Really Hard Time developing Hard Real Time
Research Activities on Component-based Distributed RT/E Systems

Etienne BORDE[1,2], Grégory HAIK[1], Virginie WATINE[1], Laurent PAUTET[2]

1 : THALES Land and Joint Systems, 5 avenue CARNOT 91883 Massy cedex France
{firstname.lastname@fr.thalesgroup.com}

2 : Ecole Nationale Supérieure de Télécommunications, 46 rue BARRAULT, 75634 Paris cedex France
{firstname.lastname@enst.fr}

***Abstract**: The development process of distributed real-time embedded systems (DRES) suffers significant limitations when addressing the antagonistic concerns of systems interoperability, flexibility, and reliability. In this paper, we first present a component-based development process and related architecture designed to enable DRES interoperability while improving developer productivity. We then describe the techniques this process uses in order to improve reliability of these systems. The component-based framework is illustrated by a practical use case. Lastly, we present research orientations addressing verification, validation, and certifiability on the one hand, and their ability to tackle with the always-growing flexibility requirements on the other hand.*

As real time embedded systems complexity is growing every year, industry software architects are facing major challenges in terms of development productivity. This issue is addressed by component-based development methodologies, as proposed by the software engineering academic community [1]. Automatic deployment and configuration, code generation, code reuse, better testability and early validation, reduced time to market, easy tuning and mitigating integration risks are some of the benefits of component-based development (CBD). In mid 1990, industry standards have been issued for applying CBD to large scale information systems (Sun's Enterprise Java Beans – EJB [2], OMG's CORBA Component Model – CCM [3]). Unfortunately, CCM and EJB frameworks are not directly applicable to distributed real time embedded systems: scarce computing and memory resources, hardware heterogeneity, real time constraints, performances and assurance issues make CCM and EJB irrelevant for development of DRE systems. Consequently, DRES software engineering community has performed extensive research for adapting component-based development to DRE systems. THALES has been involved at the OMG for defining a variant of CCM that may cover the technical requirements of embedded real time component-based frameworks. The reason of choosing CCM as a starting point was the suitability of OMG's philosophy with the requirements of THALES' clients in terms of interoperability and standard openness. Thus, the main challenge was to make CCM efficient, predictable and low footprint. Moreover, since non functional requirements of DRE systems are very versatile from one system to the other, it is crucial for a DRES component-based framework to be highly adaptable and configurable. Note that a naïve approach to adaptability may deeply impede interoperability.

Research conducted by THALES during the last four years has led to a component framework that is usable and beneficial for industry-standard real time embedded systems. Ongoing research is performed for addressing more application domains, particularly safety-critical, mission-critical or security-critical certified applications requiring deep verification and validation on the one hand, and reconfigurable and multi-mission systems on the other hand.

This paper will first present the outcome of THALES' collaborative research effort performed during the last four years (section 1). The benefits of the resulting component-based framework, namely MyCCM, are illustrated on a real world program MyCCM is being applied to (section 2). Finally, an overview of current research activities is provided in section 3.

1. Component-based Framework for Distributed RT/E Systems

This section will first present the authors' viewpoint on component-orientation, and then describe the principles and the outcome of adapting component-based development to real time embedded systems as performed in the scope of collaborative projects.

1.1. Component-Orientation

Principles

Component-orientation has been introduced to help managing the increasing complexity that Information Systems were facing and to increase the productivity of their development. A component is basically a piece of functionality that can be assembled with others in order to provide the full functional coverage of the system. Allowing to break down the whole system in smaller really independently manageable pieces, easier to develop and to reuse, makes it much cheaper to develop and integrate.

To achieve that goal, components come with three main characteristics:

- A self-described packaging format, so that components can be deployed and configured externally from the application.
- The explicit description, by means of ports, of not only the services that the component is providing (as an object does), but also of the ones it requires for functioning, to be provided by other components; this allows the components being connected externally from the application.
- The separation of the business logics (the components themselves) from the relevant[1] technical support that the infrastructure is providing to them and which is under the responsibility of the containers, parts of the infrastructure aiming at hosting the components.

