Tracing the Distribution Concern: Bridging the Gap

Nelly Bencomo, Gordon Blair, and Pete Sawyer
Computing Department, Lancaster University, Lancaster, LA1 4YR, UK
{nelly, gordon, sawyer}@comp.lancs.ac.uk

Abstract

Distribution is often presented as an example of a crosscutting concern that is difficult to modularize. This paper presents an approach for modeling distribution using a combination of AOSD and use cases. One of the aims of the paper is to bridge the gap between the handling of crosscutting concerns during the early and later phases of the lifecycle when developing distributed applications. With our approach the distribution concern is modularized in control objects in Analysis, in design control classes in Design and in distributed components in Implementation and Deployment. Use cases are used to establish a clear traceability among the analysis, design, deployment and implementation stages. In this sense, control objects of the analysis have a direct correspondence with distributed components in the implementation and deployment models.

Keywords*: distribution concern, aspect-oriented software development, use cases, traceability, separation of concerns*

1. Introduction

Aspect-oriented software development (AOSD) techniques support the modularization and composition of crosscutting concerns or aspects so that localization and reutilizations can be promoted. This results in reducing development, maintenance and evolution costs. AOSD approaches have been proposed to address the problem of crosscutting concerns at several stages during the software life cycle. However, there is a gap between the handling of crosscutting concerns during the early and later phases of the lifecycle. One of the aims of the paper is to bridge this gap when developing distributed applications.

Distribution is interesting because it is often presented as an example of a crosscutting concern that is difficult to modularize. The code associated with the distribution concern is in general scattered or spread across several units of the system [16]. Moreover, developers simultaneously think about business logic, security, performance, authorization, synchronization, and distribution concerns. The simultaneous presence of elements from each concern's implementation results in code tangling.

In this paper we propose an approach for modeling distribution using a combination of AOSD and use cases. We do not adopt an existing AOP language. Instead, our approach emphasizes what we call thematic use cases. A use case is thematic if it is related to a particular concern. In the example discussed in this paper, the distribution concern is the focus of thematic use cases. After identifying the thematic use cases, remote communication control classes are specified. These are specializations of control classes defined in UML and represent the abstraction of components that deal with remote communication and distribution. These control classes are designed and implemented using the corresponding design classes and their IDL interfaces.

With our approach the distribution concern is modularized in control objects in Analysis, in design control classes in Design and in distributed components in Implementation and Deployment. Use cases are used to establish a clear traceability among the analysis, design, deployment and implementation stages. In this sense, control objects of the analysis have a direct correspondence with distributed components in the implementation and deployment models.

The central idea is to treat distribution as an aspect of the application from the very early stages of the development, isolating the core business logic from the confines of system architecture. This brings many benefits such as making applications more resistant to change. In our specific case, it would ease the evolution of distributed applications as the middleware landscape changes over time.

The research is focused on the development of Distributed Object-based applications using CORBA. Our current research is towards an approach that addresses other technologies such as Services.

This paper is organized as follows. Section 2 presents an overview of our approach. Section 3 discusses the models of Analysis, Design, Implementation and Deployment in detail. Section 4 presents a case study. Section 5 discusses our research and compares it against some related work. Finally, Section 6 draws some conclusions and highlights some future work.

2. The Approach

The essence of our approach can be synthesized in three key phrases – *architecture, use case driven and traceability*.

2.1. Architecture

The architecture embodies the major static and dynamic aspects of a system. It is a view of the whole system highlighting the important characteristics and ignoring unnecessary details. In the context of our approach, architecture is primarily specified in terms of views of five models; the Use-Case model, Analysis Model, Design Model, Deployment model and Implementation model. These views show the "architecturally significant" elements of those models. The models have the following specific characteristics in our approach:

- The Use-Case model shows the thematic use cases related to functionality associated with distribution.
- The Analysis model illustrates how boundary, control and entity classes are associated with the thematic use cases identified in the Analysis. Remote Communication Control classes shown in this model are specializations of Control classes and represent the abstraction of components that deal with remote communication and distribution using CORBA.
- The Design model shows the design classes that trace the specialized Remote Communication Control classes in analysis. Special attention is given to the interfaces provided by these design classes. We show how some of these are represented by IDL interfaces.
- The Implementation model describes how elements in the design model are implemented in term of components.
- Finally, the Deployment model explains how CORBA-based components are assigned to nodes.

