with Web Services.

1 Introduction

The web-Pilarcos architecture provides middleware level services for establishing and controlling inter-enterprise collaborations (virtual organizations), also called eCommunities. The eCommunities involve a set of autonomous enterprise-application level services, and are controlled by eCommunity contracts.

The eCommunity contracts capture metainformation about the community structure (roles and responsibilities of the participants, behaviour in terms of interactions between the roles) in the form of an agreed business network model, agreed policies restricting that model behaviour, and information about the current participants (technical such as access information, and business oriented such as cost of service, trustworthiness of the partner). The eCommunity contract also provides for controlled methods for renegotiating the contract and making changes to partnerships, policies, technical details, etc. The contracts capture information from all ODP reference model viewpoints, ranging from enterprise (and business or legal) concerns to engineering (and technology) aspects.

The web-Pilarcos middleware [14] applies many of the topical patterns for interoperable systems: SOA (service oriented architecture) [19] and Web Services [1] use similar separation of service offers as announcements of available services, and dynamic discovery of partners into compositions of complex services. The architecture is also concerned with operational time interoperability monitoring, and on mechanisms allowing flexible, community-wide resolution of breaches.

This paper brings forth the concepts related to services, service offers, and eCommunity contracts, continuing the discussion [12] of applying ODP concepts to inter-enterprise collaboration. The discussion hopefully enlightens the potential expansions on these concepts in the ODP reference model [3–6]. In addition, the required facilities for supporting these concepts are outlined, showing some omissions in Web Services arena. The paper is structured as follows. Section 2 outlines the web-Pilarcos architecture and services. The essential concepts of service, service offer, and eContract are further discussed in Sections 3 and 4. As the architecture can be criticized for too high costs, Section 5 is included to show some practical measurements on the eCommunity establishment phase.

2 B2B interoperability middleware

The web-Pilarcos architecture is designed to compose and govern composition of services provided by autonomous enterprises. The architecture is federated (in contrast to unified or integrated), assuming that services are developed independently and their interfaces and properties are compared during collabora-

tion establishment and monitored for conformance during the operational time of the collaboration. (Unified model would make an assumption that the services would be produced using a shared collaboration model.) Figure 1 illustrates the setup. The B2B middleware provides business-applications with concepts and practical infrastructure-level services for eCommunity management through agents dealing with contracts, and transparently ensuring interoperability using both statical and dynamic techniques. The architecture does not provide a shared execution platform or enactment of the business network models. Instead, the business applications themselves are expected to include a technique forwarding their local workflows as triggered either by peer requests or by internal triggers. A set of metalevel protocols between middleware level agents for the eCommunity management tasks is completely separate from the application protocols.

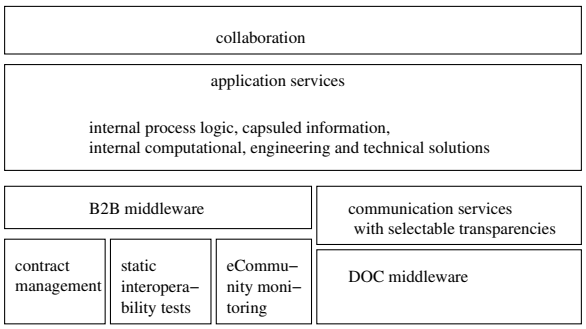

Figure 1. Architecture overview.

The issues to be addressed are interoperability between services and management of the lifecycle of the collaboration. The lifecycle management involves

- population of the eCommunity in such a way that the participating services are interoperable as discussed below, and a negotiation cycle between participants to agree or refine the suggested eCommunity properties; the population process selects from a service offer repository a suitable offer for each role in the business network model suggested by the initiator (if the business network model is not appropriate, the process fails);
- establishment of the eCommunity so that all participants are technically prepared for providing services and have done all necessary contract management chores;
- monitoring of potential breaches (nonconformance to the behaviour specified by the contract) in the interactions between participating services;
- monitoring the global, coarse-grain progression of the collaborative work in the eCommunity;
- reacting by renegotiations and contract changes for change requests initiated by involved parties;
- termination of the eCommunity either by the completion of the collaboration goal, by breach resolution, or timely termination of the contract.

