A Component Meta Model for Reused-Based System Engineering

Frédérick SEYLER, Philippe ANIORTE

LIUPPA
IUT de Bayonne – Département Informatique
Place Paul Bert
64100 BAYONNE – France
{aniorte,seyler}@iutbayonne.univ-pau.fr

Keywords : Component Model, Reuse and Integration, Model Driven Architecture, UML Profile

Abstract : The domain of research of our team crosses system engineering based on reuse, component paradigm an interoperability. Our goal is to provide the integration of existing heterogeneous distributed components. This integration must be plat-form independent, in order to face the problem of middleware proliferation. First, we position our approach with regard to Model Driven Architecture. We briefly present a new component meta model which rests on the principle of de-coupling between components, and supports the separation between the data flow and the control flow. Then we present a reuse process based on MDA. A UML Profile allows the definition of the meta model and the mappings between the PIM and the target PSMs. To complete the definition of this UML profile, we follow by presenting a language intended to describe the content of customized interactions between components.

Introduction

Because of the increasing complexity of Information Systems (IS) and their constant evolution, it is becoming more and more difficult to re-develop them. Moreover, developed applications are more expensive and are less reliable. Therefore, a strong need of reuse has been expressed by firms. The reuse provides a set of solutions developed in the research and development domain to face on the software crisis. It is defined as a new approach of system engineering allowing to build systems from existing elements, compared to traditional approach where a new system starts from scratch and needs to be re-invented each time.

The reusable components engineering's goal is to produce an infrastructure for reuse. It covers the identification and the specification of components, their organization and their implementation. The activity of identification and specification of reusable components is essential in the process because they have the objective to produce reusable resources. The approach with reverse-engineering is considered as down-top. It is characterized with the exploitation of existing development products to provide components. This approach can be declined under several forms as reuse models [CALD91], analysis of similarities [CAST92] or research of analogies [MAID91] [MAID93]. The domain analysis approach is the second trend. This is a top-down approach which may be simply defined as the identification of objects and the identification of a field of application.

The architecture of reuse constitutes a coupling device between the two essential activities of the reuse, the engineering of components and the engineering of systems. Those activities are still considered as independent activities, the activity of one may just be an input for the other. The coupling between these two activities is weak, a model does not exist to really integrate them. Practically, the most used forms of organization are the organization producer/consumer and the interlaced organization. With the first one, the system engineering uses reusable components provided by the components engineering, with the interlaced organization, we do not only use components but also contributes to the development of these components. Such an organization is used in the Care system [CALD91]. System engineering deals with two activities: the first one is the research and the selection of components, the second one is the adaptation and the integration of components. Adaptation techniques are numerous : modification of the component, instantiation mechanism, parameterization and specialization.

The reuse approach needs models of components representation, in which two principles are essential : the abstraction principle and the variability principle. Our studies contains the definition of components meta model respecting the abstraction principle and the variability principle, the proposition of component-based methodologies, and the development of tools allowing the development by reuse.

The term "component" is widely used in the reuse paradigm with a generic connotation of reusable entity. The "component" paradigm is emerging in the more specific domain of software components on which we are more concerned. Component-based approach rests on the principle of de-coupling between

components. The interest is that dependencies between entities are no more used into these entities, but defined out of the components. Component-based approach proposes a mechanism of communication. For example, with the component model MALEVA [MEUR01], components have communication marks to allow components to connect each other. Each component can be specified by its input/output marks. Authors use the term of interface. The general idea is to provide a communication abstraction independent of operational choices.

MALEVA [LHUI98] proposes a component model based on the separation of the data flow and the control flow between components, and lets two kinds of marks appear: the data marks allow communication of data between components, control marks are used to provide control. Consequently, it is possible to manage active and independent components, becoming more generic because a part of the control description is extracted of the component to be included into the graph. The graph provides to the component integrator the capacity to modify the execution policy between components. The distinction between the data flow and the control flow is not new. In another context, it has been introduced into SADT [MARC87]. Nevertheless, this separation is mainly proposed at the design level for the management of projects. In the component approach it is accessible at the implementation level.

The reuse of components needs a meta model for representing the components and an architecture to exploit this model.

Metamodeling is the primary activity of specification. Interoperability in heterogeneous environments is allowed by the building, publishing and understanding of shared metadata or models [POOL02].

