

COMPONENT-BASED SOFTWARE DEVELOPMENT : FROM COMPONENT MODEL TO SOFTWARE ARCHITECTURE

Philippe ANIORTE

LIUPPA

IUT de Bayonne – Département Informatique

Place Paul Bert

64100 BAYONNE – France

aniorte@iutbayonne.univ-pau.fr

Keywords : *Component-based software development – Component model – Software architecture – Model Driven Architecture – Reuse – Integration – Interoperability*

Abstract : *Today, Information Systems are often distributed and heterogeneous. Thus, software systems become more and more complex and their evolution is difficult to manage. Our works deal with engineering of heterogeneous distributed systems based on reuse. Such systems need a distributed adaptable software architecture to be implemented. In this paper, we propose a component model for developing adaptable software. First, we briefly present the component paradigm in which we place our works. Then we position our component model with regard to related works. The interface of the component is described by the way of points of interaction. These points are used to manage different types of interactions in order to build a graph of interactions allowing the integration of the reused components. We finish with the presentation of the distributed adaptable software architecture allowing to implement this graph. Each part of this paper is illustrated with a concrete case, the European ASIMIL project.*

Introduction

Technological and economical mutations of these last years have really modified the life of organizations. They have engendered the multiplication and the dissemination of heterogeneous information [SING98]. So, more and more, we have to qualify Information Systems (IS) as distributed and heterogeneous ones.

Because of the increasing complexity of IS and their constant evolution, it is becoming more and more difficult to 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 two previous points deal with interoperability. More and more, the idea of interoperability becomes a need to answer to new organizational demands and benefit of technical improvements, especially with networks and Internet. We can compare this approach to a monolithic vision of systems and their heavy evolutions in terms of complexity, delay and costs.

The research domain linked to these problems is very wide. We are interested in the engineering of heterogeneous distributed information systems based on reuse. Our approach is in keeping with the recent research in developing High Confidence Software and Systems (HCSS) [HCSS01]. It is based on the “component” paradigm. The objective is to reuse software existing components and to integrate them in order to make them cooperate. The application field in the context of our laboratory gives us a certain experience with the re-engineering of distributed applications [ANIO01b]. The challenge consists of conserving the quality of the existing application made of several dispatched elements developed independently.

The problem of these works is shared with several well identified research domains : distributed IS, reuse, interoperability, cooperation. The difficulty lies in the cross feature of our approach. Indeed, it is the actual trend. For example, CIS (Cooperation Information Systems) have concerns close to ours. CIS are presented as a new domain of research [DEMI97] in synergy with IS technology, CSCW (Computer Supported Collaborative Work), modeling and planning theories. Some of the encountered problems with this approach [HERI01] are near to us. This is the case with interactions between autonomous parts which is the support of coordination activities. Nevertheless, the objective is relatively far because CIS want to operate a managerial change, presented as the ultimate goal of the three activity fields related with CIS.

Our contribution in this paper consists of an original component model for developing adaptable software. At first, we do a brief state of the art on reuse and about de-coupling between components which constitute the two paradigms on which our works are based. Next, we will present our component model and we will position it with regard to the reuse paradigm and with related works. We will follow using this model in

order to construct a graph of interactions between components. We will finish with a presentation of the distributed adaptable software architecture allowing to implement the graph and therefore to integrate the components.

1 State of the art

1.1 The reuse

We propose a brief overview of the different activities linked to the reuse. The interested reader can get details in [CAUV99]. To provide the reuse, we need methods and techniques for reusable component engineering (design for reuse) and for system engineering by reuse of components (design by reuse).

1.1.1 Reusable components engineering

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. Today, several methods of domain analysis exist [NEIG84] [CAMP90] [KANG90][BAIL92][WART92][SEMM98] with strong differences about tools used, processes and results provided.

The reuse approach needs models of components representation. Two principles are essential : the abstraction principle and the variability principle.

The organization and the implementation of reusable components concerns the design and the implementation of the infrastructure for reuse. We can point out two main approaches for the design of architecture for reuse.

The static approach defines a static organization of the components. This organization describes the whole process of research and/or composition of component. It is widely used with components library in which the heritage defines statically a link between classes. This approach is used with frameworks which pre-defines the structure of the final product.