In the management of eCommunities, interoperability is a prominent issue [11, 20]. Interoperability, or capability to collaborate, means effective capability of mutual communication of information, proposals and commitments, requests and results. Interoperability covers technical, semantic, and pragmatic interoperability. Technical interoperability means that messages can be transported from one participant to another. Semantic interoperability means that the message content becomes understood in the same way by the senders and the receivers. This may require transformations of information representation or messaging sequences. Finally, the pragmatic interoperability captures the willingness of partners for the actions necessary for the collaboration. The willingness to participate involves both capability of performing a requested action, and policies dictating whether the potential action is preferable for the enterprise to be involved in. In the pragmatic view, process-awareness in terms of collaborative business process model is needed, augmented with nonfunctional aspects, some of which are related to business policies.

At the pragmatic level, issues on the business strategies, values, and rules become visible. Partially these are reflected by using business network models as a founding element in the eCommunity contracts [12]. In addition, policies and properties embedded in to the eCommunity contracts and service offers are used for carrying this kind of information. Technically, most of those features appear as plain name-value pairs, but development of the carried semantics requires large scale ontology and metrics development at international consortia appropriate for each industrial area separately. This theme is further discussed in Section 4.

3 Services and service types

We assume that business-application services are independently implemented and deployed on the enterprises' computing system. The implementations can be generated using MDA style tools, using specifications of collaborations as a starting point. However, existing legacy software can be used equally.

The available implementations establish the practical capabilities for a service in an enterprise. The

enterprise can well support multiple, slightly different implementations for the same kind of service. Reasons for this multiplicity can be strategic (different business rules embedded), or technical (different platform and communication facilities used), or evolutionary (old and new software used side by side till the old can be made obsolete as clients have become able to interoperate with the optional ones). Thus, the *service (implementation)* refers to the computational composition of application software that provides a complete business service, or supports a step in such.

The ODP reference model differentiates between types and templates for objects [3, cl. 9.7 and 9.11]. This separation is heavily used in our approach. The *service templates* are only used within an administrative domain responsible of executing a service. An *administrative domain* is a term related to federations [4, cl. 10.3, 5.1.1, and 5.1.2] and we use it for denoting a technologically consistent system within an enterprise. Outside that administrative domain, the service template has no relevance, but instead, a less restrictive concept of *service type* is of importance.

The ODP reference model indicates two methods by which objects can become known in a system, namely *instantiation* and *introduction* [3, cl. 9.13 and 9.16]. Instantiation is needed within the administrative domain responsible of running a service, but at other domains, the presence of that service is created though introduction. The introduction process is required to reveal the type of object in question, i.e, associate the object with a useful predicate it matches.

In an inter-enterprise environment, or a SOA environment, the service offer repositories (UDDI [17], ODP / OMG trader [5]) reflect the introduction process. All service offer repositories expect exporters of offers to define the identity of the service, its service type reference, and access information. The service type denotations vary, as well as the level of control in their definition. In the (web-)Pilarcos case, the service offers are expected to include in addition attributes as defined by the service type and the middleware itself (in relation to environment contract, see below).

In the web-Pilarcos architecture service types are stored into a type repository, and only those previously defined type descriptions can be associated with service offers. The service type descriptions can be published either as part of a design process or independently, by various enterprises in the global network, and have to be verified before acceptance to the repository [10, 13].

Service types are abstract descriptions of business service functionality and they define functional and non-functional properties for a class of business services. Functional part of a service type comprises of an interface signature, an interface protocol which describes the service behaviour and additionally semantic annotations for exchanged documents (messages). Engineering level information, such as binding of a service instance into a specific communication protocol or address, is not part of the service type. The non-functional properties of a service type describe issues on business level concerns, QoS requirements, and policies. The non-functional aspects are given as service attributes (name-value pairs), where each element denotes a semantic concept from a shared ontology.

Technically, the service type defines

- the service interface signatures;
- associated to the interfaces, restrictions on the ordering of invocations (a loose way of expressing a protocol, only giving restrictions on the necessary ordering relationships)
- associated with the interfaces, information contents of the messages; and
- set of attributes associated to the service, labeled either optional or mandatory.

Thus the service type repository is actually defining an application-area specific, service-type based ontology for information about the business and legal aspects of the service. The attributes can reflect the cost of service, availability, expectations on the business networks involved, and legislation to be used for evaluating the service and managing breaches. As some of the attributes are optional, this gives flexibility in making service offers with different kind of business networks in mind. Different collaborations will depend on different attributes.

