ON THE CONSTRUCTION OF DISTRIBUTED RM-ODP SPECIFICATIONS

Xavier Blanc(+)(*), Marie-Pierre Gervais(*) and Raymonde Le Delliou(+)
(*)Laboratoire d'Informatique de Paris 6-8 rue du Capitaine Scott F75015 PARIS
(+)EDF Research Division -1, av du Gnl De Gaulle F92141 CLAMART Cedex

Abstract LIP6, is association with EDF R& D[1] , proposes a framework that deals with the construction of heterogeneous and distributed specifications. This paper focuses on the part of work devoted to the distribution aspects of a specification, especially the distributed specifications consistency and management. It describes our approach to deal with these two aspects. Regarding the distributed specifications consistency, we advocate that the modeling language must enable the partitioning of the specification into several pieces and provide means to express the dependencies between them. For this reason, we make use of RM-ODP language as it includes some concepts that fit these requirements. However, RM-ODP is not prescriptive enough to be really helpful when elaborating distributed specifications. Thus we propose distributed specifications construction rules in identifying the needed concepts and in defining their usage rules. Concerning the distributed specifications management, we provide a so-called Distributed Specifications Management System (DSMS), that is an RM-ODP specifications repository built in conformance with the MOF and CORBA standards. Such a repository provides facilities to distribute specifications, to link pieces of specifications and to handle them.

Keywords: Distributed Specifications, Modelling, RM-ODP, MOF.

1. INTRODUCTION

Thanks to the object-oriented technologies and to the growing evolution in the semi-conductors, there are more and more distributed applications. For example, Information Systems of companies are now distributed, e-commerce is everywhere and we can send e-mails with our mobile phones. If these distributed applications provide helpful services to the end users, however they are more and more complex.

[1]This work is supported by grant # I26/B32617/IMA375 from EDF R& D

Modeling languages and methods can be useful to face this complexity. They lead to the elaboration of specifications that help for building, documenting and maintaining applications. In this paper, we are particularly interested in using RM-ODP standard for elaborating specifications. RM-ODP is an ISO standard that defines concepts and structuring rules for building specifications of distributed systems [1][2][4].

Considering that applications are built by big teams composed of designers, architects or developers, consequently specifications are now themselves distributed. It should be noted that specifications can also be distributed because some entities of the applications have been already specified and developed. In this case, building a specification consists in integrating existing with new ones.

Let us consider a project such as the building of an e-shop. The goal of this e-shop is to sell some products to its customers. For this, the e-shop must receive some orders, manage stocks and deliver the products. To increase its customers, the e-shop must do advertising and marketing.

The project team developing the e-shop is distributed over the USA, Europe and Japan and must build the specification. As the team is distributed, then the specification will be distributed. For example, the USA part of the team will specify the e-shop marketing while the Europe part of the team will specify the delivery service and the Japan part will specify the stocks.

Elaborating distributed specifications give new problems that the current specification techniques do not deal with. LIP6, is association with EDF R& D, addresses this issue and proposes a framework that deals with the construction of heterogeneous and distributed specifications [5]. This framework makes use of various standards from the distributed object computing and meta-modeling communities, such as RM-ODP from ISO, CORBA, MOF and XMI from OMG and XML from W3C. Considering the RM-ODP language as a reference formalism, the framework provides translation mechanisms between various formalisms enabling to build a new specification by composing heterogeneous existing pieces of specifications with new ones. Moreover, it deals with the taking into account that the various pieces to be composed can be distributed and stored in various locations.

This paper focuses on the part of work devoted to the distribution aspects of a specification, especially the distributed specifications consistency and management. As a specification can be composed of several pieces depending on each other, building one piece impacts the others. Thus maintaining the consistency of the whole specification becomes very difficult but is needed. Moreover, understanding the whole specification, which is distributed, requires that some relations be established between the various pieces stored. As a calculation can be performed by several distributed pieces of code providing the same result as

a single run-time, "linking" together pieces of specification stored in various locations must transparently provide the same view as a unique specification.

Current techniques do not provide mechanisms, which first enables the consistency maintenance and second the distributed specifications management. Actually, these problems respectively require an adapted modeling language and a Distributed Specifications Management System (DSMS). The modeling language must provide some mechanisms enabling the partitioning of the specification into several pieces and means to express the dependencies between them. The DSMS, also called repository, must provide facilities to distribute specifications, to link pieces of specifications and to handle them.