2.2. Use Case Driven

In the early steps of the life cycle, use-cases are mainly used to specify the functional requirements of the system. Later on, and based on the use-case model, developers create the models that realize the use cases. The developers review each successive model for conformance to the use-case model [5]. Our approach emphasizes *thematic* use cases. In general, the *theme* varies depending on the nature of the project. In our case, a use case is thematic if it is related to distribution. Once thematic use cases are specified identifying the Remote Communication Control, they are designed and implemented using the corresponding design classes and their IDL interfaces.

2.3. Traceability

An important part of traceability is that the final implementation is consistent with the design and analysis. As the design is refined to a concrete implementation, it is important that concepts have a clear correspondence to implementation artifacts – even if the mapping is not one-to-one [10]. In our approach, specialized control objects in analysis –that are associated with thematic use cases and are called Remote Communication Control Objects– are the abstractions of components in charge of remote communication in implementation. In between we define the design classes, specified by their IDL and UML interfaces.

3. Models

This section presents a short description of the Analysis, Design, Implementation and Deployment models.

3.1. Analysis Model

Control classes responsible for remote communication and that can potentially be mapped onto different nodes in the distributed system are identified. To do this, we define a Class Diagram (Architectural Description – Analysis View) that comprises boundary, control and entity classes of the thematic use cases. Initially, we have a control class for each use case. The generic class diagram proposed is shown in Figure 1. Control classes address the messages exchanged among boundary and entity classes to fulfill a specific functionality. Changes to identity or boundary classes are locally solved without changing their counterpart.

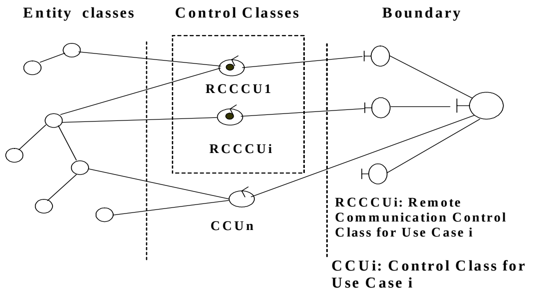

Entity classes
Control Classes
Boundary
RCCCU1
RCCCUi
CCUn
RCCCUi: Remote Communication Control Class for Use Case i
CCUi: Control Class for Use Case i

Figure 1: Class diagram objects related to thematic use cases

Because we are focusing on thematic use cases – related to distribution concerns- entity and boundary classes might be related to functionality associated with distribution. Entity and boundary classes are then abstractions of components deployed on different nodes. In these cases, the intermediary control class has to deal with remote communication and distribution. These intermediary control classes are specializations or adaptations of UML control classes in the Use Case Model. We adapted and stereotyped them to get the Remote-Communication Control Class (RCCC). As shown in Figure 1, RCCCs are graphically represented as a common control in UML with a filled circle inside. These RCCCs are the first link in a chain of artifacts that evolve from Analysis through all the process until reaching the CORBA distributed objects in Implementation.

In some cases, the nature of the application could dictate specific conditions of component distribution in the implementation. For example, two different sets of analysis objects might be required to represent implementation components deployed on different nodes. We propose to use a variation of the Analysis Class Diagram explained above. In these cases, the control classes identified are intermediaries that allow the communication among components deployed on different nodes. Figure 2 shows an example of a class diagram associated with the communication between different nodes. As in Figure 1, RCCCs have to deal with remote communication and distribution but this time entity classes are the only abstractions of components, boundary classes related with actors of the system are not included.

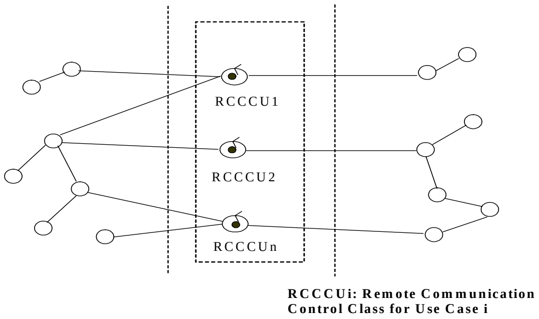


Figure 2: Class diagram associated with the communication between different nodes

In both Figure 1 and Figure 2, the dashed areas depict how abstractions related to the distribution concern are modularized in what we have called thematic use cases.