Even not always put forward, this "separation of concerns" is a key factor to master complexity and to allow an effective reuse, hardly achievable with only object orientation. Actually the two more important factors that prevent reuse are:

- Inability to master the dependencies between a piece of software and others: component model – making this explicit – provides a way to tackle this.
- Inability to master dependencies between a piece of software and its underlying infrastructure: Component/Container model offers a way of structuring this. As a side effect, this model allows to guarantee that the supported technical properties are enforced consistently across the application (e.g., the access control policy is guaranteed to be applied to all components)

Of course, the work to build the appropriate containers is no more of the application developer's responsibility. Component-based infrastructure (e.g., EJB – see below) are thus coming with the tooling allowing to automatically generate the containers, based on standardised descriptions.

This model has been implemented as support for Information Systems. The first consistent implementation has been Sun's EJB (Enterprise Java Beans) which is a real success in this specific application domain. EJB offers support for technical services that are meaningful in this area (Persistence support, Transactional support and Security – to be understood as access control) and is built on top the Java infrastructure.

CORBA CCM (CORBA Component Model) is a more generic and distributed specification of this model, even if still fully dedicated to Information Systems (it offers support for the same technical services as EJB). However, as the general trend of CORBA is to focus more on technical systems[2], CCM is moving in the same direction. This move has already started at OMG, with newly adopted Lightweight CCM specification that defines a CCM profile compatible with embedded targets.

With .NET, Microsoft has also adopted this model, even if the proprietary solution it proposes allows a slightly different, more intrusive and less structured, way of organising the software.

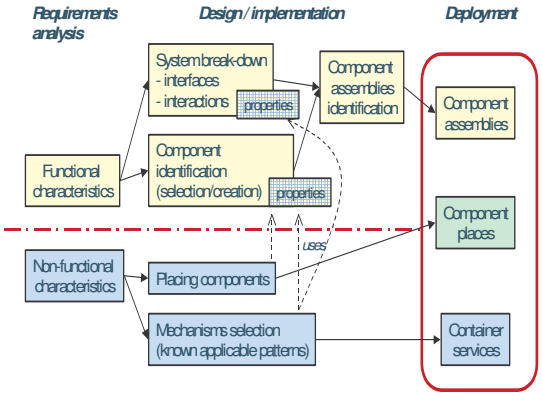

Figure 1: Impact of separation of concerns on development process

Impacts on the Development Process

Separating the functional properties from the non-functional ones and building the whole system by means of assembly, deployment and configuration of components have huge positive impacts on the development process itself, whose main phases are featured on Figure 1:

[1] What is the relevant technical support depends on the application domain.

[2] Cf. all the new specification in the area: Real Time CORBA, Minimum CORBA – special profile for embedded systems, Fault Tolerant CORBA, Data Parallel CORBA...

- It allows a clean role separation: domain experts may focus their functional expertise while platform specialists may define and implement the best technical support.
- Component reuse is made possible.
- As technical support realisation is mainly achieved based on component configuration, main design choices (such as component localisation) can be delayed until the latest and therefore adjusted during the integration, making that costly phase much easier.

1.2. Adaptation to Real-time and Embedded Systems

All the positive impacts of component-orientation would be also very desirable for RT/E systems. However current implementations are not suitable for those systems since they don't provide the relevant non-functional support nor the adequate interaction modes between components and are not meant to accommodate resource constraints, which are here of prime importance.

THALES has therefore decided to adapt an existing standard component model to specific constraints of RT/E systems and to define a dedicated framework. As basis, OMG's Lw-CCM has been selected and its adaptation initiated in the scope of two research projects (IST/COMPARE [7] and ITEA/MERCED [8]).

One important characteristic of the RT/E area is the huge variety of its systems. Trying to provide in a unique framework a solution that would fit all the RT/E systems is not achievable, nor desirable. Therefore, rather than just adding the technical support that would allow to develop the use-cases parts of those projects, it has been decided to focus on extensibility and usability.

Extensibility is achieved by providing the ability to plug within the container, technical support providers, called container services. For that purpose, has been defined an open architecture for the container with specification of dedicated interfaces to insert and configure the container services as well as a packaging format to enable their deployment.

Usability for RT/E systems is primarily conditioned by the ability to get adequate interaction between components. Actually proper interaction support is crucial for it conditions the ability of designing components and the existing one was far from being enough. A new construct, named connector, has been introduced to capture the interaction style. Connectors can then be developed for as many interaction styles as needed. Connectors are actually made of fragments, which are considered as specific container services and can be deployed and configured as such.

