An Ada 2005 Technology for Distributed and Real-Time Component-Based Applications

Patricia López Martínez, José M. Drake, Pablo Pacheco, and Julio L. Medina

Departamento de Electrónica y Computadores, Universidad de Cantabria,
39005-Santander, SPAIN
{lopezpa,drakej,pachecop,medinajl}@unican.es

Abstract: The concept of interface in Ada 2005 significantly facilitates its usage as the basis for a software components technology. This technology, taking benefit of the resources that Ada offers for real-time systems development, would be suitable for component-based real-time applications that run on embedded platforms with limited resources. This paper proposes a model based technology for the implementation of distributed real-time component-based applications with Ada 2005. The proposed technology uses the specification of components and the framework defined in the LwCCM standard, modifying it with some key features that make the temporal behaviour of the applications executed on it, predictable, and analysable with schedulability analysis tools. Among these features, the dependency on CORBA is replaced by specialized communication components called connectors, the threads required by the components are created and managed by the environment, and interception mechanisms are placed to control their scheduling parameters in a per-transaction basis. This effort aims to lead to a new IDL to Ada mapping, a prospective standard of the OMG.

Keywords: Ada 2005, Component-based technology, embedded systems, real-time, OMG standards

1 Introduction[1]

While in the general-purpose software applications domain the component-based software engineering (CBSE) approach is progressing as a promising technology to improve productivity and to deal with the increasing complexity of applications, in the embedded and real-time systems domain, instead, its usage has evolved significantly slower. The main reason for this delay is that the most known CBSE technologies like EJB, .NET, or CCM, are inherently heavy and complex, they introduce not easily predictable overheads and do not scale well enough to fit the significant restrictions on the availability of resources usually suffered by embedded systems.

Trying to find an appropriate solution to this problem, european research projects like COMPARE [1] and FRESCOR [2], tackle from different points of view, the development

[1] This work has been funded by the European Union's FP6 under contracts FP6/2005/IST/5-034026 (FRESCOR project) and IST-004527 (ARTIST2 One) and by the Spanish Government under grant TIC2005-08665-C03 (THREAD project) and the ITEA SPICES project. This work reflects only the author's views; the EU is not liable for any use that may be made of the information contained herein.

F. Kordon and T. Vardanega (Eds.): Ada-Europe 2008, LNCS 5026, pp. 254–267, 2008.

of a real-time component-based technology compatible with the embedded systems constraints. Their approach is based on the usage of the Container/Component model pattern defined in the LwCCM specification developed by OMG [3], but avoiding the usage of CORBA as communication middleware, which is too heavy for this kind of applications. With this pattern, the interaction of the component with the run-time environment is completely carried out through the container, whose code is generated by automatic tools with the purpose of isolating the component developer from the details concerning the code of the execution environment.

The recent modification of the Ada language specification [4], so called Ada 2005, provides an enhanced option for the implementation of fully Ada native component-based technologies, which is really suitable for embedded platforms. Ada's native support for concurrency, scheduling policies, synchronization mechanisms, and remote invocations has always been a strength for implementing real-time and distributed systems. New to Ada 2005 is the concept of interface, which provides support for multiple inheritance. This is a key aspect in a component-based technology because it allows the components to inherit characteristics from both the technology with which they are developed as well as the application domain to which they belong. Besides, interfaces are used to encapsulate the services offered by the components (Facets in LwCCM) and also as the mechanism to make reference to the required services (Receptacles in LwCCM).

This paper proposes a component-based technology based on Ada. It implements the LwCCM framework, with the container/component model, and both the code of the environment and the code of the components are written in Ada 2005. The technology incorporates mechanisms to the running environment, and extends the specification of the components, in such a way that the timing behaviour of the final application is totally controlled by the automatically generated execution environment. In this way, real-time models of the application can be elaborated and analysed in order to verify its schedulability when the application is run in closed platforms, or to define the resource usage contracts required to operate in open environments like FRESCOR[2][5]. The description and deployment of applications and components in the technology follow the "Deployment and Configuration of Component-Based Distributed Applications" standard of the OMG [6] (D&C). The paper is focused in the description of the framework that is the base of the technology, particularly on the resources used to guarantee the required predictability.