The approach proposed by the OMG[OMG02] with Model Driven Architecture (MDA) [SOLE00] is to migrate the reuse from the component definition in a specific middleware (like CORBA, EJB, xml/SOAP) to its conceptual model itself. The goal of MDA is to capitalize and reuse models of application instead of regularly migrate software components from a middleware to another. MDA proposes an architecture for models that provides a set of guildeness for structuring specifications expressed as models [MILL01].

For creating MDA-based applications, the first step is to create a Platform Independent Model (PIM), which should be express in UML. There is multiple levels of PIM: all PIM level can be an abstraction or a refinement of another: for example, computational business model is a high abstract PIM, and a Platform Independent Component View is a refinement of this, including interfaces, interactions etc... All those models are bound by refinement correspondences.

Such a PIM can then be mapped to a Platform Specific Model (PSM) to target platforms like the CORBA Component Model (CCM), Enterprise Java Beans (EJB) or Microsoft Transaction Server (MTS), etc.... Standard mappings should allow tools to automate some of the conversion. Such a PSM, again expressed in UML, can then be actually implemented on that particular platform.

A UML Profile (core Model) is an extension of the MOF meta model allowing to express new specific views of systems. The UML Profile also express mappings from one model to another. Thus, one way to exploit a new meta model is to create an UML profile with its own graphical representation and its own mappings to target middlewares.

The reuse Architecture

The reuse paradigm lets two activities appear : the engineering of reusable components, and the engineering of system by reuse of components, with a weak coupling between these two activities. Even if we first present the component meta model, then its use, we want to have an interlaced approach.

Our meta component model is the formalization of our reuse infrastructure, the traditional goal of the engineering of reusable components. In our case, essentially, these are the results of a specification phase. We can point out that we are working on autonomous software components to re-deploy. Identification activities, organization activities and implementation activities do not present interest here.

. *A Component Meta Model*

Our component meta model also supports the separation between the data flow and the control flow, and lets two kinds of interaction points appear : input/output information points and control points. At this level, our approach is comparable to the one of [MEUR01]. We will now enter into the detail to show the originality of our approach. Intuitively, Input Information Points (IIP) are acquisition points for a component. On the contrary, Output Information Points (OIP) are points to restitute. The following schemas illustrate this with a graphical representation associated to our component model :

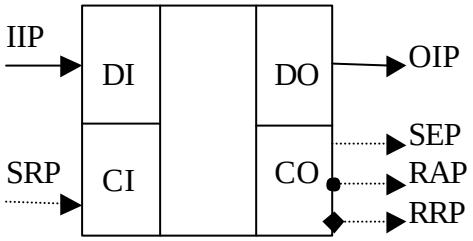

Figure 1 :Component Interface

The component model MALEVA only offers one type of control marks. We propose several types of control points : synchronization points and resource sharing points. Synchronization points are dedicated to the synchronization of components. When a component synchronizes another one, the first one is called "the synchronizer" and the second one is called "the synchronized". The synchronizer has a Signal Emission Point (SEP) whereas the synchronized has a Signal Reception Point (SRP).Resource sharing points constitute an answer to the problems of sharing of resources. This is typically encountered with concurrent executions. This is the case with autonomous software components we propose to manage. Moreover, the resource sharing and the synchronization that we have previously presented, appear in works dealing with coordination. To provide the resource sharing, we use Resource Access Points (RAP) and Resource Release Points (RRP). The first ones allow to get the resource when available. The second ones is to release the resource when used.

. *ASIMIL: an example of Components definition*

Aero user-friendly SIMulation-based dIstance Learning (ASIMIL)[ASIM02] is an European project to improve training for aeronautical staff with the help of Intelligent Agents. Different programming languages (Java, C, C++, Macromedia Director, script CGI…) have been used to develop these different parts. Each part represents a component which must be reused to develop a distributed application. The job of our research team is to integrate these different components on a net. In these sense, the ASIMIL project is a good field of application of our research domain. Let us present the interface of different components of ASIMIL:

- **PFC** : Procedure Follow-up Component. It manages training procedures and exercises. It tracks learner's progression during the procedure. After each action performed by the trainee on the simulator and after each step of the procedure, PFC sends a « Required Action » value to its OIP. PFC is able to synchronize other components using a SEP. Therefore, PFC has its own RAP and RRP, which serve to allow printing of a procedure or data showing in a shared window. A IIP allows to PFC to get the error type and gravity of trainee's action.