The dynamic approach consists of defining an architecture for reuse as the coupling of a set of components and a process which automates and/or guides the construction of a product by reuse of components. With this approach, we do not pre-define the research of components or their composition. We construct dynamically the product according to the specifications of the system currently developed.

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. We will now focus on this last point.

1.1.2 System engineering by reuse of components

Engineering by reuse of components consists of using a specific architecture to produce the system. This engineering may be very different relative to the available architecture.

We have to notice that the engineering of reusable components and engineering of systems by reuse of components are still considered as two different independent activities on which 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].

The first activity is the research and the selection of components. It consists of researching into the library one component which corresponds to the specific problem of development. In most of situations, there is not only one candidate but several. Therefore, a comparison is needed to select the one which corresponds more to the needs expressed.

The adaptation and the integration of components constitute the second activity. Adaptation techniques are numerous : modification of the component, instantiation mechanism, parameterization and specialization. With some approaches, the adaptation may be guided if the reuse system proposes alternatives with arguments helping the choice. The reuse becomes effective if the component is integrated to the product currently developed. This integration problem can be simple if the development language and the specification language of components are the same : the integration is done with communication mechanisms

integrated to a common language. It becomes more difficult if the development and the specification languages are different.

The problems of these different activities are complex. Today, works in this domain are dispatched. Few solutions realize an integration to propose a systematic reuse approach.

This synthetic presentation of reuse allows us to show the wide research field with this domain. It has also shown the variety of approaches proposed to answer sometime to the same preoccupations. It allows one to bring to the fore new problems to study : the definition of components model respecting the abstraction and the variability principles, the proposition of component-based methodologies, and the development of tools allowing the development by reuse.

1.2 The principle of de-coupling between components

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. In this part, we will present the principle of de-coupling which we are working on.

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. Therefore, this approach can be seen as a superior paradigm to the objects one [PESC00].

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. Each output mark is potentially linked with a connection to an input mark of another component to constitute a graph of interconnections. The general idea is to provide a communication abstraction independent of operational choices.

Absence of references to the called entity in the definition of the calling entity contains two major advantages. On the first hand, the communication mechanism authorizes modification of graph connections without modifying components. The encapsulation principle is respected. On the second hand, components become entities potentially reusable if their definition is independent to their software environment. This aspect is interesting because of our objectives shown in the introduction.

Approach with components is not totally opposed to the object world, even if it often tries to solve problems inherent to the whole object. The coupling is one of the more significant points. In the object world, when an object communicates with another, implicitly with a message, a structural and a behavioral coupling appears. An exchange of data and an exchange of control takes place. Approach with components tries to loosen this coupling. With a static point of view, a component does not reference directly with another component. With a dynamic point of view, the message passing between components does not imply the control passing. As an illustration, 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. Firstly, the data marks allow communication of data between components. Secondly, control marks are used to provide control.

A bigger freedom to express control results. Therefore, it is possible to manage active and independent components. This last point is very interesting in relation to our objective. This new powerful expression is provided with the graph of interconnections and no longer into the internal code of the component. As a consequence, components become 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.

2 The points of interaction

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 model, then its use, we want to have an interlaced approach.

Our 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.

Our component 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.

2.1 Input/output information points

Intuitively, Input Information Points (IIP) are acquisition points for a component. On the contrary, Output Information Points (OIP) are points to reconstitute. The following schemas illustrate this with a graphical representation associated to our component model :

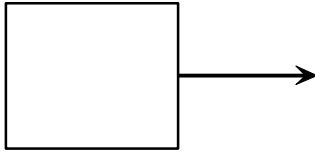


Figure 1 : OIP of a component

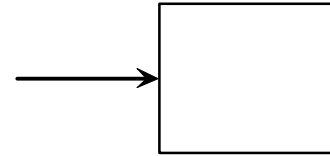


Figure 2 : IIP of a component

2.2 Control points

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

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). The following schemas show the corresponding graphical representation.