For example, Web Services technologies are still missing type repository type of services. The benefit that is provided lies in the trustworthiness of the type definition. For a repository, some quality requirements can be placed: static verification of the models, persistent and continuous availability of items, correctness of assertions on relationships between types, etc.

Besides the attributes, the service offers are expected to contain attributes that form an offer of environment contract [10]. The *environment contract* [3, cl. 11.2.3 and 13.2] of ODP refers to commitments between an object and its environment, i.e., prerequisites under which the service in question can be provided. In the web-Pilarcos case, the environment contract part of the service offers include selection of binding type, channel type, and configuration parameters for the abstract communication layer depicted in Figure 1. The binding type defines what services the abstract communication layer is expected to provide and what role-related interfaces for communicating peers are provided. The

channel type denotes the architecture of the communication channel so that the middleware is able to configure it appropriately. In addition, requirements on shared computing or information repository resources can be placed.

4 eCommunities and business services

In the eCommunity establishment process, the business level commitment to the contract takes place, as well as the computational setup of the collaboration. There are two sources for regulating information involved: the business network model, and the service offers selected for the contract.

The business network model we use is a expansion of the *ODP enterprise description* [7, 8]. Several, functionally separate *community descriptions* are integrated by denoting which of the roles have to overlap for the network. Each functional community is defined in terms of roles comprising of a role name, service type required, and free form constraints on a) properties of the service offer to be selected to that role, and b) rules to be used for monitoring the service behaviour during the eCommunity operation. In the population process, the service type requirement and selection criteria are used for retrieving service offers. The environment contract and policy requirements in the service offers further cause interdependent requirements, thus making the matching process quite complicated. Although this process is at first sight very expensive, it stays within reason, due to the time-based and memory-space restrictions on the searches. For further evidence, Section 5 shows measurements done on a CCM platform.

At the level of eCommunity, the relationship between the business environment and the provided service becomes another aspect of the environment contract, thus forming another defining aspect of the service itself. As the participants in the eCommunity are in some extent dependent on each others' services, the commitments they make are made on the premises that others fulfill their commitments according to the contract too. Therefore, the concept of business service cannot be fully defined in isolation, but is dependent on the business network it is part of.

A *business view of the service* can be seen as a computational service, associated with environmental constraints from the providing enterprise's computing environment, its role in the business network in question, and the business and computing restrictions set by its peers in the business network.

Because business network models gain such an important role in interpretation of all service related concepts, the web-Pilarcos architecture provides a repository for these models. The benefits of the globally available set of verified models (with related service types in the type repository) arise from the facilities for reasoning from the process models, process-level interoperability support in the middleware, and guidance to the service markets.

Having captured all these aspects to the eCommunity contract, we still cannot claim that the correct business behaviour or legally correct interoperation is guaranteed. The models provided for interoperability checking are just models indicating the intention of the service provider. Even if the models were used to generate the software elements from those specifications, there would be uncertainties involved. Therefore, monitoring of the behaviour conformance against the model is essential. Depending on the level of distrust and acceptable performance penalties, the conformance requirements need to be selected appropriately. The monitoring aspects are presented in [15], so they are not discussed here further.

Mechanisms to enforce conformance to business needs and interoperable computing requirements must involve dynamic methods in addition to the traditional static analysis. One of the essential reasons for this is the need for capturing enterprise policies and business rules. Those can change any time, disregard of the eCommunity contracts in use – thus causing contradictions at operational time. Furthermore, the ontologies for eCommunity contracts may omit some of the locally important policies, and therefore be unaware of the contradiction.

We use the following terms. Enterprise policies are used to govern the resources and services of an enterprise. They may oblige, permit, or prohibit access to some service or information. Enterprise policies or eCommunity policies are used to modify the basic business process models; negotiation of a policy value is used to refine which alternative path through the model is in use. Business rules are declarative statements that define or constraint some aspect of business service behaviour. Business rules may define different pricing policies or service availability for different client categories. A business rule typically affects the non-functional aspects or information exchanged in a service interaction.

The above definition of business service did not explicitly mention the policies. However, policies are an integral part of the context of the business service, and is to be seen as part of the environmental constraints. Therefore, policies come as a time dependent and enterprise specific element into the picture.