We present in this paper our approach to deal with these two aspects. Regarding the modeling language aspect, we make use of RM-ODP language because it includes concepts that fit the requirements presented above. However, RM-ODP is not prescriptive enough to be really helpful when elaborating distributed specifications. Thus we propose distributed specifications construction rules in identifying the needed concepts and defining their usage rules. Concerning the DSMS, we detail how we built an RM-ODP specifications repository enabling the management of these specifications. Our DSMS is developed in conformance with the MOF and CORBA standards [9] [10].

The paper is structured as follows. We first introduce main concepts of the RM-ODP standard we use. For sake of simplicity, we only focus on the Enterprise viewpoint[2] and illustrate the Enterprise specification construction rules we prescribe with the e-shop example. We then present our Distributed Specifications Management System by describing its construction based on the MOF and CORBA standards.

2. CONSTRUCTION OF AN RM-ODP ENTERPRISE SPECIFICATION

2.1. The RM-ODP Enterprise Viewpoint

RM-ODP (Reference Model for Open Distributed Processing) is an ISO standard that defines concepts and structuring rules for specifying open distributed systems [1][2][4]. In particular, RM-ODP defines the concept of "viewpoint" that deals with the separation of concerns needed to specify different facets of a system. Five viewpoints are then defined. The set of RM-ODP concepts is composed of concepts common to all the viewpoints and those that are specific for some viewpoints.

[2]Although there is no sequence or hierarchy between the RM-ODP viewpoints, it is generally considered that building RM-ODP specifications often starts by building the Enterprise one.

The Enterprise viewpoint focuses on the purpose, scope, and the policies that apply to a system [3]. The basic concepts of RM-ODP Enterprise language are object, role, community, objective, behavior and action. An *object* is a model of an entity, either an entity of the system to be specified or an entity of the system environment. Objects can be grouped to form a *community*. In that case, they exhibit the *behavior* needed to realize the *objective* of the community. By doing this, they fulfill roles of the community since a *role* identifies a behavior. This is a set of *actions* with constraints on when they appear. Actions can be *interactions* between several objects or internal actions.

More sophisticated concepts are defined in the Enterprise viewpoint, namely the *C-object and Interface role* concepts. They are particularly useful when building distributed specifications. A *C-Object* is an object that represents a community [3]. It enables to specify interactions between communities. As it represents a community, it is composed of several objects. Thus, when it performs an action, in fact one of its components really performs the action. To express this, RM-ODP defines the concept of *interface role* as an identifier of a behavior exhibited by a C-Object [3]. Thus the system and its environment can be modeled in several interacting communities.

2.2. Steps of the construction of an Enterprise specification

RM-ODP is not prescriptive on how to build an Enterprise specification, that's why we propose the following steps [13]:

1 Defining the objective;
2 Enumerating all the roles enabling to perform this objective;
3 Among the roles of the community, identifying those that can correspond to a distinct community. Assign then to these communities the roles that must be attached to them;

For each community:

4 Identifying the Enterprise objects fulfilling the roles of the community. If some objects are C-objects, describing the corresponding interface roles;
5 Describing the behavior of the community;
6 Describing the policies.

2.3. RM-ODP Enterprise specification of the e-shop

Applying these steps on the e-shop example, we elaborate the Enterprise specification of the e-shop. It should be noted that the specification we present here is a very simplified one, but sufficient to highlight our proposal. In partic-

ular, some steps (namely step 6) will not be considered as they are not relevant in the context of this paper.

The objective of the e-shop is to sell products to the customer. To accomplish this objective, we have identified the role "Customer", the role "Delivery Service" (DS) , the role "Stock", the role "Marketing Service" (MS) and the role "Order Taker".

The role "Customer" defines the behavior of the customer. The Customer can **create a new profile**, which is needed to order some products. The Customer can **order some products**. The Customer can **trace its order**, to know where it is. The Customer can **receive its order**. And finally, the Customer can **pay its order**.

The role "Delivery Service" (DS) defines the behavior of the delivery service. The DS can **package an order**. It can **deliver the package**. And finally, it can **provide the trace of an order**.

The role "Stock" defines the behavior of the stock. It can **buy new products**. And finally, it can **prepare the products for an order**.