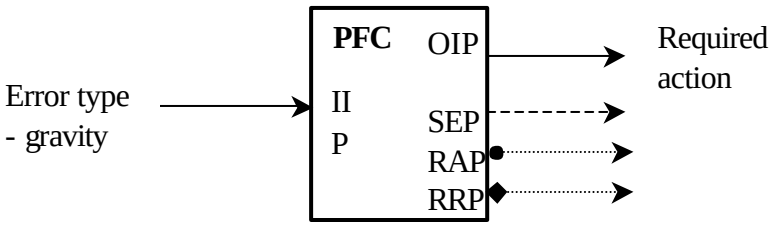

Figure 2 :Procedure Follow-up Component

- **FSimu** : Flight Simulator. Every action performed by the trainee on FSimu, is put in a OIP. The learner uses directly the graphic interface in order to pilot the aircraft. This component can be started by another one via a SRP.

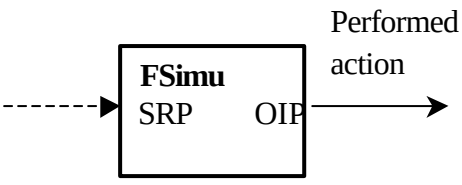

Figure 3 : Flight Simulator

- **MultiAgentSystem** (MAS): The goal of this component is to interpret trainee's errors. Every couple of data (Waiting Action, Performed Action) it receives via IIP (Input Information Point) is analyzed . When an error is detected an characterized , MAS may perform an action by sending a signal on a SEP (Signal Emission Point).With the help of RAP and RRP , MAS is able to print of show the help instruction in another shared resource like printer or window in order to help the trainee during his exercise. This component can be activated by another one via a SRP (Signal Reception Point)

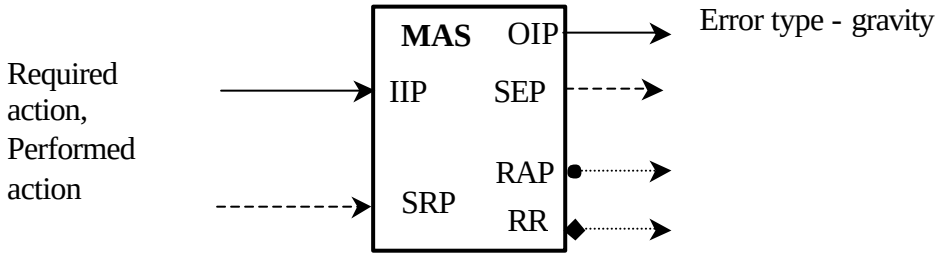

Figure 4 : MultiAgentSystem

- **HistoryManager :**This component is intended to manage the history of learner's interactions with the system , for pedagogical analysis purposes. The SRP is intended to allow to it to start.

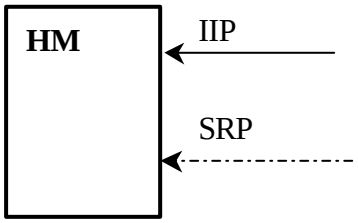

Figure 5 : HistoryManager

Interaction mechanisms between components

The reengineering by reuse of components consists of using our component model. This use concerns mainly the components integration and, a bit less, their adaptation. The selection activity is not suitable with our objectives. Components integration consists of managing interactions between components [ANIO01a]. We find information interactions, control interactions and mixed interactions. We also propose two predefined tools to help the architect in his work: the mailbox and the shared resource

. *Information interactions*

We can transfer information between The OIP of a component is linked to the IIP of another component.

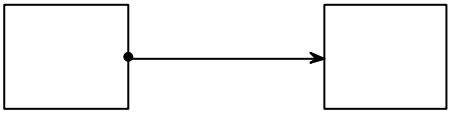

Figure 6 : continuous flow transfer

When we write "building of information" we mean producing a needed piece of information from one or several pieces of information provided by one or several components.