3.2. Design Model

In Design, there are two important activities to be performed: architecture definition and the specification of design classes. We study the use case realizations in analysis and define the corresponding design classes and their sequence diagrams.

Some design classes can be initially sketched from analysis classes; this is the case of design classes that deal with remote communication. A RCCC associated with the *use case i* in analysis will correspond to a pair of design classes. In Figure 3, the *trace* relationship between a RCCC and its two corresponding design classes is shown.

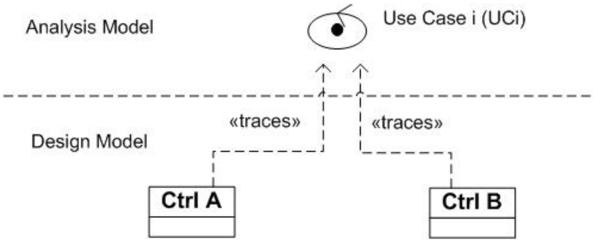


Figure 3: Correspondence between remote-communication control class in analysis and control classes in design

Basically, design classes expose two kinds of interfaces. One interface has the common UML semantics and the other is an IDL interface. IDL interfaces let CORBA objects communicate and send/receive the messages that components are receiving/sending. Methods of these interfaces are specified from the interaction diagrams. A concrete example of these interaction diagrams is shown in the case study of Section 4.

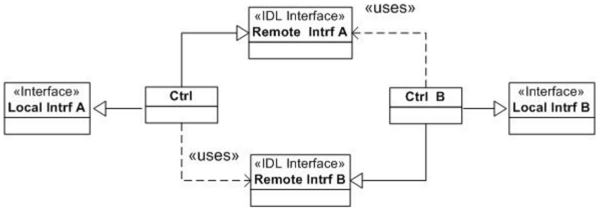


Figure 4: Control classes and their interfaces

The graphic notation used in Figure 4 has an alternative where IDL interfaces are represented by T-connectors, see Figure 5. The T-connector notation is based on [15].

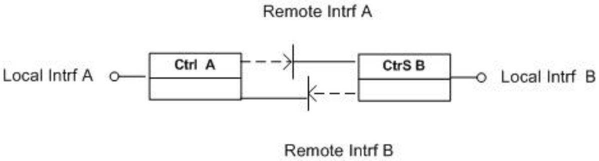


Figure 5: Another notation for IDL interfaces

3.3. Implementation and Deployment Models

In implementation, we have to program the code associated with CORBA objects and components based on the IDL interfaces in design. The Deployment model shows the mapping of CORBA components onto nodes.

Each design class traces to a CORBA Component in the implementation. Each CORBA component is a fundamental part of the system architecture. The graphic notation adopted to identify a CORBA component is based on [15]. The small ellipses and arrows in the top left corner represent remote interfaces and local (non remote) interfaces respectively.

To describe the functionality and interactions among components we define a diagram that includes and modularizes only CORBA components and their interfaces. This Diagram is used to define the Deployment Model**.**

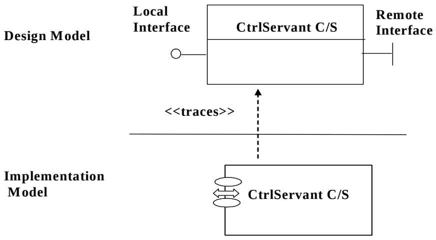


Figure 6: Correspondence between a control design class and a CORBA component in implementation

3.4. Traceability

Figure 7 shows the traceability among the different artifacts in Analysis, Design, Implementation, and Deployment models. Note that the Remote Class Control RCCUCi is related to the use case i.