Various proposals dealing with the adaptation of CBSE to real-time systems have appeared in the last years, though none of them have fully satisfied the industry requirements [7]. In the absence of a standard, some companies have developed their own solutions, adapted to their corresponding domains. Examples of that kind of technologies are Koala [8], developed by Philips, or Rubus [9], developed by Arcticus Systems and used by Volvo. These technologies have been successfully applied in the companies that created them, though none of them have stimulated an inter-enterprise software components market. However, they have served as the basis of other academic approaches. The Robocop component model [10] is based on Koala and adds some features to support analysis of real-time properties; Bondarev et al. [11] have developed an integrated environment for the design and performance analysis of Robocop models. Similarly, Rubus has been used as the starting point of the SaveCCT technology [12]; the component concept in SAVE is applied at a very low granularity. Under appropriate

assumptions for concurrency, simple RMA analysis can be applied and the resulting timing properties introduced as quality attributes of the assemblies. SaveCCT focuses on control systems for the automotive domain. In a similar way, COMDES-II [13] encapsulates control tasks following a hierarchical composition scheme, applied in an ad-hoc C based RT-kernel. The technology presented in this paper follows the idea proposed by PECT (Prediction-Enabled Component Technology) [14]. Sets of constraints in the components allow one to predict the behaviour of an assembly of components. In our case, this approach is applied to obtain the complete real-time model of the application. Though the Ada language is significantly used in the design and implementation of embedded real-time systems, we have not found references of its usage in support of component-based environments. This is probably due to the lack of support for multiple inheritance in the previous versions of the language.

The rest of this paper is organized as follows. Section 2 describes the two main processes involved in a components technology, emphasizing the main contributions of the proposal. Section 3 describes in detail the reference model of the framework, and the aspects included for developing analysable applications. Section 4 details the architecture and classes to which a component is mapped in the technology. Finally, Section 5 and 6 shows some practical experiences, conclusions and future work.

2 Real-Time Component-Based Development

A component technology defines two different development processes, shown in Figure 1. The components development process comprises the specification, implementation, and packaging of components as reusable and independently distributable entities. The development of component-based applications includes specification, configuration, deployment and launching of applications built as assemblies of available components. Both processes are independent and they are carried out by different agents in different stages, however, they require to be coordinated because the final products of the first process are

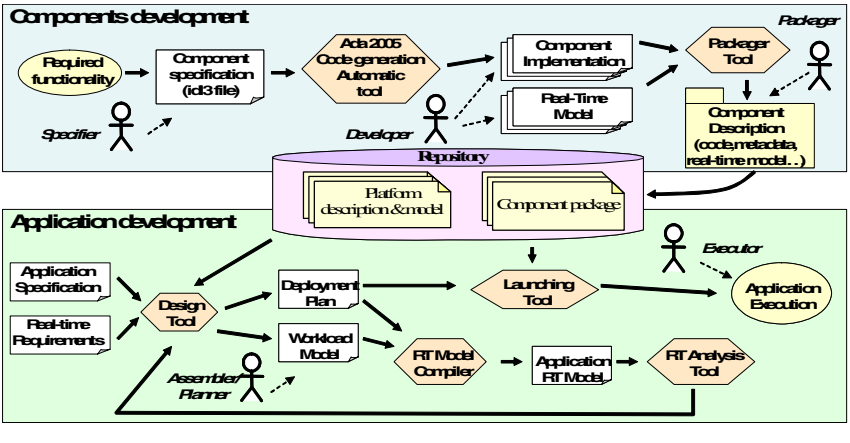


Fig. 1. Main processes in a component technology

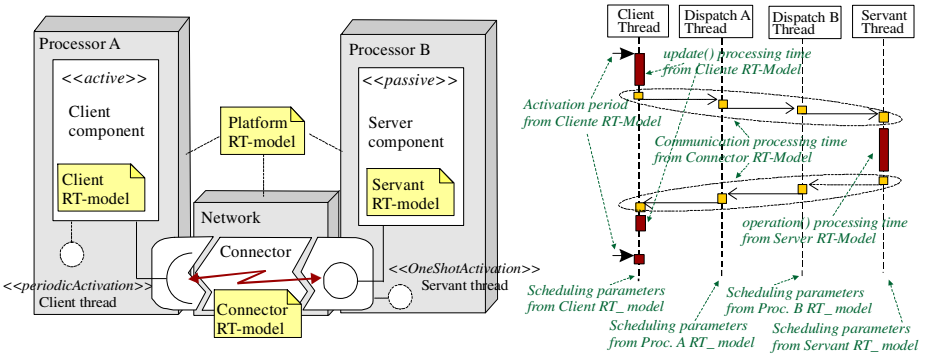
the inputs for the second. So, in order to guarantee their coherence, a component technology must define a set of rules about the kind of products and information that are generated in each phase of the process, and the formats in which they are supplied. A key aspect in a component technology is the opacity of the components; during the process of application development, components must be used without any knowledge of the internal details of their implementation or code. To achieve this opacity, models and information concerning functional and non-functional aspects of the component must be added to its implementation in the package that describes the component.