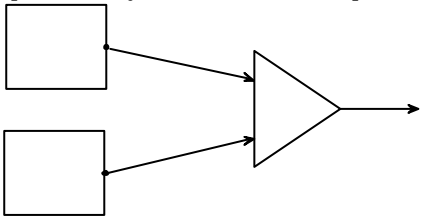

Figure 7 : building of information from two pieces of information from two components

This offers very interesting perspectives because it allows to create “ad hoc” interactions and thus facilitates the integration of components. The unique constraint is that the building of information have to be algorithmically expressible. More generally, we offer a solution to the classical problem of the finite number of tools, not enough to process the variety of problems.

. *Control interactions*

Control interactions deal with the synchronization of components, the resources sharing between components and the complex control interaction. Concerning the synchronization, we only have to link a SEP with a SRP. Then, the component whose SEP is used synchronizes the one with the SRP.

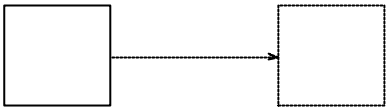

Figure 8 : synchronization of two components

Complex interaction control is similar to the building of information. In other words, it allows to create another element of control from several elements of control.

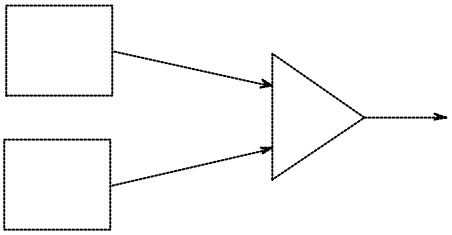

Figure 9 : complex interaction control

. *Predefined tools*

The first tool offer by the platform is the transfer via mailbox. Let us notice that our purpose is not at the implementation level. In this case, contrary to the stream transfer, the information is not directly addressed to a component but deposited into a mailbox from where an unnecessary identified component will get it.

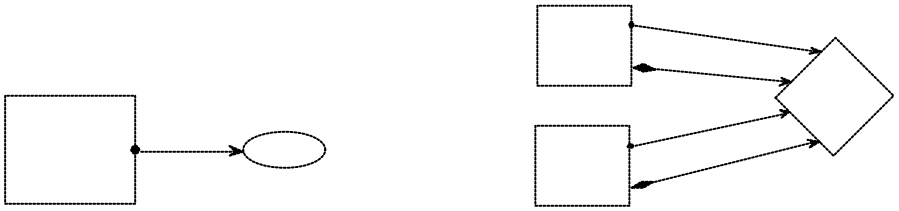

Figure 10 : mailbox transfer, shared resource

If several components need the same resource during their execution, we need to provide the resource sharing thanks to RAP and RRP of the concerned components :

A reuse process based on MDA

The main interest of MDA in our research domain is to help us with formalize and use our propositions about a meta component model and its exploitation. The core of MDA comprise a few UML profiles, each representing a specialized environment.

The heart of our process for reuse is a UML Profile allowing the integration of our Component Meta Model to MDA, in order to exploit our architecture for reuse (ie the reusable components). This UML Profile contains the formal and a graphical definition of all the elements intuitively introduced in the meta model and their interactions previously presented, it also contains mappings to specific platforms as EJB, CCM, and a

specific notation to introduce a graphical representation of the components and interactions as in UML profile for EDOC [EDOC01].

A first fundamental aspect of this UML profile is a meta model for designing high level autonomous components, and their data or control flow interactions, it uses UML notation with some extensions and conventions to make diagrams more readable.

The Component element represents the autonomous software components to re-deploy. A high level Component is able to interact with other components with three kind of information interaction: event, methods, continuous flow. The InOutInteractionPoint is the superclass of this element: OutputInformationPoint is the point from which data (methods, events, continuous flow) are going out, the InputInformationPoint is the entry point for the functional dependences (methods, events, continuous flow) of the component. SignalEmissionPoint, SignalReceptionPoint, ResourceAllocationPoint, ResourceReleasePoint are other elements of the meta model expressing the control points. An Interaction is intended to connect two components, there is two kinds of interactions: data and control one. A complex interaction is defined as a predefined component, which can be connected with data or control interactions. The generic tools proposed by the reuse platform to help the architect : Mail Box, Shared Resource are also defined in the profile as subclasses of Components.

The second aspect of this profile is the introduction of a new diagram, added to the standards diagrams of UML, and following the intuitive graphical representation of Components, interactions and tools of the previous chapter.