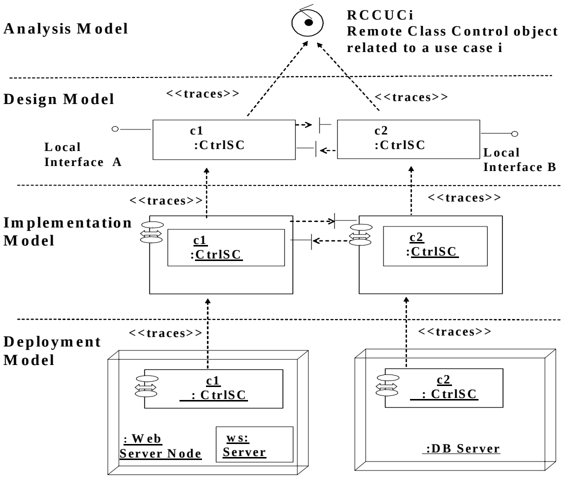


Figure 7: Traceability among the Analysis, Design, Implementation, and Deployment models related to the use case UCi

We have aspectized distribution from the very early stages of the development, isolating the business logic from the confines of system architecture. We start from a use case i and its RCCUCi. This object control is represented by two design classes in design that communicate using their IDL interfaces. These design classes are implemented by two CORBA components (Implementation Model) that are finally deployed onto different nodes.

4. Case Study: Banking System using ATMs and the Internet

We have a Banking software system that includes client services through ATMs and the Internet. A client uses the system to *withdraw, deposit, transfer* and *view* the balance of her/his accounts. Clients can use these services using ATMs or the Internet, see Figure 8.

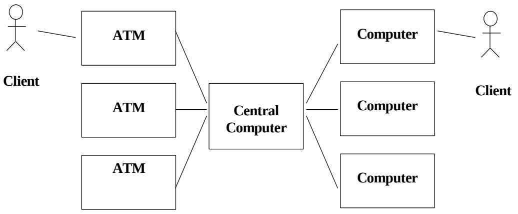


Figure 8: Banking system

4.1. Use Cases

Figure 9 shows the use cases of this system. The functionality is offered through ATMs (*withdraw, deposit, view balances, and transfer*) or through the Internet (*view balances and transfer*). For both cases, ATM and the Internet, we consider the use cases to *login* into the system.

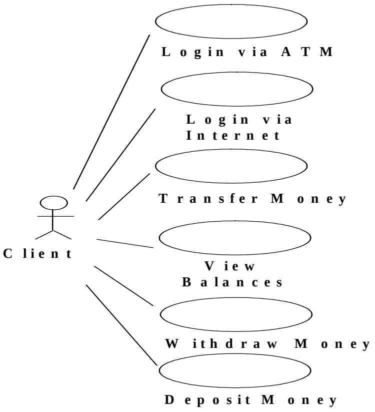


Figure 9: Use Cases of the system

4.2. Analysis: Class Diagrams and Packages

The class diagram related to use cases *Login via Internet, View Balances, and Transfer Money* when the user is using the Internet is shown in Figure 10. All these use cases are thematic as we can see that boundary and entity objects are abstractions of components deployed on different nodes. The intermediary control classes involved have to deal with the communication among boundary and entity objects and are specialized as Remote-Communication Control Classes.

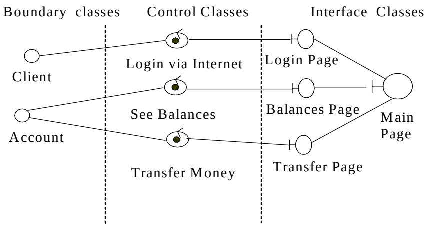


Figure 10: Class Diagram of the Case Study Banking System

It was convenient in terms of modularization of the system to group the analysis classes in three kinds of analysis package; an analysis package that contains classes related to boundary classes of the Graphical User Interface (GUI), an analysis package that contains the entity classes, and an analysis package that contains the control classes in charge of the logic of the remote communication between the boundary package and the application domain. In Figure 11 we have two analysis packages associated with the GUI, one related to the ATM and the other related to the GUI via Internet.

Figure 11 shows how abstractions related to the distribution concern are modularized in the Distribution Management Analysis Package.