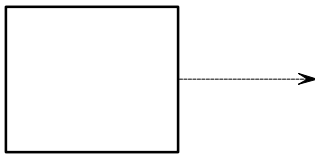


Figure 3 : SEP of a component

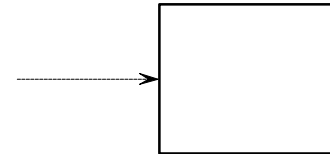


Figure 4 : SRP of a component

Resource sharing points

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. The following schema show the graphical representation.

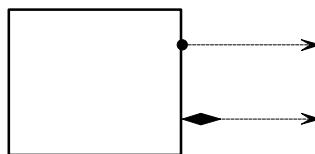


Figure 5 : RAP (upper arrow) and RRP (lower arrow)

In our presentation of components, we focalized on the graphical representation associated to introduce the following part which deals with its use. Nevertheless, we have to note that we have developed a description language to describe the different interaction points.

2.3 Illustration : the European ASIMIL project

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.

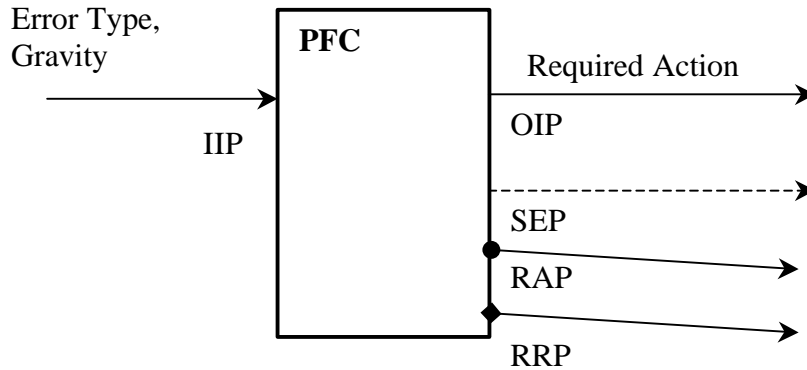


Figure 6 : Procedure Follow-up Component

- 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.

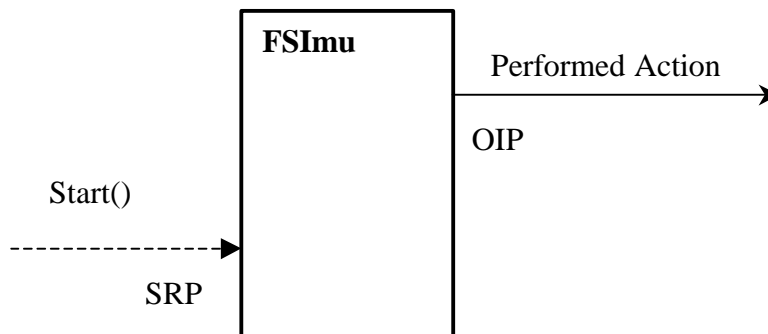


Figure 7 : Flight Simulator

- 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)

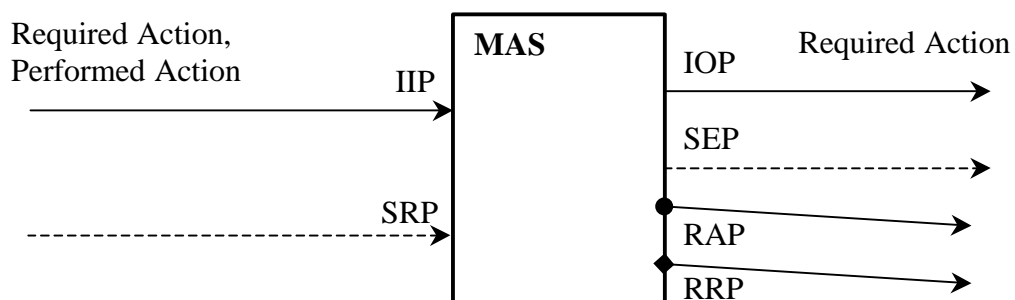


Figure 8 : MultiAgentSystem

- **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.

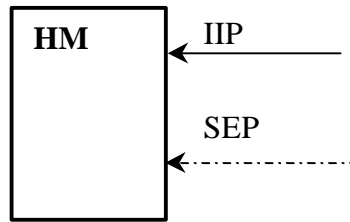


Figure 9 : Historic Manager

3 The graph of interaction

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.