Finally, a few words on another time dependent property. One of the very essential properties of a busi-

ness service is that of trust. In the TUBE project [23], we are extending the web-Pilarcos architecture with trust concepts. Decisions on trusting a business service is made first at the establishment of an eCommunity; business service trustworthiness is estimated by reputation information and local experience of that enterprise and that business service especially, and has to exceed a treshold dependent on assumed risks and importance of the eCommunity participation for that enterprise. As the correct behaviour of the peers in the eCommunity are not completely trusted even after this, the behaviour is monitored, and new experience information and new reputation information is generated by successes and breaches of conformance.

5 Cost of eCommunity establishment

While Pilarcos middleware provides facilities for dynamically forming eCommunities there is overhead cost from the use of the middleware services. This cost has been estimated by a series of performance measurements on the prototype application and infrastructure services. First, the overall overhead of Pilarcos middleware in terms of CPU load and effect on response times seen by a client was considered. Second, the main elements of the cost were factored out at the areas of business network population, eCommunity establishment and termination and adaptation of communication across the CCM and EJB platforms. Third, some scalability tests were run to see how the input rate of client requests affect the response times. A set of separate measurements were done on the population process which is critical for the feasibility and scalability of the middleware. The measurements were performed with a system that included all the Pilarcos infrastructure services and an application case. The application case does not include much of application-specific processing, but is there mainly to generate a meaningful mixture of requests to the middleware under study.

In concrete terms, the Pilarcos prototype consists of Pilarcos infrastructure service components and Tourist Information Service application components that take advantage of the Pilarcos services. The infrastructure services and most of the application components have been implemented on two new CORBA Component Model platforms, the Java-based OpenCCM [18] and the C++-based MicoCCM [16]. In addition, one of the application domains is built on Enterprise JavaBeans technology, running on the JBoss [9] application server.

The sample case study is built around an idea of a *Tourist Info* portal service, which provides traveller access to vertical tourist services like travel information, hotel bookings, and weather services. Neither the portal service nor the vertical services are free, but the traveller has some electronic payment instrument (e.g. credit card) available. Figure 2 shows the business entities and the communities formed by them. The tourist info community contains three domains for tourist info client, tourist info service, payment service and describes the business community related to providing the portal service. The hotel info community contains roles for hotel info client, hotel info service, and payment service and describes the business community related to providing the only implemented vertical service (hotel bookings). Payment service represents a trusted third party used to mediate the transfer of funds between the other entities in the community.

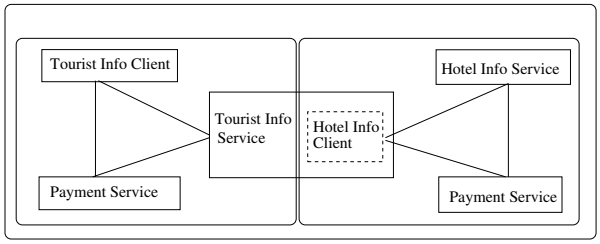

Figure 2. Business entities and communities.

The measurement environment consists of seven 1GHz Pentium III workstations with 512MB RAM memory, running CS Linux with kernel 2.4.18. Workstations were connected with closed 100 Mbps Ethernet. Different components of the Pilarcos prototype were distributed to the workstations. The components related to one domain were deployed in one machine with the exception that in Hotel service -domain CCM and EJB components were distributed to two separate machines. Background clients used two additional machines and were evenly distributed in them. Before measurements the system was warmed up with 3000 measurement rounds done by the client. The measurements themselves included 8000 rounds. Each benchmark used one measurement client and two to six background clients. In all measurements the IBM Java 1.4.1-platform was used, with throughput optimization flags. For the Pilarcos trader measurements, the client program was run on one workstation and the Pilarcos trader on another. Java-based ORBacus trader 2.0.0 [2] was run in compiled mode on the same workstation as the Pilarcos trader.

Looking at the measurement results, the cost of Pilarcos middleware use appears to be acceptable. As we consider a “round” from a user, the application runs the following steps:

- request population for a business network;
- tell the business network manager to create the community; this actually involves the population and creation of the second community in the application scenario;
- run a group of application scenario-specific operations and requests for the servers;
- end the session and ask the business network manager to terminate the communities.

The population, eCommunity establishment, and eCommunity termination phases can be seen in Figure 3. Under normal load conditions, the population process takes less than 25 milliseconds, eCommunity creation under 40 milliseconds (the measured 100 ms includes two eCommunity creation operations and a population operation), and eCommunity termination about 20 milliseconds (again, the measured numbers cover two terminate operations). Time consumption for getInformation and makeReservation operations are also noticeable in the measurements. This is caused by the fact that the execution process of these operations includes several service calls between components residing in different domains.