Usability also depends on the ability of the framework to accommodate stringent resource requirements. Key elements for that purpose are (i) the ability to tailor the containers at their minimum functional coverage by plugging in only what is strictly needed, (ii) the platform isolation provided by the containers, which allows the use of a huge variety of technical services (including small ones if needed) as well as (iii) the implementation of specific trade-offs to get the better use of the scarce resources. As a proof of concept of the ability to accommodate various platforms, two implementations of the framework have been developed during those research projects: the first one on top of RT-CORBA, the second one on top of OSEK-VDX.

In the scope of these two projects, interaction support for the use-cases has been implemented, as well as some container services providing basic development facilities (tracing...) and timing properties enforcement (locking protocols and appropriate setting of POSIX scheduling parameters).

1.3. MyCCM

Based on these research results, THALES has started to industrialise a framework, called MyCCM (Make Your CCM). MyCCM building blocks are represented on **Figure 2**.

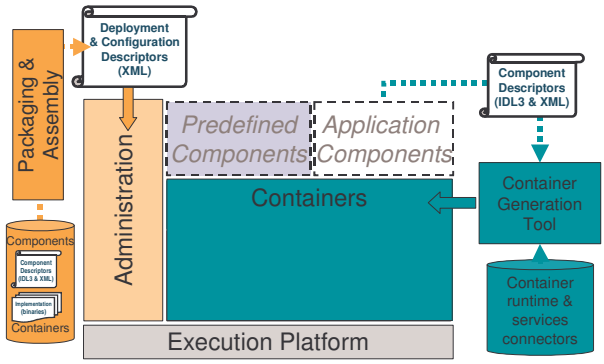

Figure 2: MyCCM building blocks

The central part represents MyCCM runtime, that is mainly two folds: (i) infrastructure and its services mediated by the containers, which in return are hosting structures for the components, and (ii) a tool called Administration, dedicated to deployment, configuration and connection of components – and later on, monitoring and control. On the right side is represented the container generation tool, which generates the containers and

component envelopes[3] based on component description (IDL3 and XML files); on the left side, the packaging and assembly tool which prepares what the Administration needs as input, namely the component packages and the deployment plan.

Adaptation to a particular domain

As it can be seen, the framework is a combination of runtime support and associated tooling. Porting it to a given platform will affect not only the runtime support, but the generated code as well. Therefore attention has been paid to make the code generation mechanisms as flexible as possible.

Besides porting to the target platform, adapting the framework to a specific domain consists just in ***selecting or defining and implementing proper container services and connectors***.

Real Time Issues

Container services are used for configuring time and scheduling parameters of the application, while other services provide real time locking mechanisms. This way, the software architect addresses all the scheduling issues in configuration files. The corresponding activation model is illustrated on **Figure 3**: threads, that may be either ***periodic*** or "***one shot***" are associated to components entry points. One shot threads are meant for interrupt handling as well as any input I/O functions: where necessary, the architect connects one shot threads to a never ending incoming event handler encapsulated in the receiving component.

MyCCM also enables the software architect to set the scheduling parameters of the threads handling the framework-supported communication mechanisms. Such communication threads may have either a (i) ***static*** scheduling configuration as given by the architect, a (ii) ***dynamic*** configuration where scheduling parameters are inherited from the calling activity or finally (iii) ***no-configuration*** – applicable only to components in the same address space – where the calling thread performs the local method invocation. MyCCM uses the underlying OS and middleware (POSIX and RT-CORBA for instance) for implementing these policies.

[3] A component envelope is a generated piece of code allowing a user-written piece of code to be hosted by a container. What is generally called 'component' is the business code (written by the application-developer) plus its envelope.

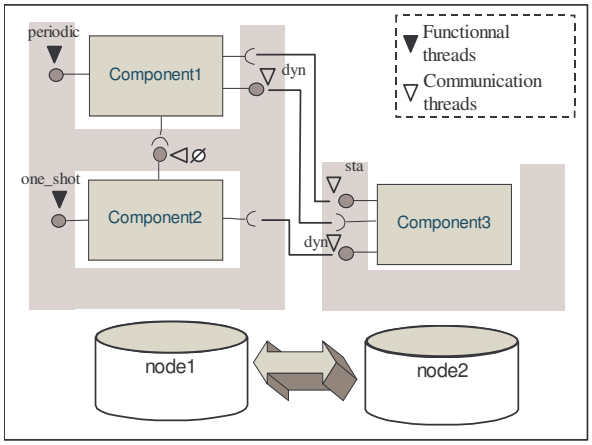

Figure 3 : Threads configurations

2. A Practical Use Case

Beyond experimentations and use case developments performed in the scope of research projects, MyCCM has been recently adopted by THALES for the development of an Infra-Red Search and Track (IRST) Signal Processor called ARTEMIS.