The role "Marketing Service" (MS) defines the behavior of the marketing service. The MS can **send some advertising**. It can **perform some survey**. And finally, it can **define some special prices**.

The role "Order Taker" (OT) defines the behavior of the order taker. It can **create a new profile** for a new customer. It can **accept or deny a new order**. It can **deliver the order**. And finally, it can **cash the order**.

Among the actions listed for each role, some of them are interactions. For example, when a Customer creates a new profile, it interacts with the OT in order that this creates a new customer profile.

The figure 1 represents the e-shop community, with the interactions between these roles.

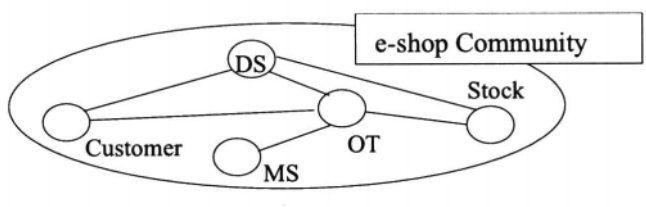

Figure 1. The e-shop community with its roles and their interactions

Applying the step 3 and according to our modeling choices, we consider that the DS role identifies a behavior that can be defined more precisely in a distinct community called the DS community. This defines the roles of the

"Trace Service" (TS), the "Stock Manager" (SM), the "Global Delivery Service" (GDS) and the "Local Delivery Service" (LDS).

The role "Trace Service" (TS) can receive information about an order to know where it is localized and **provide a trace of the order**.

The "Stock Manager" (SM) is responsible for finding the nearest stocks from the client that contain the products of an order.

The "Global Delivery Service" (GDS) **asks for some stocks to the SM** and **then delegates the delivery of the order** to a Local Delivery Service.

The "Local Delivery Service" (LDS) is responsible for **packaging the order** and **delivering the package** to the customer.

We can now, for each community, allocate roles to objects (step 4). Depending on modeling choices, various results can be obtained.

In the e-shop Community, for sake of simplicity, we choose to allocate one role to one object, except for the customer role where several objects can fulfill the role, one for each custom of the e-shop. Let us notice that the object fulfilling the DS role in this community is a C-object that represents the DS community.

In the DS Community, for the same reason of simplicity, we decide that each role is fulfilled by a distinct object.

The figure 2 shows the relationship between the e-shop community and the DS community.

As mentioned previously, the C-Object does not really perform actions. To define which component performs the action, we must describe its corresponding interface roles. In our example, the C-Object performs three actions that are **package an order**, **deliver the package and trace an order**

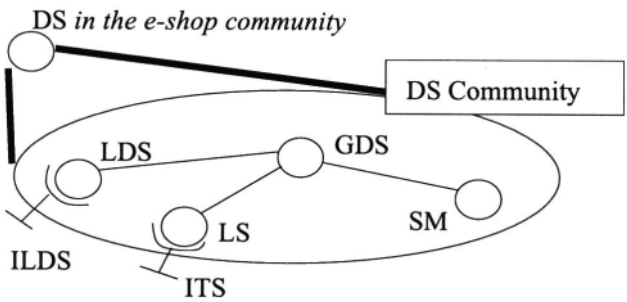

Figure 2. Relationships between a C-Object and its corresponding community

To define that the actions **package an order** and **deliver the package** are performed by the object fulfilling the role LDS, we define an interface role into the DS community that identifies these two actions. It should be noted that behavior identified by this role is only a part of the behavior already identified

by the LDS roles of the community. This interface role we define is named ILDS for Interface of LDS. It expresses that the actions "package an order" and "deliver the package" specified in the behavior of the C-object fulfilling the DS role in the e-shop community is in fact achieved by an object fulfilling the LDS role in the DS community. To define that the action **trace an order** is performed by the object fulfilling the role TS, we define another interface role into the DS community that identifies this action. This interface role, named ITS, expresses that the action "trace an order" specified in the behavior of the C-Object fulfilling the DS role in the e-shop community is in fact achieved by an object fulfilling the TS role in the DS community. It should be noted that the DS role of the e-shop community is a composition of the interface roles of the DS community. By this way, dependencies between the two communities are well expressed.