. *Meta model + Predefined Tools = second level PIM*

The previously defined technology independent profile is used to define Platform Independent Models of reusable component-based applications in accordance to MDA specifications. The expression of the integration of the component is made by the architect of the application with the help of a graph of interactions using the specific diagram defined in the profile. In the MDA point of view, this graph is a second level PIM, or a platform independent component view. Therefore the only expression of the PIM following the specification of MDA don't allow the architect to express the content of customized interactions. This content can only be specified by the architect of the application, without implementations considerations. The following graph shows the PIM of an ASIMIL application.

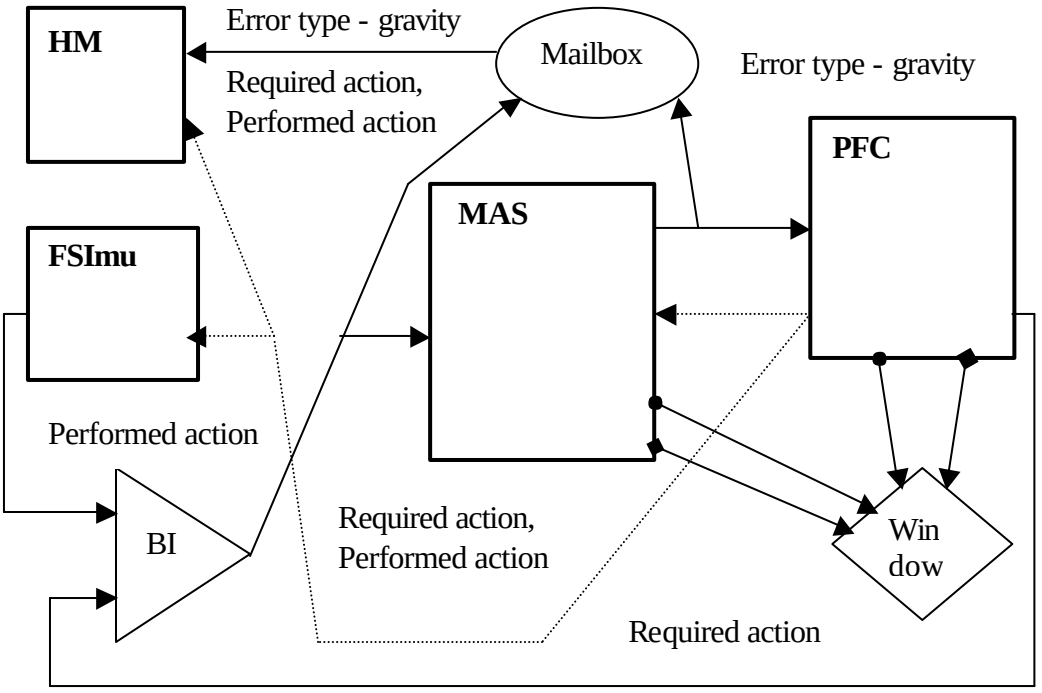

Figure 11 :The PIM of ASMIL application

This PIM uses predefined interaction (direct connected data flow, synchronization) and dedicated interactions. PFC(Figure 2) and MAS (Figure 4) cannot be directly connected one to another, because PFC provides a Required Action on its OIP and MAS needs a couple Required Action, Performed Action on its IIP. This yields that a customized interaction called "building information" (BI) is construct

between PFC, FSimu and MAS. Nevertheless, all the interaction points provided by MAS are not necessary connected in the interaction graph, which illustrate the principle of adaptability. In this graph, HM(Figure 5) has an asynchronous operating mode. When an operation is requested to it for an example by the MAS, HM consults MailBox to get messages addressed to it.

ASIMIL application is managed by PFC which starts all the components. The trainee can perform an action on FSimu (Figure 3), action which coupled with a required action provided by PFC and analyzed by MAS with the help of HM. If the action is not required, PFC shows an error message on its window.

. *Mapping: from PIM to PSM*

The next step of the using of MDA is to transform the PIM to a PSM, those mappings are the last aspect of our UML profile. The PSM, also expressed in UML is the formalization of the architecture of the application according to a technology or another. As in the EDOC profile [EDOC01], the UML profile is able to provide a set of mappings allowing to transform this PIM in target platforms as CCM or EJB.