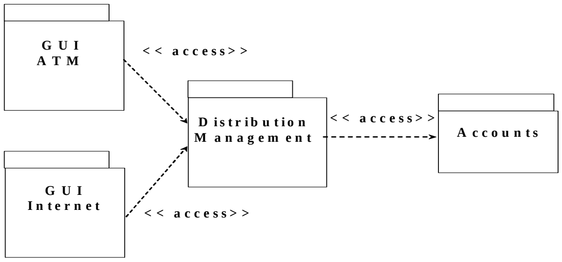
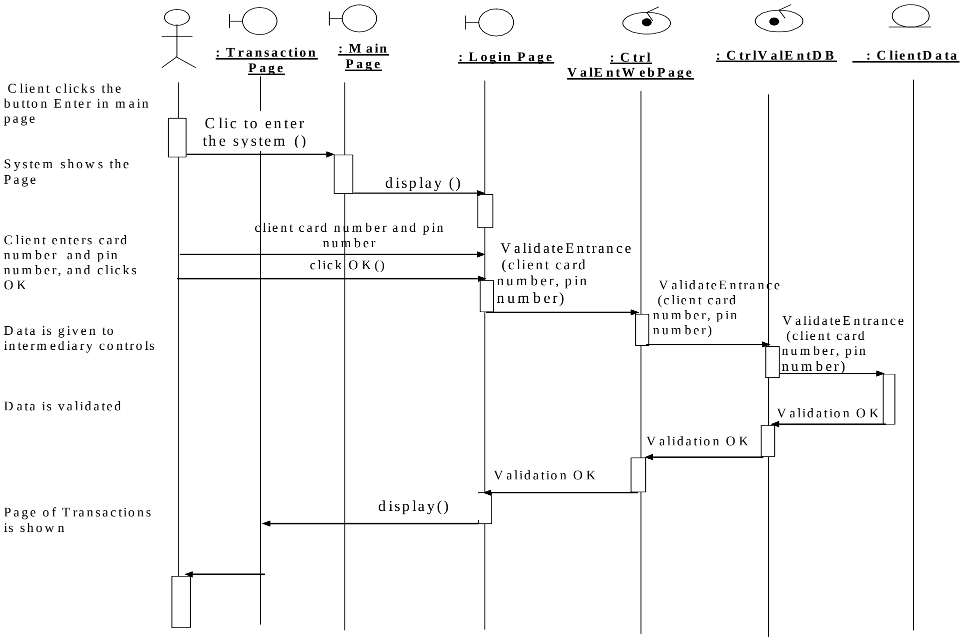


Figure 11: Analysis Packages

4.3. Design

We illustrate our approach in Design using the use case *Login via Internet*. This use case presents the following sequence diagram:



: Transaction Page
: Main Page
: Login Page
: Ctrl ValEntWebPage
: CtrlValEntDB
: ClientData
Client clicks the button Enter in main page
Clic to enter the system ()
System shows the Page
display ()
Client enters card number and pin number, and clicks OK
client card number and pin number
click OK()
ValidateEntrance (client card number, pin number)
ValidateEntrance (client card number, pin number)
ValidateEntrance (client card number, pin number)
Data is given to intermediary controls
Data is validated
Validation OK
Validation OK
Validation OK
Page of Transactions is shown
display()

Figure 12: : Sequence diagram for the use case Login via Internet

Messages *ValidateEntrance* and *ValidationOK* in the sequence diagram are candidates to be operations in the IDL interfaces of the control design classes *CtrltValEntDB and CtrlValEntWebPage*. The given name is related to the services provided on each side, Web Page side and Client Data side.

4.4. Remote Interfaces (IDL) and Local Interfaces (UML)

We have two control classes in the design; *CtrltValEntWebPag* and *CtrlValEntBD*. Figure 13 shows the IDL interfaces designed from the sequence diagram. Note that interface *CtrltValEntWebPag* contains the operation *validationOK()* and the interface *CtrlValEntBD* contains the operation validateEntrance(), operations that were identified from the sequence diagram.