2.1. The ARTEMIS Program

ARTEMIS IRST system has been selected to equip the Future European Multi-Role Frigates (FREMM). It is mainly composed of three infra-red sensors – located around a mast – and a Signal Processor. Each sensor covers a third of the frigate horizon (120°). The three video streams are sent to the IRST Signal Processor (ISP) by a specific protocol on top of UDP. The ISP applies visualisation and tracking algorithms to incoming streams. Data processing is distributed on different single board computers. Each algorithm is encapsulated in a component that communicates either locally or remotely with its counterparts via the communication support of the component framework. For controlling and configuring the components, the framework relies on a free RT-CORBA middleware, while high bandwidth data streams are handled by a fast, unreliable implementation of CCM event support. Finally, processed video and tracks are sent to the frigate's Combat Management System using a dedicated protocol.

A module in a ***UML modeller*** enables the ISP software architect to design the application in a top down approach. ***Scheduling settings*** (priorities and periods) and parameters of components are defined ***in UML models as well***. The software architect may also define functional validation scenarios, or target performance measurement scenarios in the UML modeller. CCM deployment and configuration descriptors are then produced for

each validation scenario. On-target execution of these scenarios is handled automatically by MyCCM deployment mechanisms.

2.2. Benefits for the Software Architect

Usage of a component framework such as MyCCM has good consequences on ***productivity*** of software development. Several factors contribute to increasing productivity. First, MyCCM development process is based on ***intensive code generation***: configuration of scheduling parameters such as POSIX priorities and thread periods, priorities of RT-CORBA communication threads, data marshalling code and method dispatch, and configuration, deployment and connection of application components are simply generated from the architecture description provided as a set of UML models. With MyCCM, the program manager doesn't have to provision development efforts for these tasks. Additional cost reduction resides in the dramatic ***simplification of internal communication protocols***: with handcrafted communications, detailed specifications of datagram/stream formats are necessary; with a communication middleware such as CORBA, equivalent specifications are much more abstract, and consequently simpler. Specification of interfaces is done at operation level, with in/out parameters and exceptions, not at socket level. Moreover, ***generation of threading artefacts*** and ***deployment code*** makes a program relying on MyCCM clearly more productive than a regular CORBA or RT-CORBA application, for which only communication support is generated.

Integration of MyCCM with modelling tools is another productivity factor. Indeed, the software architect is not meant to write complex architectural descriptors by hand: definition of application architecture, design, configuration and deployment is rather done in a UML-based GUI. The first benefit is that graphical models constitute a ***key communication support*** between team members. Moreover, model-to-descriptor generators may come with potentially ***intensive model verification***, which leads to early discovery of model inconsistencies.

Another productivity factor is ***easier testability***: because MyCCM architecture enables the component developer to avoid platform dependencies, the functionalities of such platform-independent components can be validated not only on target, but also on development host. Components might be specifically enveloped and compiled for either the host or the target by just modifying few parameters of the UML architectural description, so that a ***complete functional validation*** can be performed on host.

Finally, integration costs and risks are significantly reduced by the ***late binding*** between application and the platform: when all software components are finally put together for final integration, it is almost inevitable that integration engineers have to fine tune the thread number and their associated parameters (period, priority…). With MyCCM, since all these parameters are in the models and not in the application code, engineers don't have to deeply analyse component code for finding out where threads are created and configured. Another typical situation where this late binding property mitigates integration risks is when hardware happens to be ill dimensioned with regard to software execution times and system timing requirements. Application components only have to be re-targeted (in UML models) to the new hardware. MyCCM will take into account the new hardware environment by generating some other technical code, with no impact on component implementations.

To sum up, component-based development might have very positive impact on productivity of software development. Still, our real-world experimentation with ARTEMIS program has not yet reached the point where productivity benefits might be factually assessed. This point will be reached by the end of all development and integration activities.