A component development process starts when the "specifier", who is an expert in a particular application domain, creates the specification of a component with concrete functionality in the domain. The "developer" implements this specification and creates models that describe the installation requirements of the component. This work is supported by automatic tools, which generate the skeletons for the code of the component based on the selected technology. Therefore, the developer task is reduced to design and implement the specific business code of the component without having to be aware of internal details about the technology. Finally, the "packager" gathers all the information required to make use of the component, and creates and publishes the distributable element that constitutes the component. Relevant aspects of the proposed technology related to components development are:

- The methodology for functional specification of components and the framework proposed by the LwCCM specification have been adopted as the basis for the technology. Hence, a container/component model is used in the component implementations, but CORBA is replaced by simpler static communication mechanisms with predictable behaviour, and suitable for the execution platform. Remote communication between components is achieved by using connectors. They are special components whose code is completely generated by the tools and which encapsulate all the support for interactions among components.
- Since component implementations are generated in Ada2005, it has been necessary to define the set of Ada packages to which the components and the elements of the LwCCM framework are mapped. An automatic code generation tool has been developed. This tool takes the specification of a component as input and generates all the code elements that provide support for the component inside the framework.
- The technology follows the D&C specification for the description of the package that holds the distributable component.

In order to apply the technology to hard real-time component-based applications, both standard specifications, D&C and LwCCM, have been extended with new elements that are used to describe the temporal behaviour of components and the requirements they impose on the resources in order to meet timing requirements:

- D&C specification has been extended in order to associate a temporal behaviour model to the specifications and implementations of components. This real-time model is used to describe the temporal responses of the component and the configuration parameters that it requires. This paper does not detail the modelling approach used. For a complete explanation of the approach see [15]. The basic idea is that the real-time model of a component is a parameterized model, independent of the



(a) Component model (b) Reactive and RT-Model

Fig. 2. RT Modeling of component-based applications

application in which the component is used, which describes the component temporal behaviour through references to the models of the platform in which the component is executed and to the models of other components that it uses in order to implement its functionality. Once all these elements are known in the context of an application deployed in a concrete platform, as it is shown in Figure 2a, the real time model of the complete application can be generated by composition of the individual real-time models of the software and hardware components that form it. This model describes the set of real-time transactions [16] executed in the application, as the one in Figure 2b, and can be used to obtain the response time of services, analyse the schedulability or evaluate the scheduling parameters required to satisfy the timing requirements imposed to the application. In our case, the real-time models of the components are formulated according to the MAST model [16], so that the set of tools offered by the MAST environment can be used to analyse the system.

- The LwCCM functional specification of a component has been refined with the purpose of controlling threading characteristics of the components. These characteristics include the number and assignment of threads and scheduling parameters. A component can not create threads inside its business code. Instead of that, for each thread that a component requires, it declares a port in its specification. This port implements one of the predefined interfaces OneShotActivation or PeriodicActivation (see Section 3).
- Interception mechanisms are used to control the scheduling parameters with which each invocation received by a component is executed. The specification of a component declares the configuration parameters required to assign concrete values of these scheduling parameters to a component instance.

The application development process consists in assembling component instances, choosing them from those which have been previously developed, and stored in the repository of the design environment. This process is carried out by three different agents in three consecutive phases. The “assembler” builds the application choosing the required component instances and connecting them according to their instantiation requirements.