Before entering into details, we have to notice that we follow the idea that components integration consists of making a de-coupling as strong as possible between components. With a general point of view, this idea is interesting because the reuse can be more systematic, a bigger number of components may be candidates to the reuse. With a practical point of view, this approach is absolutely necessary with our application field.

3.1 Information interactions

We can transfer information between components. This transfer may be preceded by the Building of Information (BI).

Concerning the transfer, we can point out two cases. The first one is the stream transfer [SHAW96]. The OIP of a component is linked to the IIP of another component.

The second one 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 : continuous flow transfer

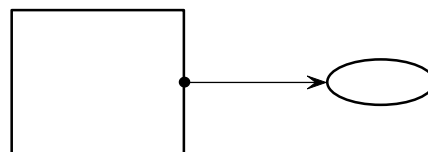


Figure 11 : mailbox transfer

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.

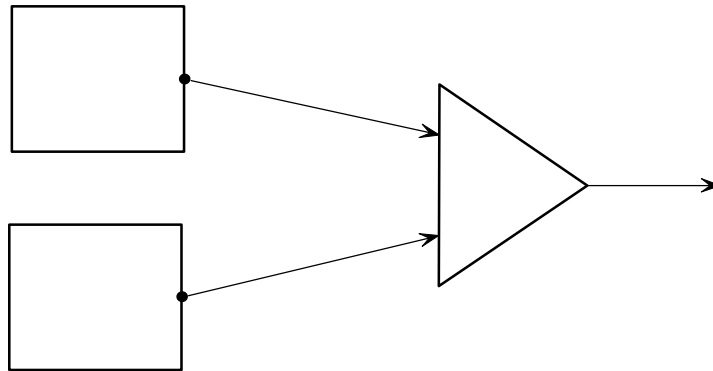


Figure 12 : building of information from two pieces of information from two components

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 devices, not enough to process the variety of problems.

If we compare our approach with [MEUR01], we note that our proposition is richer than theirs in terms of information interactions. In fact, we add the mailbox transfer at the stream transfer. The experience we got with enterprises convinced us of this need. Moreover, we offer the possibility to build “ad hoc” interactions facilitating a lot the integration of components.

3.2 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.

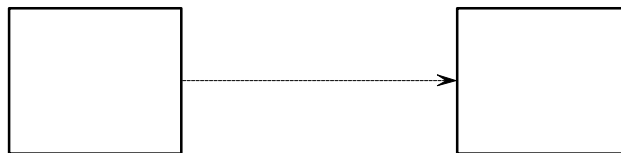


Figure 13 : synchronization of two components

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 :

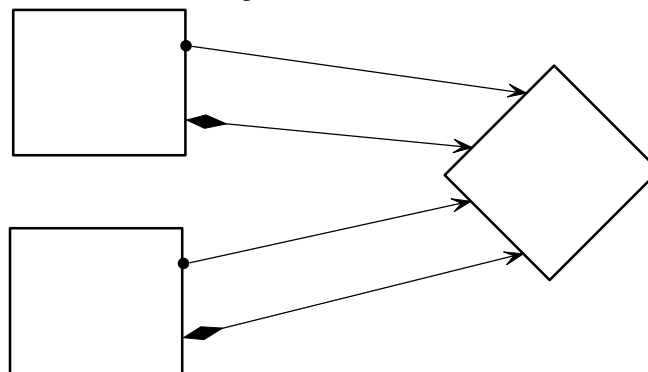


Figure 14 : resource sharing between two components

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.

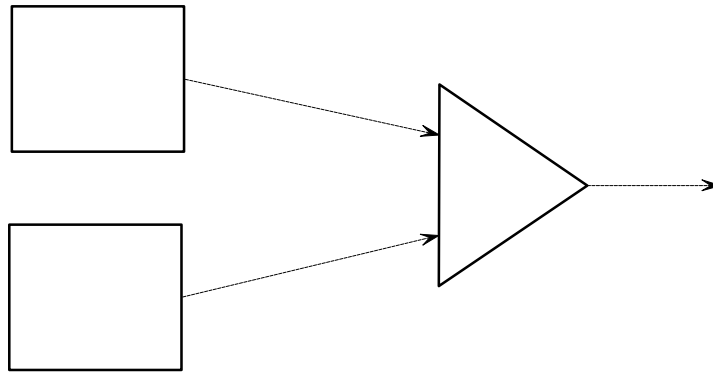


Figure 15 : complex interaction control

3.3 Mixed interactions

We had the idea to generalize the building of information and the complex control interactions. To do that, we propose mixed interaction where information and control can be mixed together. Its goal is to provide a piece of information or an element of control from both a piece of information and/or an element of control.

