Distributed software : from component model to software architecture

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

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 prolonging life of heterogeneous distributed systems with the help of component reuse. Such systems need a distributed adaptable software architecture to be implemented. In this paper, we propose a component model for redeveloping 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*

Keywords. Distributed platforms, middleware, component model, reuse, integration, interoperability

1. Introduction

Technological and economical mutations of these last years have really modified the life of organisations. 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 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 two previous points deal with interoperability. More and more, the idea of interoperability becomes a need to answer to new organisational 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 co operate. 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, co-operation. The difficulty lies in the cross feature

of our approach. Indeed, it is the actual trend. For example, CIS (Co-operation 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), modelling 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 co-ordination 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 redeveloping 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, and the metamodelling domain. We will follow using the component model in order to construct a graph of interactions between components. We will finish with a presentation of the "Framework" allowing to implement the graph and therefore to integrate the components.

2. State of the art

2.1. The reuse

To provide the reuse, we need methods and techniques for reusable component engineering (design for reuse) and for systems engineering by reuse of components (design by reuse) [CAUV99].

The reusable components engineering covers the identification and the specification of components, their organisation and their implementation. Two principles are essential in the model representation of components needed by the activity of identification and specification : the abstraction principle and the variability principle. The organisation and the implementation of reusable components concerns the design and the implementation of the infrastructure for reuse.

The architecture of reuse constitutes a coupling tool between the two essential activities of the reuse, the engineering of components and the engineering of systems. The architecture of reuse consists in using a specific architecture to produce the system. The engineering of reusable components and engineering of systems by reuse of components are still considered as two different independent activities. With the interlaced organisation used in the CARE system [CALD91], we use components to produce an application, and also we contributes to the development of those components. The first activity consists in researching into the library one component which corresponds to the specific problem of development. The adaptation: modification, instantiation, parameterisation, specialisation and the integration of components constitute the second activity. This integration problem becomes difficult if the development and the specification languages are different.

2.2. The principle of de-coupling between components

The "component" paradigm is emerging in a specific domain of software components. Component-based approach is based on the principle of de-coupling between components. According to this approach, dependencies between entities are no more used into these entities, but are defined outside the components [PESC00]. For example, in the component model MALEVA [MEUR01], components are specified by input/output marks potentially linked with a connection to another component, and constitute a graph of interconnections. The general idea consists in providing a communication abstraction independent of operational choice. Absence of references to the called entity in the definition of the calling entity allows to respect the encapsulation principle, and the communication mechanism authorises modification of connections between components without modifying components.

When an object communicates with another, by a message, a structural and a behavioural coupling appears implicitly. A component does not reference directly another component, and the message passing between components does not imply the control passing [LHUI98]. Therefore, it is possible to manage active, independent, and generic components because a part of the control description is already included into the graph of interconnection. The distinction between the data flow and the control flow 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.

2.3. Metamodeling: from meta-model to applications

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

The approach proposed by OMG [OMG02] with Model Driven Architecture (MDA) [MDA02] 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 capitalise and reuse models of application instead of regularly migrate software components from a middleware to another. Then, only conceptual model like UML models can be capitalised during the full software lifecycle. Those conceptual models can be the only reusable components of a software. In Model Driven Architecture, software components frame are automatically generated by a mapping of model in a target middleware.

MDA is build on the four-level representation model of OMG:

- meta-meta level, which offers Meta Object Facility as standard meta-meta – model.
- meta level, where meta-models are all defined as MOF instances, in order to allow exchange and interoperability. Those meta-models define model language which are used to describe models. OMG has defined, on this level, UML language which is a meta model instance of MOF.
- Class level, where models are described with an instance of a meta-model. For example, a UML model, issued from the specification of a software component is defined on this level as an instance of the meta-model of UML
- Instance level where are built instances of class model

For creating MDA-based applications, the first step is to create a Platform Independent Model (PIM), which should be express in UML. 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). 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.

3. 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 formalisation 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, organisation 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.

3.1. 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 resituate. The following schemas illustrate this with a graphical representation associated to our component model :

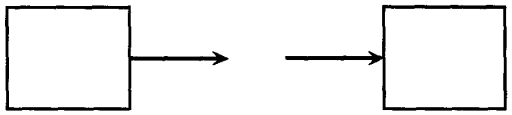

Figure 1. OIP of a component, IIP of a component

3.2. Control Points

The component model MALEVA only offers one type of control marks. We propose several

types of control points : synchronisation points and resource sharing points

3.3. Synchronisation points

Synchronisation points are dedicated to the synchronisation of components. When a component synchronises another one, the first one is called "the synchroniser" and the second one is called "the synchronised". The synchroniser has a Signal Emission Point (SEP) whereas the synchronised has a Signal Reception Point (SRP). The following schemas show the corresponding graphical representation.