This work is led by the functional specification of the application, the real-time requirements of the application, and the description of the available components. The result of this first stage is a description of the application as a composite component, which is useful by itself. The "planner" (usually the same agent as the assembler) takes this description and designs a deployment model for the application. This model includes assigments of component instances to nodes and the communication mechanisms between them. The result of this stage is the deployment plan, which completely describes the application and the way in which it is planned to be executed. Finally, the "executor" deploys, installs, and executes the application, taking the deployment plan and the information about the execution platform as inputs. This labour is usually assisted by automatic tools. Relevant aspects of the proposed technology regarding application development are:

- As well as describing components, the D&C specification is the basis for the process of designing and deploying an application. D&C defines the structure of the deployment plan that leads this process. It describes the component instances that form the application, their connections, the configuration parameters assigned to each instance and the assignment of instances to nodes.
- A deployment tool processes the information provided by the deployment plan. It selects the code of the components suitable for the target platform and generates the code required to support the execution of the components in each node. Specifically, it automatically generates the connectors, which provide the communication mechanisms between remote component instances, as well as the code for the main procedures executed on each node.

The specific aspects included in the application development process to support hard real-time applications are:

- Once the planner has developed the deployment plan, the local or remote nature of each connection between component ports is defined. Then, an automatic tool generates the code of the connectors based on the selected communication service and its corresponding configuration parameters, which were assigned to the connection in the deployment plan. The communication service used must hold a predictable behaviour, hence, the tool generates also the real-time models that describe the temporal behaviour of those connectors.
- Once the connectors have been developed together with their real-time models, and based on the deployment plan, a tool elaborates the real-time model of the application by composition of the real-time models of the components that form it (connectors included) and the models of the platform resources. This model is used either to analyse the schedulability of the application under a certain workload, or to calculate the resource usage contracts necessary to guarantee its operation in an open contractual environment [5]. In the latter case, these contracts will be negotiated, prior to the application execution, by the launching tool.
- The execution environment includes a special internal service as well as interception mechanisms that are used to manage in an automated way the scheduling parameters of the threads involved in the application execution. The configuration parameters of this service, whose values may be obtained by schedulability analysis, are specified in the deployment plan and assigned to the service at launching time.

3 Reference Model of the Technology

The proposed technology is based on the reusability (with no modification) of the business code of the components, and the complete generation by automatic tools of the code that adapts the component to the execution environment. This code is generated according to the reference model shown in Figure 3. It takes the LwCCM framework as a starting point, and adds to it the features required to control the real-time behaviour of the application execution. Each of the elements that take part in the execution environment are explained below.

Component: A component is a reusable software module that offers a well-defined business functionality. This functionality is specified through the set of services that the component offers to other components, grouped in ports called *facets*, and the set of services it requires from other components, grouped in ports called *receptacles*.

With the purpose of having complete control of the threading and scheduling characteristics of an application, and in the look for being able to analyse it, components in our technology are passive. The operations they offer through their facets are made up of passive code that can call protected objects. But this does not mean that there can not be active components in the framework, concurrency is provided by means of *activation ports*. When a component requires a thread for implementing its functionality, it declares a port that implements one of the two special interfaces defined in the framework: OneShotActivation or PeriodicActivation. These ports are recognized by the environment, which creates and activates the corresponding threads for their execution once the component is instantiated, connected and configured. The interface OneShotActivation declares a `run()` procedure, which will be executed once by the created thread, while the interface PeriodicActivation declares an `update()` procedure, which will be invoked periodically. A component can declare several activation ports, each of them representing an independent unit of concurrency managed by the component, and which are independent of the business invocations.

Activation ports are declared in the component specification (in the IDL file), and all the elements required for their execution are created by the code generation tool. Their configuration parameters, which include the scheduling parameters of the threads as well as the activation period (in case of PeriodicActivation ports) are assigned for each component instance in the deployment plan.

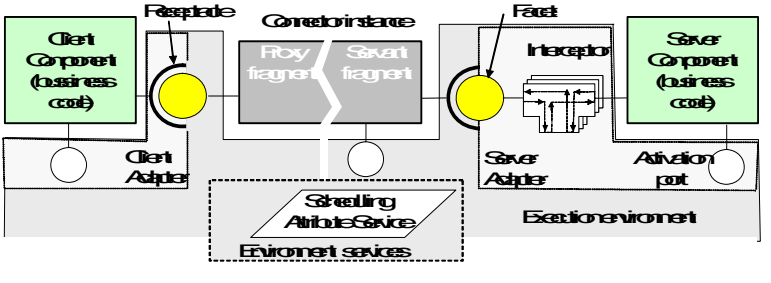


Fig. 3. Reference model of the technology

Adapter: An adapter is the part of the component's code which provides the run-time support for the business code. All the platform related aspects are included in the adapter. Its code is automatically generated according to the component/container model. With this programming approach the component developer does not need to know any detail about the underlying technology, he is only in charge of business code development.