2.3. Current Limitations

There are key issues in DRES development that are not currently addressed by MyCCM. For instance, although MyCCM enables software architect to implement priority-based real time systems, no support is provided yet for taking advantage of RT-CORBA support for ***real time network protocols***. Network scheduling must be performed by other means. In today's MyCCM, priority inversion may occur in network stacks.

No support is provided either for proving that the system will meet its end-to-end requirements. This issue is generally referred to as schedulability. Another key issue is proving that the system is deadlock-free. Those two properties (schedulability and deadlock verification) are mandatory for ***high assurance systems***, might they be mission-critical, safety-critical or security-critical.

Yet another issue is that MyCCM only offers a programmatic support for reconfiguration: if needed, ***reconfiguration must be programmed by hand***. A descriptive, framework-supported reconfiguration mechanism would contribute to further increase productivity. Last but not least, ***multi-mission systems*** have specific requirements on software architecture that are not currently addressed by MyCCM.

Next section will discuss these issues and provide the reader with insights about ongoing research activities on these topics.

3. Ongoing Research Activities

The two directions followed by our research activities are verification and validation on the one hand, and support for greater flexibility on the other hand. While verification and validation address safety-critical, mission-critical or security-critical systems, research on flexibility addresses context-aware systems, fault tolerant systems, load balanced systems, autonomous systems and multi mission systems.

3.1. Verification & Validation

Worldwide academic community has performed extensive research towards verification of DRE systems. During the last decades, formal methods such as model checking, computational logics, schedulability algorithms and static code analysis have been foreseen to be able to provide the software architect with advanced tools and methods for verifying and validating DRE systems. Yet, ***these formal methods lack proper integration with the actual execution infrastructure***, making them more difficult to use efficiently. Numbers of properties might be verified, although resource dimensioning is under particular focus in the DRE domain.

Typical resources to be dimensioned are time, memory and energy. The following will focus on time.

Several tools are available to analyse real time properties of the system. Some are based on model checking techniques (UPAAL [14] for instance), others on algebraic methods inherited from Rate Monotonic Analysis, such as MAST [15] and Cheddar [17]. Not surprisingly, both methods have their respective limitations: model checking is subject to exponential state space explosion, while algebraic methods are not able to analyse all possible situations.

Research has been conducted to integrate model checking techniques with component-based development [11, 12, 13]. University of Cantabria (Spain), THALES and ENST (among others) are currently performing research towards integration of CBD and algebraic methods. This requires the user to provide a ***characterisation of the temporal properties of each component***, so that end-to-end execution times can be synthesised. Such end-to-end execution times can finally be provided, together with other parameters such as periods, priorities and deadlines, to the scheduling analysis tool.

As a matter of fact, these temporal characterisations might even be automatically generated by an ***on-target performance measurement*** setting of the component framework, provided that a more abstract behavioural *profile* – omitting numbers – is available. For complex components, execution time measurement scenarios could be generated by ***static analysis of component code*** as developed at CEA-List.

Finally, on-target measurements may also be performed for OS, middleware and network ***traversal timings***, provided that quantitative data flow descriptions are available.

Other verifications might be performed besides schedulability analysis. Most notably, Ocarina [19] has been developed in order to enable the configuration of the middleware from an AADL [16] description. In parallel with the generation of the middleware code, a behavioural model of this middleware is generated into a well formed coloured Petri net in order to verify behavioural properties of the middleware thanks to model checking techniques.

As another verification example, the AADL Error annex is used [18] to model possible faults, their probability of occurrence, and their propagation through the system. Automatically mapped into a stochastic Petri net, this allows analysing fault occurrence and propagation in terms of probability.

THALES is currently involved in ITEA-SPICES[4] project, whose objective is to integrate such verification tools with component-based frameworks. SPICES strategy is to make all descriptors and tools input to be AADL models, so that users may apply various verification and simulation tools to their own AADL application models. The target outcome is a featured component framework connected the relevant verification tools for addressing ***safety-critical avionics applications***.

3.2. Flexibility

Antagonistic with enforcing safety critical requirements, enhanced flexibility of the component framework will ease application developers and – as we will see – users of multi-mission systems to handle variability of the environment, as well as variability of the system itself.

[4] SPICES is an ongoing ITEA funded collaborative project.

Dynamic reconfiguration

Many research activities have been performed to provide dynamic reconfiguration mechanisms, and use case examples are numerous: reduction of functionalities due to a battery or bandwidth limitation, recuperation of functionalities in case of fault recovery, fine tuning and debug of the application, load balancing, user requests, are typical examples that require reconfiguration mechanisms.