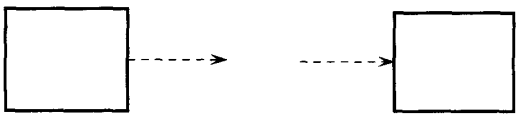

Figure 2. SEP of a component, SRP of a component

3.4. 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 synchronisation that we have previously presented, appear in works dealing with co-ordination. 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..

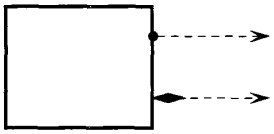

Figure 3: RAP (upper arrow) and RRP (lower arrow)

3.5.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) 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 synchronise 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.

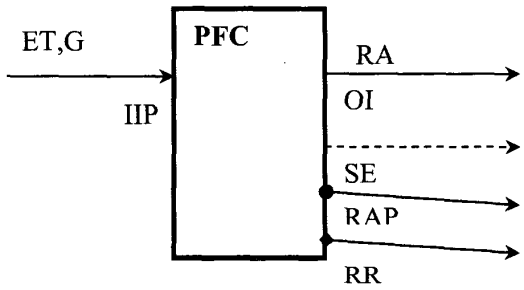

Figure 4:Procedure Follow-up Component

Every action performed (PA) 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.

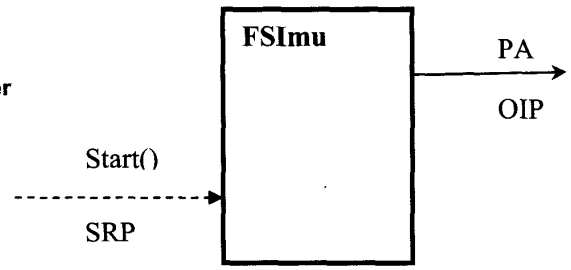

Figure 5:Flight Simulator

Multi Agent System: 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 analysed . When an error is detected an characterised , 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).

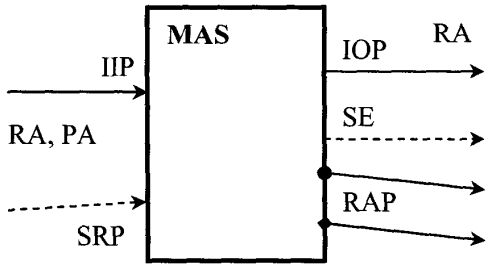

Figure 6: Multi Agent System

The component History Manager 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.

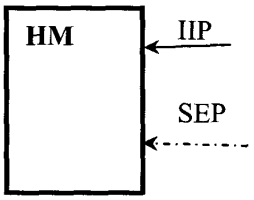

Figure 7: HistoryManager

4. 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. 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.

4.1. 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 8: continuous flow transfer

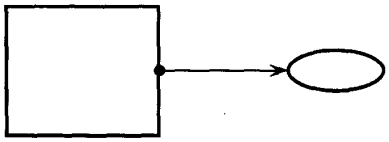

Figure 9: mailbox transfer

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.

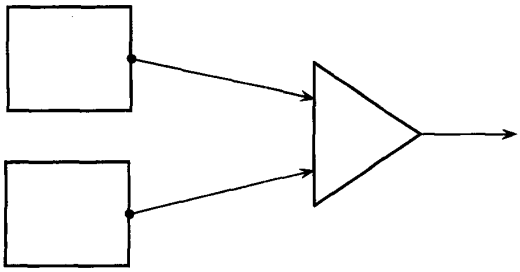

Figure 10: building of information from two pieces of information from two components

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.

4.2. Control interactions

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

Figure 11: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 :

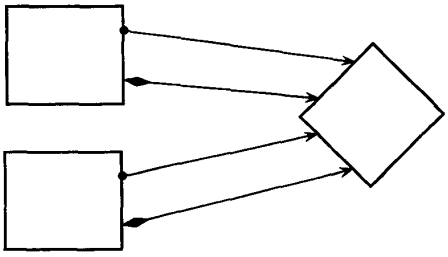

Figure 12: resource sharing between two components

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.

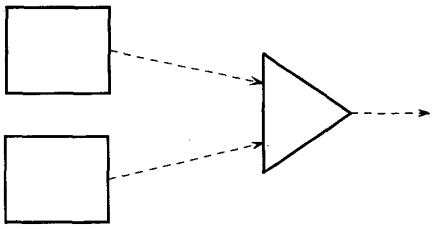

Figure 13: complex interaction control

4.3. Mixed interaction

We had the idea to generalise 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.