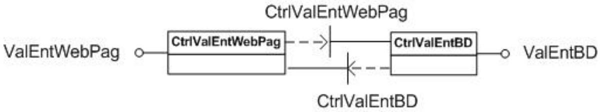


Figure 13: **Control classes in design, CtrlValEntWebPag and CtrlValEntBD**

CORBA offers the notion of IDL modules. Modules are used to encapsulate IDL interfaces. Example specifications of IDL interfaces are given in the IDL Module as follows:

```
//IDL
// Module:  Control of data verification
when entering the system via Internet
Module CtrlValEntWebPage {
// Operations on the Web page side
interface CtrlValEntWebPage  {
            void validationOK();
            void validationError();
};
// Operations DB side
interface CtrlValEntBD {
void validateEntrance(
            in long client_card,
            in long pin_number);
};
}
```

4.5. Implementation

Figure 14 shows the component diagram related to the use case *Login via Internet*. A complete diagram of all CORBA components in a system is given by the union of all CORBA component diagrams associated with all the use cases.

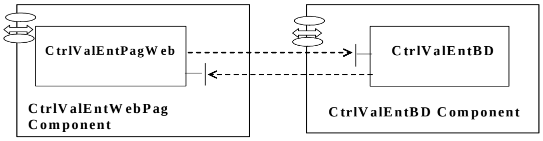


Figure 14: Component diagram: CORBA control components associated to the use case Login via Internet

4.6. Distribution

Figure 15 shows the deployment of CORBA components on nodes of the system. Specifically they describe the components related to the Use Case *Login via Internet*. Intermediary control CORBA components CtrlValEntWebPage and *CtrlValEntBD* are c1 and *c2* respectively. *c1* is on the Bank Web Server side node and *c2* is on the DB Server side (data of Bank clients).

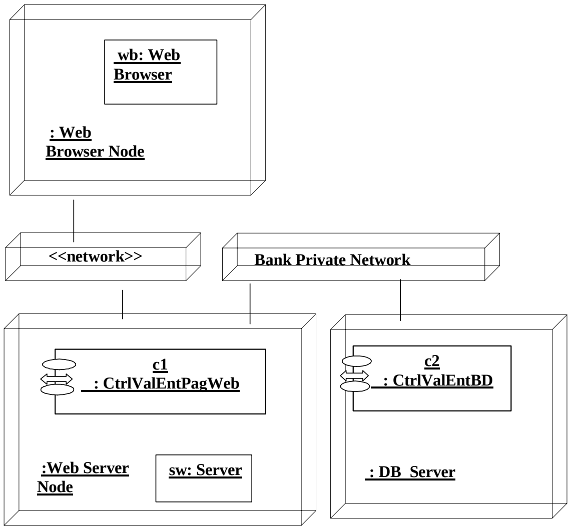


Figure 15: Distribution model: mapping of CORBA components onto nodes of the system

5. Discussion and Related Work

Simmonds et al. [14] describe their experience using a framework for software development incorporating the use of aspects to model middleware technologies based on the MDA vision. Their main goal is to decouple the design of an application from its target middleware promoting middleware-transparent development. The high level design architecture is independent from the middleware. Abstractions of the application that are specific to the middleware technology are modeled separately using aspects that are integrated (woven) into the application later in the developments process.

We also use aspects to model distribution, but our approach focuses on traditional methodologies of software development that are very different from the MDA vision. In Simmonds' framework, details of middleware technology implementations are treated as crosscutting concerns and modeled as aspects. Our approach does not give any details of which AOP techniques to use in implementation but introduces aspects, and hence separation of crosscutting concerns, at the analysis and design levels. We argue that our research helps in the process of definition of what an aspect is at early stages of the development process and how it maps to artifacts at later stages when developing distributed applications.

Ivar Jacobson [6] writes about the relationship between use cases and AOP claiming that both can be viewed as equivalent. Jacobson proposes slicing the system, use case by use case, over several of the most interesting lifecycle models to keep the use cases separate all the way down to the code. At some later time these slices are recomposed into a consistent whole—an executing system. This is different from our approach, where thematic use cases (use cases that are related to distribution concerns) and their subsequent realizations through all the lifecycle models permit the distribution concern to be encapsulated in separate modules. As a consequence, thematic use cases act as aspects to provide both localization and traceability among the different artifacts in Analysis, Design, and Implementation and Deployment models.

6. Conclusions and Future Work

We have presented our experience in implementing distribution using a combination of AOSD and use cases. We use the concept of thematic use cases to treat distribution explicitly as an aspect of the application from the very early stages of the development and help insulate the programmer from its cross-cutting characteristics.