A proper reconfiguration support in a component framework is made of both runtime mechanisms and description language. Regarding runtime mechanisms, component frameworks following Lw-CCM specification implement component instantiation and removal, component connection and disconnection. Openness of MyCCM container implies instantiation, removal and (dis-)connection capabilities of container services as well.

Besides runtime support, ***expression of dynamic reconfiguration policies is the real issue***. A distinction might be done between two reconfiguration categories, depending on whether the different configurations are ***statically enumerated*** and fully specified, or if on the contrary configurations are unknown or ***underspecified before actual occurrence*** of reconfiguration. The former case is referred to as pseudo-static reconfiguration, while the latter is simply called dynamic configuration. Each category might be better addressed by a specific description language for reconfiguration policies. Guarded state automaton would be suitable for pseudo-static reconfiguration, while more expressive language would be necessary for dynamic reconfiguration. Note that *programmatic* expression of reconfiguration policies is already available in MyCCM. Reconfiguration logics can be implemented in dedicated components behaving like deployment agents. A key benefit in stepping beyond this programmatic support is to verify and control reconfiguration policies, since reconfiguration is often subject to timing requirements as well.

In [22], for example, the reconfiguration may be delayed depending on the time of service interruption that the reconfiguration process requires. Reconfiguration priority is another key parameter. For instance, the reconfiguration of a cell phone due to a low-battery signal should not impede the end user to receive or emit calls. At the opposite, if reconfiguration is due to the breakdown of a computing resource, the reconfiguration may be more important than the execution of some functional parts of the application. In the scope of Flex-eWare[5] project, reconfiguration capabilities will be added to MyCCM framework.

Multi-mission support

The objective of a framework-based support to multi-mission systems is to ease the configuration of a multi-mission platform system by an end user. Typical example is an Unmanned Aerial Vehicle (UAV) usually used to monitor forest fires that could be reconfigured in emergency for supervision of evacuation operations after an earthquake. In this case, the infrared camera must be replaced by a standard camera, and the system must be reconfigured.

As part of Flex-eWare project, we will design a tool dedicated to ease configuration of multi-mission systems, based on ontology of the application domain. This tool helps the end user to select components providing and using consistent services. The correspondence of services is evaluated within the ontology, relying on ***semantic annotations*** attached to the services. This tool also supports the assembly of those components by ***generating the connectors*** for two components that are semantically equivalent but syntactically different.

The following example constitutes a use case of such a tool: supposing that the two formerly mentioned cameras drivers have different interfaces (different operation names, parameters type, etc…). Note that if the cameras are done by different vendors, the probability of this scenario is high. Still, the platform integrator might have attached to these two camera drivers a common goal in the ontology, and equivalent concepts attached to interfaces as well. Then a mechanism of data representation may be generated to enforce the compatibility of those interfaces.

Very few research works have been undertaken in order to tackle the flexibility issue in the field of DRES. These techniques, based on semantic representation of application domains have mainly been studied for information systems and web services. Indeed, a major issue when using such a technique is to guarantee that the behavioural properties that have been verified on the previous system will still be verified in its new configuration. In the case of critical systems, in which properties are very difficult to check, this issue constitutes an open issue.

[5] Flex-eWare is a ANR (*Agence Nationale de la Recherche*) project focusing on flexibility of component-based systems

4. Conclusion

As shown by the description of FREMM IRST use case, component-based frameworks are already applicable to industry-standard distributed real time embedded systems. Moreover, MyCCM internal design makes it ***easy to adapt*** to the non functional requirements of a particular application domain.

State-of-the-art real time operating systems and communication middleware provide a level of performance that enable component frameworks to address ***highly constrained systems***, such as vetronics and robotics.

Although with positive impact, component frameworks will not solve all the issues of developing hard real time. No matter whether a component framework is used or not: what is hard is to come out with an application design globally satisfying the antagonistic constraints of predictability and resource usage optimisation. However, ***component frameworks have good consequences on productivity*** and potentially integrate verification tools with simulation capabilities in a consistent manner. In other words, component frameworks might become the working environment of real time embedded software architects.