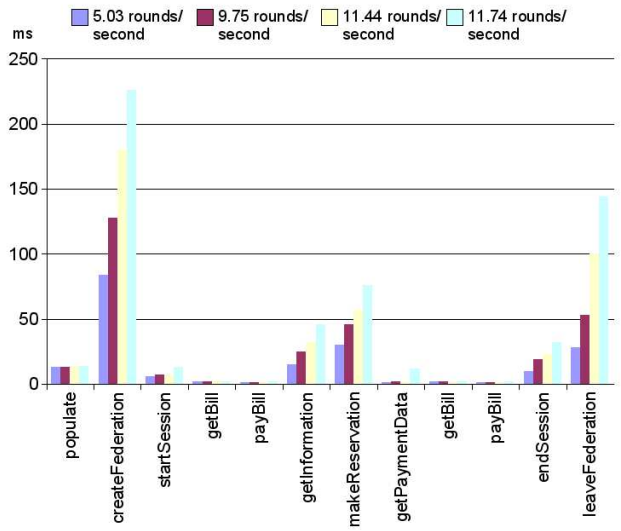

Figure 3. Response times for the client [22].

For the measurements presented in Figure 3 the number of request-making client computers were increased so that the input rate of requests reached a level where saturation on one of the computers was approaching. However, no unexpected behaviour on response times is seen here.

Figure 4 show that the Hotel Server domain is becoming a bottleneck in the system as the load increases. Within the Hotel Server domain the CCM platform running the infrastructure services is a bit more loaded than the EJB platform running the application components. This ratio depends, however, on the computational complexity of the applications.

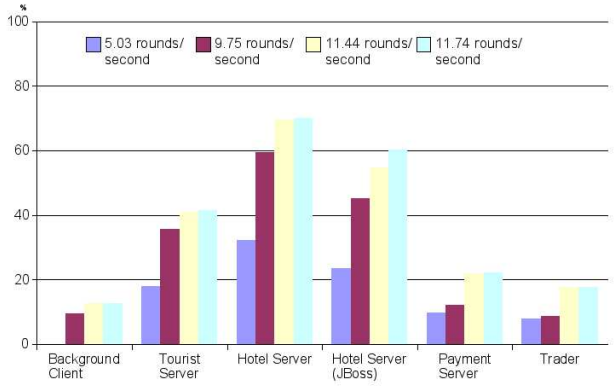

Figure 4. Processor load in servers [22].

The conclusions from this behaviour are twofold. First, additional measurements are needed in a more realistic and complex environment where the network latencies are longer (e.g. Internet). Second, in the current implementation the cost of adapter creation is considerably high for interactively used applications. A library-based solution has been planned but was not yet available for measurements.

To support interpretation of the above measurement results, Figure 5 shows the same application components running without the help of Pilarcos middleware. The configuration of application components is here fixed, and only OpenCCM is used as a platform. Thus, the flexibility of finding appropriate partners for the eCommunity is missed, and also heterogeneity support. All application components have been selected beforehand and their locations are resolved by a name server at the start-up time. The measurements show that the overhead load is mostly visible in the new operations offered by the Pilarcos middleware. An additional few milliseconds fixed cost is visible on all operations.

Although measurements on the prototype were much as expected, the platforms are not mature enough and caused scalability problems themselves, so scalability tests were not done as thoroughly as we wished. More thorough measurements could be done on the Pilarcos trader. In this paper we only bring up a few comments, and refer to [21, 22] for details.

The population process was able to provide one eCommunity proposal in an average of 22 milliseconds.

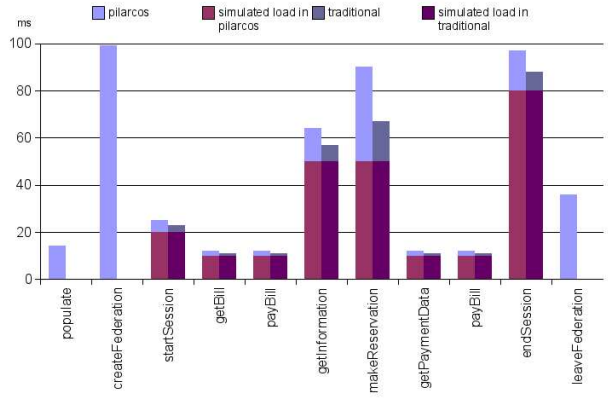

Figure 5. Time used in different phases seen by client with simulated load [22].