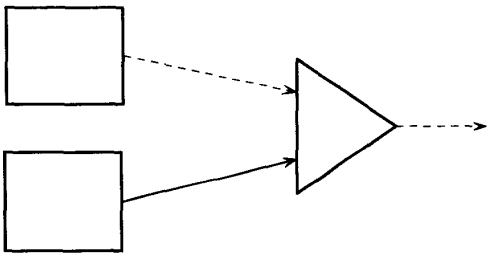

Figure 14 : mixed interactions

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".

4.4. Illustration : the integration of ASIMIL components

This part shows the integration of the ASIMIL components in order to develop the ASIMIL application. This integration in represented as a graph of interaction (Figure 15) which uses pre-defined interaction (continuous flow transfer, mailbox transfer, synchronisation and resource sharing) and dedicated interactions.

PFC (Figure 1) and MAS (Figure 6) 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 (Figure 5) to provide an other piece of information needed by MAS.

HM (Figure 7) 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 "synchronisation 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 generics of the components. Each component is reusable and adaptable to different applications through its points of interaction.

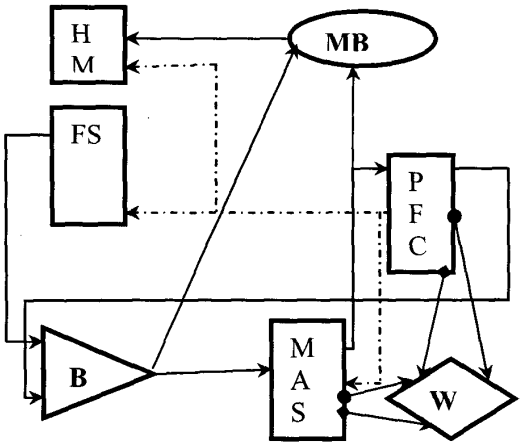

Figure 15 : graph of interaction of the ASIMIL application

5. A framework of Components

Our objective is to build a framework allowing the architect of the distributed application to construct a graph of interaction with the help of the tools provided by the platform. The specifications of this graph or architecture must be middleware-independent.

We have decided to use MDA specification, to express our component model with UML notation with defining a UML Profile, to store this meta model of components on the meta level of MDA as a core model, to instanciate this model on the class level in order to build a Platform Independant Model of the application.

5.1. Uml profile

The UML Profile for our component model called UPPA is designed to provide a standard means for expressing its semantic using UML notation and thus to support expressing these semantics with UML tools. It is a way for us to express our model with the help of a standard language. When one wishes to represent a UPPA type via UML notation, the usual approach is to model it as a Classifier and to stereotype the Classifier to indicate whether it represents an Component, a IIOP, etc. This is a legitimate approach, since a Stereotype is one of UML's official extension mechanisms. The first step is to define core elements of our model as UPPAComponent, UPPAOIP, UPPAIIP,UPPA,

UPPAInteraction. The second step is to define tools we propose to allow the integration of the components, as UPPADataInteraction, UPPAControlInteraction, UPPAMixedIntercation

5.2. Platform Independant model

The architect of the application can construct a Platform Independent Model with connecting each component with another. The architect can also implement the behaviour of UPPA ad'hoc interaction with the help of UML language

5.3. Platform Specific model

Platform Specific Model is the mapping of the Platform Independent Model to the target middleware (CORBA, XML/SOAP, EJB, our specific middleware). This step which can be called integration step must automatically generate the wrapper of each component defined.

5.4. A specific middlware

We have decided to implement our specific middleware, which is the target of the mapping step of the PSM.

At the implementation level, we have to point out operative devices, control devices, a transfer device and an exchange device. With operative devices, we mean the building of information, the complex control interaction and the mixed interaction. All three have as a common point the production of a piece of information or an element of control after a process expressed with an algorithm. These three devices are implemented as autonomous and parallel tasks with Java. Actually, a prototype of a building of information device is under development. This prototype will connect three ASIMIL components.

Control devices are the mailbox, the synchronisation and the resources sharing. Their implementation must be realised 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.

6.Conclusion

This paper seems to us an interesting contribution for the redevelopment of distributed software. From general point of view, we offer a global solution from the component model to the implementation. 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 co-operative 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 co-operative 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 Java Beans. Interoperability between each Bean is realised 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.

9. References

[1] 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

[2] [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, PAPAZOGLOU 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

[POOL02] POOLE J D, Model Driven Architecture: Vision, Standards and Emerging Technologies, ECOOP 2001 ,2001

[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