Treating distribution in this way provides a clear traceability from the Use Case model through Analysis, Design, Implementation, and Deployment models. Specialized control objects in analysis, called Remote Communication Control Objects, are the abstractions of components in charge of remote communication aspects in implementation. This approach is based on practical experience [1,2,3,4].

We have used use cases and their subsequent realization through all the lifecycle models to encapsulate distribution concern in separate modules promoting localization and reutilization. These modules and localizations are reflected in the architecture of the system. As a consequence, the approach lets us guarantee the traceability through the different artifacts in Analysis, Design, and Implementation and Deployment models.

The next step in our work is to identify the set of (sub) concerns than can derive from distribution (for example authorization and security) [9] to investigate how our approach can take these into account. We eventually plan to apply our model to case on Service-Oriented applications.

7. References

[1]Bencomo N., and Matteo A., A Unified Process Adaptation for Distributed Objects Applications, Submitted to Software Engineering and Middleware, SEM 2004

[2]Bencomo N.,: CORBAdapted-UP: Una adaptación del Proceso Unificado para la Construcción de Aplicaciones CORBA, Promotion Project to get the Cathegory of Assistant Lecturer, Escuela de Computación, UCV, Venezuela, 2002

[3]Bencomo N. Matteo A.: Correspondencia entre Modelos en el Desarrollo de Aplicaciones CORBA, Chapter in book on Avances en teconologías de la Información, Universidad de los Andes, Venezuela 2003

[4]Bencomo N., et all.: An experience using CORBA and OOSE in the construction of a Graphical multi-user Interface based on Distributed Objects, Proceeding of Practical Experience Segment of Objetos Distribuidos 2000, Sao Paolo, Brazil, 2000

[5]Hunt J.: The Unified Process for Practitioners Object Oriented Design, UML and Java, Springer, 2000

[6]Jacobson I.: Use Cases and Aspects – Working Seamlessly Together, in Journal of Object Technology, vol. 2, no. 4, July-August 2003, pp. 7-28.

[7]Jacobson, I., Booch G., Rumbaugh J.: The Unified Software Development Process, Addison-Wesley , 1999

[8]Jacobson, I., Magnus C., Patrik J., Gunnar O.: Object-Oriented Software Engineering: A Use case driven Approach, Addison-Wesley, 1993

[9]Farooqui K., Logrippo L., Meer J.: The ISO Reference Model for Open Distributed Processing- An Introduction, 1996

[10]Ovlinger J.: From Aspect-Oriented Model to Implementation, Position Paper for AOM, 2003

[11]Rashid A., Sawyer P., Moreira A. and Araujo J.: Early Aspects: A Model for Aspect-Oriented Requirements Engineering, IEEE Joint International Conference on Requirements Engineering. IEEE Computer Society Press. Pages 199-202.

[12]Rashid A., Moreira A. and Araujo J., Modularisation and Composition of Aspectual Requirements, Proceedings of 2nd International Conference on Aspect-Oriented Software Development: ACM Press, pp. 11-20.

[13]Rashid A. and Chitchyan R.: Persistence as an Aspect, Proceeding of 2nd International Conference on Aspect-Oriented Software Development. ACM. Pages 120-129.

[14]Simmonds D., Ghosh S., and France R..: An Aspect-Oriented Model Driven Architectural Framework for Middlware Transparency, AOSD Workshop on Early Aspects 2003: Aspect-Oriented Requirements Engineering and Architecture Design, March, 2003.

[15]Slama D., Garbis J., Russelm P.: Enterprise CORBA, Prentice Hall, 1999

[16]Soares S., and Borba P.: PaDA: A pattern for distribution aspects. Proceedings of: Second Latin American Conference on Pattern Languages Programming - SugarLoafPLoP. 2002 Brasil. ICMC - Revista da Universidade de São Paulo, páginas 87-99.

[17]Soares S., Laureano E., and Borba P.: Implementing Distribution and Persistence Aspects with AspectJ. In Proceedings 17th ACM Conference on Object-Oriented programming systems, languages, and applications, OOPSLA'02, ACM Press p.p. 174-190.

View publication stats