5. References

[1] : C. Szyperski. Component Software: Beyond Object-Oriented Programming. *Adison-Wesley, 1998.*

[2] : T. Valesky, Enterprise JavaBeans: Developing Component-Based Distributed Applications. *Addison-Wesley 1999.*

[3] : Object Management Group. CORBA Component Specification. In: *OMG Document formal/02-06-65, Juin 2002.*

[4] : R. Marvie, P. Merle, and J.M. Geib. Separation of concerns in modeling distributed component-based architectures. In: *proc. of Enterprise Distributed Object Computing Conference, 2002 (EDOC '02).*

[5] : A.D. Birrell, and B.J. Nelson. Implementing Remote Procedure Calls. In: *ACM Transactions on Computer Systems 2, 1 (February 1984): 39-59.*

[6] : Object Management Group. Light Weight CORBA Component Model Revised Submission. In: *OMG Document realtime/03-05-05 edn. (2003).*

[7] : S. Robert, A. Radermacher, V. Seignole and S. Gérard. The CORBA connector model. In: *Proceedings of adaptive and reflexive middleware, 2005 (ARM 05).*

[8] : S. Robert, A. Radermacher, V. Seignole, S. Gérard, V. Watine, and F. Terrier. The CORBA Connector Model. In: *proceedings of Software Engineering and Middleware, ACM digital library, September 2005, Lisbon, Portugal.*

[9] : Object Management Group. Deployment and Configuration Adopted Submission. In: *OMG Document ptc/03-07-08 edn. (2003)*

[10] : A. Bailly. Test et validation de composants logiciels. In: *PhD Thesis in computer science, at Université des Sciences et Techniques de Lille (USTL/LIFL).France, 2005.*

[11] : G. Madl, S. Abdelwahed, and D.C. Schmidt. Verifying distributed real-time properties of embedded systems via graph transformations and model checking (invited paper, submitted).. In: *The International Journal of Time-Critical Computing, 2005.*

[12] : J. Carlson, J. Hakansson, and P. Pettersson.. SaveCCM: An Analysable Component Model for Real-Time Systems. In: *Proceedings of Formal Aspects of Component Software, 2005.*

[13] : J.P. Etienne, and S. Bouzefrane. Vers une approche par composants pour la modélisation d'applications temps réel. In: *Proceedings of sixth francophone conference on Modeling and verification, 2006.*

[14] : K.G. Larsen, P. Pettersson, and W. Yi. UPPAAL in a Nutshell. In: *Journal on Software Tools for Technology transfer, 1(1-2):134-152, October 1997.*

[15] : M. González Harbour, J.J. Gutiérrez, J.C. Palencia and J.M. Drake. MAST: Modeling and Analysis Suite for Real-Time Applications. In: *Proceedings of the Euromicro Conference on Real-Time Systems, Delft, The Netherlands, June 2001.*

[16] : SAE – Society of Automative Engineers. SAES AS5506. In: *Embedded Computing Systems Committee, SAE, November, 2004*

[17] : F. Singhoff, J. Legrand, L. Nana, and L. Marcé. Scheduling and Memory requirements analysis with AADL. In: *Proceedings of the 2005 annual ACM SIGAda international conference on Ada, Atlanta, GA, USA, 2005.*

[18] : A.E. Rugina, K. Kanoun and M. Kaâniche. AADL-based Dependability Modelling. In : *LAAS report N°06209, 2006.*

[19] : T. Vergnaud. Modélisation des systèmes temps-réel embarqués pour la génération automatique d'applications formellement vérifiées. In: *PhD thesis, at Ecole Nationale Supérieur des Télécommunications de Paris, France, 2006*

[20] : K. Fujii and T. Suda. Component Service Model with Semantics (CoSMoS) : A new Component Model for Dynamic Service Composition. In : *Proceedings of Applications and the Internet Workshops (SAINTW'04), Tokyo, Japan, 2004, p. 348-355.*.

[21] : L. Apvrille. Contribution à la reconfiguration dynamique de logiciels embarqués temps-réel : Application à un environnement de télécommunication par satellite. In : *PhD thesis, Laboratoire d'Analyse et d'Architecture des Systèmes du CNRS (LAAS),Toulouse, France, 2002.*

[22] : E. Schneider. A Middleware Approach for Dynamic Real-Time Software Reconfiguration on Distributed Embedded Systems. In: *PhD thesis, at Université Louis Pasteur, Strasbourg, France, 2004.*