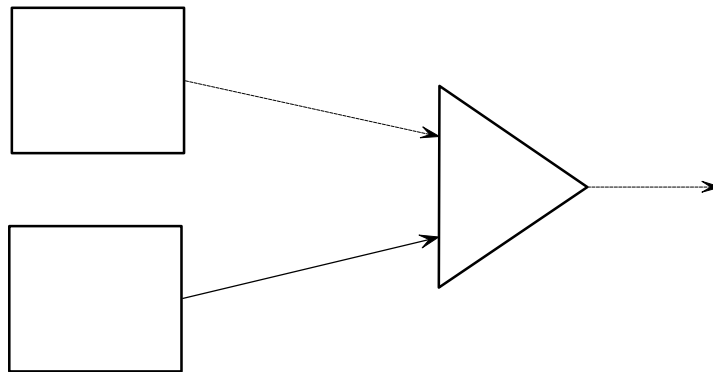


Figure 16 : mixed interactions

After we have provided all needed interactions to solve a problem, we obtain a graph where each component represents a node. During the work to integrate components we create other nodes as we showed previously. The set of these nodes and the totality of arcs express the level of the de-coupling between components. If we compare graphs obtained with our approach and interconnection graphs obtained with MALEVA, we can measure the effort we made to manage dependencies between components and put them on the graph. This allows to modify substantially the application without intervening on components. With regard to the reuse paradigm, we situate our works clearly on a reverse-engineering approach based on “components/connection”.

3.4 Illustration : the integration of ASIMIL components

This part shows the integration of the ASIMIL components in order to develop the ASIMIL application. This integration is represented as a graph of interaction (Figure 17) which uses pre-defined interaction (continuous flow transfer, mailbox transfer, synchronization and resource sharing) and dedicated interactions.

PFC (Figure 6) and MAS (figure 8) 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. To solve the problem, we use a *dedicated interaction* (BI - Building of Information) built with two pieces of information (from PFC and Fsimu) to provide an other piece of information needed by MAS.

HM (figure 9) has an asynchronous operating mode. Thus we use a *mailbox* in which HM gets messages sent by MAS.

ASIMIL application is managed by PFC which starts all the components using *synchronization* interactions.

The window, in which PFC shows an error message if the action of the trainee is not required, must be shared with MAS with a *resource sharing* interaction.

We can note that all the interaction points of the interfaces are not necessary used in the graph of interaction (see MAS for example). This is an illustration of the genericity of the components. Each component is reusable and adaptable to different applications through its points of interaction.

Control devices are the mailbox, the synchronization and the resources sharing. Their implementation must be realized within an heterogeneous and distributed environment [BOUG98]. This point has not been implemented yet. We are currently studying numerous technologies available to choose the best suited to our works.

The transfer device concerns exclusively the transfer of information. This is a critical point with distributed systems. This is the role of the communication manager.

The exchange device is the interface between the architecture and the components. This role is dedicated to component managers allowing the coupling between the components and the architecture.

Conclusion

This paper seems to us an interesting contribution for component-based software development. From general point of view, we offer a global solution from the component model to the software architecture. Moreover, we have the opportunity to test our component model on an industrial application, with the ASIMIL project.

If we detail a bit more, we have to focus on different points. First of all, our approach is based on a strong de-coupling between components. We have explained it in the first part. 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.