2.4. Conclusion

RM-ODP standard proposes very useful concepts such as C-Object and interface roles for specifying the relationships between communities, that are themselves very useful to partition specifications. However, the standard does not mention any guideline for elaborating an Enterprise specification in using these concepts. By proposing our construction rules of an enterprise specification, we provide such a guideline helpful for building specifications that can be distributed between the designers, as demonstrated in our example. This illustrates the specification of the e-shop can be distributed in such a way that the DS Community building is under the responsibility of the Europe team while the Stock community building is achieved by the Japanese team and so on.

Using these concepts while applying the rules we defined enables the designers to maintain consistency of the whole specification. Each piece of the specification is clearly delimited (community concept) and can be independently defined once the interfaces between communities (c-object and interface role concepts) have been properly identified and expressed.

3. THE DISTRIBUTED SPECIFICATIONS MANAGEMENT SYSTEM (DSMS)

When developing a specification distributed in various locations, the concerned team in a location must be able to access to the other pieces of the specification developed by the other teams. Reciprocally, it must offer access to its own piece. Thus all pieces of the specification must be accessible for all the teams who handle them.

We then propose a Distributed Specifications Management System (DSMS) that is a repository enabling the management of specifications. A DSMS stores specifications and offers facilities to handle them. Since the project team work-

ing on the elaboration is distributed, then the DSMS is distributed too. More precisely, it is designed as a set of local repositories interconnected by a CORBA bus. A specification is then distributed among these repositories. Each local repository stores and manages the piece of specification of the local team. Each team can access the piece of another team, i.e., access a remote repository.

The figure 3 shows the specification of the e-shop distributed into three repositories.

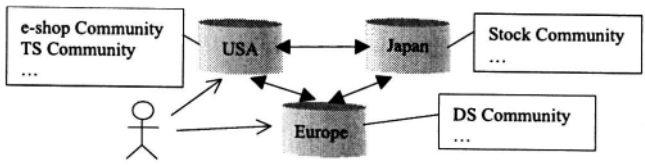

Figure 3. The e-shop specification in three repositories

Building a repository requires defining first how to encode the RM-ODP specifications and second how to provide the access to the repository. To address these two issues, our DSMS is built according to the MOF standard. We detail hereafter the use of the MOF standard we made in order to achieve the DSMS.

3.1. The MOF (Meta Object Facility)

The MOF is an OMG standard that defines mechanisms to handle meta-data [9]. Meta-data are data that describe other data. The MOF standard is mainly composed of two parts. The first part defines the so-called MOF model, that is a set of concepts needed to define the structure of meta-data. The main concepts defined in the MOF model are packages, classes, attributes and associations. This explains why the meta-model looks like UML class diagrams. The second part of the standard is a set of rules used to generate APIs for handling meta-data. These APIs are defined in IDL (Interface Description Language) [10]. They are generally used to build repositories.

3.2. Building an RM-ODP specifications repository

Our approach for the building of our RM-ODP specifications repository (i.e., the DSMS) in conformance with the MOF standard can be summarized as follows.

Advocating that specifications can be considered as meta-data, thus it is possible to define the structure of a specification in terms of a meta-model. As we deal with RM-ODP specifications, then we defined the structure of the RM-ODP specifications by building the so-called RM-ODP meta-model. Then we

applied on it the rules to generate the repository APIs. Since APIs are described in IDL, we have implemented classes that realize the interfaces of the APIs.

To achieve these steps detailed hereafter, we have developed M3J, a tool dedicated to the construction of MOF compliant repositories.

3.2.1 The RM-ODP meta-model. The RM-ODP meta-model we built makes use of the MOF model concepts. The figure 4 illustrates a part of this meta-model. It is composed of five classes representing the RM-ODP concepts of Community, Role, Object, C-Object and Behavior. Relationships between these classes represent the relationships between the RM-ODP concepts as defined in the RM-ODP standard. For example, a relationship between Role and Behavior concepts expresses that a role identifies a behavior. For sake of simplicity, we do not represent here some elements of the meta-model, such as the multiplicity of the associations or the attributes of the classes.