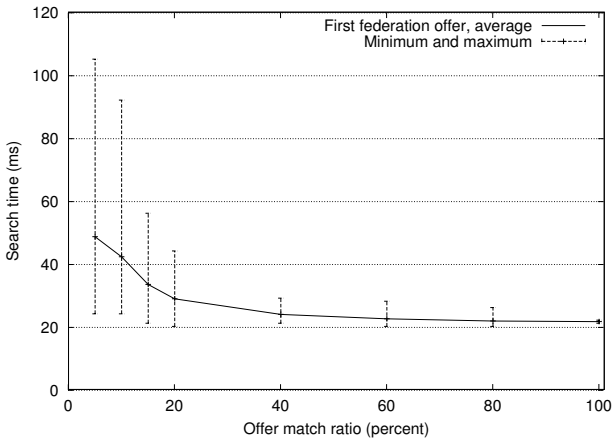

Figure 6. Effect of offer match ratio on search.

In addition, transferring the request and the result over CORBA took an average of 30 ms, raising the total to 52 ms. Most of the transferring time was result of marshalling delays. These times were not much affected by the number of service offers in trader database. With offers distributed evenly between the roles, no significant effect on the search time was seen with database sizes of up to 2550 offers, since the population algorithm never needs to search the entire offer database. Instead, the search time is directly proportional to the number of requested eCommunity proposals.

Search times grow in linear proportion to the number of policies in service offers. In the baseline case, a rather high estimat of 32 policies per offer was used.

The cost of marshalling is fairly high, because the eCommunity proposal data structure is designed for readability and flexibility, not speed. It contains several nested sequences and makes heavy use of the CORBA `Any` type, making marshalling very resource consuming. The marshalling delay could be significantly reduced by using known CORBA IDL optimization techniques. For the rest of the results, only the time spent in the search process is presented, since the marshalling delay is constant and predictable.

One of the potential problems were cases where there are very few acceptable service offers in the trader database. However, we found that both the average response time and its variation grow significantly with low match ratios, but are still tolerable even at a match ratio of 5 %. Based on these results, the practical usage area for the Pilarcos trader is with offer match ratios at and above 5 %. Figure 6 shows this effect.

Figure 7 shows the effect of the number of roles in the business network model. Again, the dependency is linear, as expected. Based on these results, large network models with up to ten roles would be practical; most probably, for such large networks, other aspects than the population process are more significant.

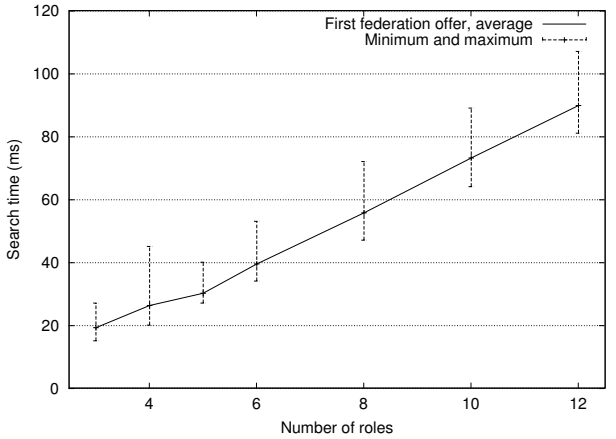

Figure 7. Effect of number of roles on search.

The above results were based on measurements on a standalone Pilarcos trader that includes its own offer database. We also tried the effect of using ORBacus trader for storing the service offers. The search times are nearly ten-fold compared to the standalone case, with significantly larger variation. These are results from the block-wise transfer of service offers from the ORBacus trader. When a CORBA trader is used as a back-end, a significant speedup could be achieved by collocating the Pilarcos trader and the CORBA trader in the same process. This trial is interesting because it

shows an effect that is expected when using federated (linked) CORBA traders.

6 Conclusion

This paper discusses concepts of computational and business services, and illustrates how business network models and service types bring in an ontological route for enforcing appropriate properties to be brought in to the eCommunity contracts. The properties spanning over all five ODP viewpoints capture regulations and restrictions as well from business strategical as from computing system perspective and bring them to the operational environment to reflect the state of the eCommunity.