If we consider the component model, we can underline the variety of interaction points related to other works. This allows a rich activity of component specification, adapted to the problems we propose to deal with. Concerning the construction of the graph of interaction, numerous basic devices 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 architecture warranty the implementation of the graph of interaction. 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), a “light” version of CORBA for Java environment. 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

- [ANIO01a] ANIORTE P. - *Engineering of distributed systems : a component-based approach rested on an architecture for interoperability* - Proceedings of the 2001 International Symposium on Information Systems and Engineering (ISE'2001), Published by Computer Science Research, Education & Applications Press (CSREA : USA Federal EIN # 58-2171953), Las Vegas, USA, June 25-28 2001
- [ANIO01b] ANIORTE P. - *A method and tools to migrate applications to distributed systems* - Proceedings of the 7th International Conference on Reengineering Technologies for Information Systems (ReTIS), Published by the Austrian Computer Society, Lyon, France, July 4-6 2001
- [ASIM02] Presentation of ASIMIL project -<http://www.cordis.lu/ist/projects/99-11286.htm>
- [BAIL92] BAILIN S. - *Domain analysis with Kaptur* - Courses notes, CTA Inc. Rockville, MD20852, 1992
- [BOUG98] BOUGUETTAYA A., BENATALLAH B., ELMAGARMID A. - *Interconnecting heterogeneous information systems* - Editions Kluwer Academic Publishers, 1998
- [CALD91] CALDERIA G., BASILI R.V. - *Identifying and qualifying reusable software components* - IEEE computer, February 1991
- [CAMP90] CAMPBELLG., FAULK S., WEISS D. - *Introduction to synthesis* - Technical report intro_synthesis-90019-N, Software productivity Consortium, June 1990
- [CAST92] CASTANO S., DE ANTONELLIS V. - *A model for reusable requirements* - Esprit project, report pdm 2-1-3-r1, 1992
- [CAUV99] CAUVET C., SEMMAK F. - *La réutilisation dans l'ingénierie des systèmes d'information* - Dans Génie objet - Sous la direction de OUSSALAH C., Hermès, 1999

- [DEMI97] DE MICHELIS G., DUBOIS E., JARKE M., MATTHES F., MYLOPOULOS J., POHL K., SCHMIDT J., WOO C., YU E. - Cooperative Information Systems : A manifesto - In Cooperative Information System : Trends and directions, PAPA ZOGLOU M.P., SCHLAGETERG. Editors, Academic Press, 1997
- [HCSS01] HCSS (High Confidence Software and Systems) Coordinating Group. "HCSS Research needs : a White Paper". Interagency Working Group in Information Technology Research and Development (IWG/IT R&D), White House National Science and Technology Concil, USA, 2001
- [HERI01]HERIN D., ESPINASSE B., ANDONOFF E., HANACHI C. - *Des systèmes d'information coopératifs aux agents informationnels* - Dans « Ingénierie des systèmes d'information » sous la direction de CAUVET C. et ROSENTHAL-SABROUX C., Hermès, 2001
- [KANG90] KANG K., COHEN S., HESS J., NOVAK W., PETERSON S. - *Feature-Oriented Domain Analysis (FODA)* - Feasability study CMU/SEI-90-TR-21, software engineering institute, Carnegie-Mellon university, Pittsburgh, Pennsylvania, 1990
- [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
- [NEIG84] NEIGHBORS J.M. - *The DRACCO approach to constructing software from reusable components* - IEEE transactions on software engineering SE-10(5), p.564-574, 1984
- [PESC00] PESCHANSKI F., MEURISSE T., BRIOT J.P. - *Les composants logiciels : Evolution technologique ou nouveau paradigme ?* - Conférence OCM 2000, 2000, p. 53-65
- [SEMM98] SEMMAK F. - *Réutilisation de composants de domaine dans la conception des systèmes d'information* - Thèse de doctorat de l'Université Paris I – Février 1998
- [SHAW96] SHAW M., GARLAN D. - *Software architecture : Perspective on an emerging discipline* - Prentice Hall, 1996
- [SING98] SINGH N. - *Unifying heterogeneous information models* - Communications of the ACM, vol. 41, n°5, 1998
- [SIRA00] : *Projet SIRAC : Systèmes Informatiques Répartis pour Applications Coopératives* - Rapport d'Activité 2000, INRIA
- [WART92] WARTIK S., PRIETO-DIAZ R. - *Criteria for comparing domain analysis approaches* - International journal of software engineering and knowledge engineering 2(3), p.403-431, 1992