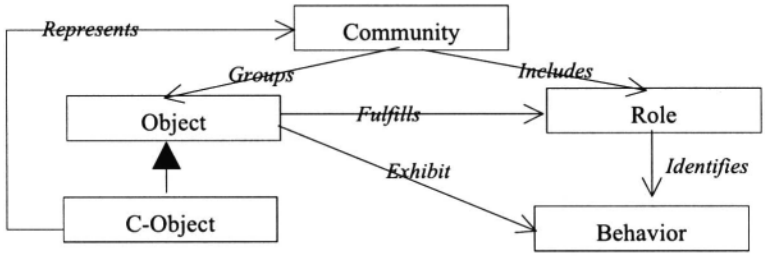

Figure 4. A part of the RM-ODP meta-model

The whole Enterprise RM-ODP is composed of 15 classes and defines the structure of the Enterprise RM-ODP specifications. It should be noted that this meta-model is a part of our contribution to the ISO working group elaborating the "RM-ODP Enterprise Language" standard, which is still in work.

3.2.2 The APIs and implementation classes generation. Rules defined in the MOF standard for generating the APIs generate two IDL interfaces for each class of the meta-model. One interface represents the instance of the concepts while the other represents a factory used to create instances.

For example, considering the Community concept, one interface named "Community" will represent the instance of a community. This interface will describe services to handle a community, such as the service "add_role" enabling the addition of a role in a community. The other interface named "Community-Class" will contain a service named "create.community" to create communities.

Moreover, an interface representing the repository itself is also generated. It describes services to obtain references to the factories. For example, the service

"community_ref" enables to obtain the reference to the communities factory. The table 1 presents a simplified version of the API of the repository.

Table 1. A simplified version of the API of the repository

interface Community {
add_role(in Role r);
add_object(in Object o);
...
};
	interface CommunityClass {
	Community create_community();
	...
	};

interface Object{
set_role(in Role r);
...
};
	interface ObjectClass {
	Object create_object();
	};

interface Repository{
CommunityClass community_ref();
Object object_ref();
...
};	

The APIs definition in IDL provides two advantages. First, the API is independent of any programming language, i.e., repositories can be compliant to the API while developed in any language such as Java or C. Secondly, repositories can be developed using CORBA and consequently can be distributed.

We make use of these benefits in developing our DSMS with CORBA and Java, thus a repository is composed of a set of CORBA objects. For example, let us consider an Enterprise RM-ODP specification composed of one community with three roles. Then, the repository storing this specification is composed of five CORBA objects, one representing the repository itself and four others representing the community and the three roles.

3.2.3 The M3J Tool. To implement the steps described above, we developed M3J, a tool dedicated to the construction of MOF compliant repositories [5] [7]. M3J is a tool that proposes a graphic interface to elaborate MOF compliant meta-models. It provides IDL interface generation as specified by the rules defined in the MOF standard. It also provides generation of implementation classes. M3J has been developed in Java. The description of other tools with similar facilities can be found in [6][8].

3.3. Use of the an RM-ODP specifications repository

To illustrate how a set of repositories enables the management of distributed specifications, let us consider that in our example, we have three sites, respectively in the USA, in Europe and in Japan. Each of these sites implements a

local repository as a CORBA object. Thus, there are three CORBA objects, each representing the Enterprise specifications repository of each location.

The USA team starts in building the e-shop community. For this, it uses the service "create_community" proposed by the "CommunityClass" interface (see table 1 the list of services of each interface corresponding to the concept). Then it creates the roles of the e-shop community using the "create_role" service of the "RoleClass" interface and adds the roles to the community using the "add_role" service of the "Community" interface. Then the USA teams creates the objects of the community using the "create_object" service of the "ObjectClass" interface and it links the objects to the roles using and the "set_role" service of the "Object" interface. For the C-Object, the "create_c_object" service of the "C-ObjectClass" interface is used.

The Europe team creates the DF community using the "create_community" service of its "CommunityClass" interface. As the DF community is represented by a CORBA object, the Europe team can export its reference towards the USA team. Getting this reference, the USA team can use the "set_community" service of the "C-Object" interface to set the link between the C-Object and its representing community, namely the DF community. The Europe team creates the interface roles, giving that the links between the C-Object and its interface roles are derived from the link between the C-Object and the represented community.

The figure 5 represents this part of the e-shop specification and the corresponding CORBA objects. The Enterprise specification is totally represented by CORBA objects and is distributed between the USA and Europe. The same mechanism can be applied to the complete specification and its distribution between the USA, Europe and Japan.