Infrastructure services like global service type repositories and business network model repositories, and trust management facilities are the latest contributions to the arena of inter-enterprise interoperability and federated architectures.

A set of practical measurements on the prototype infrastructure services show, that the federated architecture model is not infeasible in practice, and also illustrates that the basic concepts for eCommunity management can be implemented on current distributed computing platforms.

Acknowledgment

The Pilarcos and web-Pilarcos projects at the University of Helsinki were funded by the National Technology Agency TEKES in Finland, VTT, Elisa, SysOpen, Nokia and Tellabs.

References

[1] B. Benatallah, O. Perrin, F. Rabhi, and C. Godart. *Web Service Computing: Overview and Directions*, chapter xx. Springer Verlag, 2004.

[2] IONA Technologies. *ORBacus Trader, version 2.0.0*, 2001. http://www.iona.com/products/orbacus/trader.htm.

[3] ISO/IEC JTC1. *Information Technology – Open Systems Interconnection, Data Management and Open Distributed Processing. Reference Model of Open Distributed Processing. Part 2: Foundations*, 1996. IS10746-2.

[4] ISO/IEC JTC1. *Information Technology – Open Systems Interconnection, Data Management and Open Distributed Processing. Reference Model of Open Distributed Processing. Part 3: Architecture*, 1996. IS10746-3.

[5] ISO/IEC JTC1. *IS13235 ODP Trading function.*, 1997.

[6] ISO/IEC JTC1. *Information Technology – Open Systems Interconnection, Data Management and Open Distributed Processing. Reference Model of Open Distributed Processing. ODP Type repository function*, 1999. IS14746.

[7] ISO/IEC JTC1. *Information Technology – Open Systems Interconnection, Data Management and Open Distributed Processing. ODP Enterprise Language, Ammendments*, 2003. PAM13235.

[8] ISO/IEC JTC1. *IS15414 ODP Enterprise Language*, 2003.

[9] JBoss web site. Jan. 2003. `http://www.jboss.org`.

[10] L. Kutvonen. *Trading services in open distributed environments*. PhD thesis, Department of Computer Science, University of Helsinki, 1998.

[11] L. Kutvonen. Automated management of interorganisational applciations. In *EDOC2002*, 2002.

[12] L. Kutvonen. Challenges for ODP-based infrastructure for managing dynamic B2B networks. In A. Vallecillo, P. Linington, and B. Wood, editors, *Workshop on ODP for Enterprise Computing (WODPEC 2004)*, pages 57–64, 2004.

[13] L. Kutvonen. Relaxed Service-type matching and Transformation management. In *Workshop on Enterprise Modelling and Ontologies for Interoperability, EMOI-INTEROP 2005*, June 2005.

[14] L. Kutvonen, T. Ruokolainen, J. Metso, and J. Haataja. Interoperability middleware for federated enterprise applications in web-Pilarcos. In *INTEROP-ESA'05*, 2005.

[15] J. Metso and L. Kutvonen. Managing Virtual Organizations with Contracts. In *COALA2005 workshop*, June 2005.

[16] MicoCCM web-site. Dec. 2001. http://www.fpx.de/MicoCCM.

[17] OASIS Consortium. *Universal Description, Discovery and Integration of Web Services (UDDI) 3*, 2002. http://www.oasis-open.org/committees/uddi-spec/tcspecs.shtmluddiv3.

[18] OpenCCM web-site (LIFL). Dec. 2001. http://www.lifl.fr/OpenCCM.

[19] M. P. Papazoglou and D. Georgakopoulos. Service oriented computing. *Commun. ACM*, Oct. 2003.

[20] T. Ruokolainen and L. Kutvonen. Interoperability in service-based communities. In *First International Workshop on Enterprise and Networked Enterprises Interoperability (ENEI2005), in conjuction with BPM2005*, 2005. To appear.

[21] M. Vahaaho. Trading with architecture models. Master's thesis, Department of Computer Science, University of Helsinki, Feb. 2003. In Finnish.

[22] M. Vahaaho, J.-P. Haataja, J. Metso, T. Suoranta, E. Silfver, and L. Kutvonen. Pilarcos prototype II. Technical Report C-2003-12, Department of Computer Science, University of Helsinki, Jan. 2003.

[23] L. Viljanen, S. Ruohomaa, and L. Kutvonen. The TuBE approach to trust management. In *Proceedings of the 3rd iTrust internal workshop*, 2004.