After this automatic our handle mapping, the developer is intended to implement the content of new components as customized interactions following the "Adapter" pattern [GAMM94]. This phase also contains a part of adaptation of the components.

Conclusion

This paper seems to be a contribution for the design of distributed software architectures based on component reuse. From general point of view, it offers a global solution from the component model to the implementation and an industrial application, with ASIMIL project.

First of all, this approach is based on a strong de-coupling between components. This allows a more important reuse of existing components. More and more, firms express the hope to "superpose" new systems to existing ones. This is often encountered with cooperative systems, framework of our application field, but also with knowledge based management systems. In these cases, objectives of these firms consist of reusing the existing reliable system the more widely possible, before envisaging a value added thanks to a cooperative system or a knowledge management system.

The component meta model constitute the heart of the contribution, and allows a rich activity of component specification, adapted to the problems of the reuse of components. As shown with the description of ASIMIL components, this model provides multiple interaction points to express the use of components black-box including their control. The integration of this component meta model in MDA in defining an UML profile will allow the component meta model to be used and tested as existing standards.

Concerning the construction of the graph of interaction, numerous basic tools are completed by devices which can be developed especially for the application. This allows a higher de-coupling between components to reach a more systematic reuse.

To finish, the platform dependant model warranty the implementation of the graph of interactions after the mapping from PIM to PSM. A first version has been achieved. It has been developed with Java, and especially JavaBeans. Interoperability between each Bean is realized with RMI (Remote Method Invocation). Nevertheless, all of these devices have not been implemented . This is the case for the control and consequently for some operative devices (control and mixed interactions). We are currently working on these aspects, with the help of ASIMIL experimentation. Therefore we look for implementations issued of the research domain as [SIRA00], but also for "commercial" implementations around the domain of components. After considering the limits of CORBA, we are interested in SOAP (Simple Object Access Protocol) based on TCP/IP and so much better adapted to heterogeneous environment whereas CORBA use its own protocol (IIOP). Moreover, SOAP uses XML as the description language inter-acting with UDDI (Universal Description, Discovery, and Integration), kind of interoperable yellow page mechanism.

References

[ASIM02] *Presentation of ASIMIL project* -http://www.cordis.lu/ist/projects/99-11286.htm

[CALD91] CALDERIA G., BASILI R.V. - *Identifying and qualifying reusable software components* - IEEE computer, February 1991

[CAST92] CASTANO S., DE ANTONELLIS V. - *A model for reusable requirements* - Esprit project, report pdm 2-1-3-r1, 1992

[EDOC01] *A UML Profile for Enterprise Distributed Object Computing*- Joint Final Submission, Part I Version 0.29,18 June 2001

[GAMM94] Gamma E., Hebn R., Johnson R., Vlissides – *Design Patterns, Elements of Reusable Object- Oriented Software*. Addison-Wesley,1994

[LHUI98] LHUILLIER M. - *Une approche à base de composants logiciels pour la conception d'agents – Principes et mise en œuvre à travers la plate-forme MALEVA* - Thèse de l'Université Paris 6, Février 1998

[MAID91] MAIDEN N. - *Analogy as a paradigm for specification reuse* - Software engineering journal 6(1), 1991

[MAID93] MAIDEN N. , SUTCLIFFE A. - *The domain theory : object system definition* Nature report CU-93-OOA, 1993

[MARC87] MARCA D. A., MCGOWAN C. L. - *SADT Structured Analysis and Design Technics* - McGraw-Hill, New york, 1987

[MEUR01] MEURISSE T., BRIOT J.P. - *Une approche à base de composants pour la conception d'agents* - Technique et Science Informatique Vol. 20 n°4/2001, 2001, p. 567-586

[MILL01] Miller J., Mukerji J., -*Model Driven Architecture,* - Architecture OMRSC, July 9 2001,

[POOL02] *Model Driven Architecture: Vision, Standards and Emerging Technologies*, ECOOP 2001 ,2001

[SOLE00]: R.SOLEY,-*Model Driven Architecture-White Paper*, November 27, 2000

[SIRA00] : Projet SIRAC *: Systèmes Informatiques Répartis pour Applications Coopératives* - Rapport d'Activité 2000, INRIA