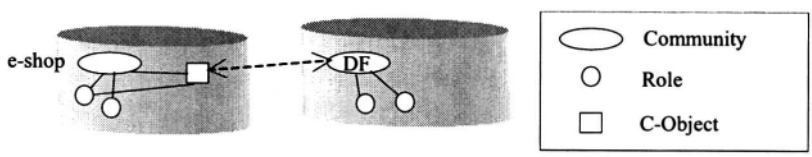

Figure 5. A part of the e-shop specification distributed over the USA and the Europe

4. CONCLUSION

Due to the fact that teams leading projects are more and more distributed and to the emergence of new paradigms such as the component paradigm, all specifications will be distributed in a few years. However, current technolo-

gies do not provide the mechanisms required in the elaboration of distributed specifications.

We identify that two aspects must be encompassed when dealing with distributed specifications. First, the modeling language must define some concepts to ensure the consistency of the distributed specification and secondly, some facilities must be available in order to manage and to handle the distributed specifications.

We then propose an approach to build Enterprise RM-ODP distributed specifications. It is based on the use of RM-ODP concepts and the definition of construction rules together with the provision of the M3J tool enabling the construction of an RM-ODP repository that is MOF compliant. Such a repository, also called DSMS, is based on CORBA. This enables interactions between remote repositories and thereby the provision of a distributed DSMS. Let us notice that M3J is more than an RM-ODP repository constructor. Actually, it enables the construction of any repository, which is MOF compliant. For example, this tool can be easily used to create a UML repository enabling the elaboration of UML distributed specifications.

This paper focused on the distribution aspects on the specification and illustrated how CORBA technology can be used to manage the distribution. However, other technologies are available and we have already investigated the building of repositories based on XML. For this, we use the XMI standard to generate the structure of the XML documents [5].

This work highlights the potential benefits of distributed specifications. It is particularly of interest to face the growing use of the component paradigm in the distributed software development. According to the approach presented in this paper, one can easily consider the storage of components specifications in repositories providing access facilities. Then, designers should be able to look at the description of a particular component. They could choose to get this specification in order to check if the component can be easily integrated or not.

References

[1] ISO/IEC, "ISO/IEC 10746-2 Information Technology Open Distributed Processing Reference Model: Foundations", 1996

[2] ISO/IEC, "ISO/IEC 10746-3 Information Technology Open Distributed Processing Reference Model: Architecture", 1996

[3] ISO/IEC "ISO/IEC 15414 Information Technology – Open Distributed Processing – Reference Model – Enterprise Language" Committee Draft Madrid 2000 Output 10 July 2000.

[4] P.F. Linington "An ODP approach to the development of large middleware systems" IFIP TC6 WG6.1 Second International Working Conference on Distributed Applications and Interoperable System (DAIS'99) June 28-July 1, 1999, Helsinki, Finland pp.61-74

[5] X. Blanc, M.P. Gervais and R. Le Delliou, *The Specifications Exchange Service of an RM-ODP Framework*, to appear in Proceedings of the 4th International Enterprise Distributing

Object Computing Conference (EDOC'OO), IEEE Press (Ed), Makuharin, Japan, September 2000

[6] Unisys, Universal Repository (UREP), http://www.unisys.com

[7] X. Blanc "M3J Project Web Site", http://www.lip6.fr/meta/Projects/M3J

[8] DSTC, "dMOF 1.0 An OMG Meta Object Facility Implementation", http://www.dstc.edu.au/Products/CORBA/MOF/.2000

[9] OMG "Meta Object Facility (MOF) Specification v1.3" TC. Document ad/99-09-05. OMG 1999. http://www.omg.org

[10] OMG "The Common Object Request Broker: Architecture and Specification v2.4" TC. Document ad/00-11-07 OMG 2000. http://www.omg.org

[11] OMG "XML Metadata Interchange (XMI) v1. 1" TC. Document ad/99-10-02. OMG 1999. http://www.omg.org

[12] OMG "Unified Modeling Language Specification v1.3" TC. Document ad/00-03-01. OMG 2000. http://www.omg.org

[13] M.P. Gervais,"ODAC : une méthodologie de construction de systèmes à base d'agents fondée sur ODP", rapport LIP6 n°2000.028 (in French), novembre 2000, http://www.lip6.fr/reports/lip6.2000.028.html