Connector: A connector is the mechanism through which a component communicates with another component connected to it through a port. In our technology, a connector has the same structure as a component, but its business code is also generated by the deployment tool, based on:

- The interface of the connected ports. The connectors are generated from a set of templates which are adapted so that they implement the operations of the required interface.
- The location of the components (local vs remote), and the type of invocation (synchronous or asynchronous). Combinations among these different characteristics lead to different types of connectors. For local and synchronous invocations, the connector is not necessary, the client component invokes the operation directly on the server. For local and asynchronous invocations the connector requires an additional thread to execute the operation (obtained through activation ports). If the invocation is distributed, the connector is divided in two fragments: the *proxy fragment*, which is instantiated in the client node, and the *servant fragment*, which is instantiated in the server node. The communication between the two fragments is achieved by means of the communication service selected for the connection. In this case, the connector can also implement synchronous or asynchronous invocations, including the required mechanisms in the proxy fragment.
- The communication service or middleware used for the connection and its corresponding configuration parameters, which are assigned for each connection between ports in the deployment plan.

Interceptors: The concept of interception is taken from QoSforCCM [17]. It brings a way to support the management of non-functional features of the application. An interceptor allows to incorporate calls to the environment services inside the sequence of an invocation by executing certain actions before and after the operation is executed on the component. The support for interceptors is introduced in the adapter, so it is hidden to the component developer. Their introduction is optional for each operation, and it is specified in the deployment plan.

In our technology, interceptors are used to control the scheduling parameters with which each received invocation is executed. Based on the configuration parameters assigned to it in the deployment plan, each interceptor knows the scheduling parameter which corresponds to the current invocation, and uses the SchedulingParameterService to modify it in the invoking thread. With this strategy, different schemes for scheduling parameters assignment can be implemented. Besides common assignment policies, like Client Propagated or Server Declared [18], our technology allows to apply an assignment based on the transactional model of the application. With this policy, a service can be executed with different scheduling parameters inside the same end-to-end flow depending on the particular step inside the flow in which the invocation takes place. This

scheme enables better schedulability results [19]. The values of these parameters are obtained from the analysis using holistic priority assignment tools like the ones included in MAST, which is used as analysis environment in our technology.

SchedulingParameterService: It is an internal environment service which is invoked by the interceptors to change the scheduling parameters of the invoking thread. The kind of scheduling parameters that will be effectively used depends strongly on the execution platform, it may be a single priority, deadline, or the contract to use in the case of a FRESCOR flexible scheduling platform.

4 Architecture of a Component Implementation

There are two complementary aspects that a component implementation must address:

- The component has to implement the functionality that it offers through its facets, making use of its own business logic and the services of other components.
- The implementation must include the necessary resources to instantiate, connect and execute the component in the corresponding platform. This aspect is addressed by implementing the appropriate interfaces which allow to manage the component in an standard way. In our case, those defined by LwCCM.

Each aspect requires knowledge about different domains. For the first aspect, an expert on the application domain corresponding to the component functionality is required. For the second, however, what it is required is an expert on the corresponding component technology. The proposed architecture for a component implementation tries to find an structural pattern to achieve independency of the Ada packages that implement each aspect. Besides, the packages that implement the technology related aspects are to be automatically generated according to the component specification. With this approach, the component developer only has to design and implement the business code of the component.

The proposed architecture is based on the reference one proposed by LwCCM, but adapted for:

- Making use of the abstraction, security and predictability characteristics of Ada.
- Including the capacity for controlling threading characteristics of the components.
- Facilitating the automatic generation of code taking the IDL3 specification of the component as input and generating the set of classes that represent a component in the technology.
- Providing a well-defined frame in which the component developer designs and writes the business code.

In the proposed technology, the architecture of a component is significantly simplified as a consequence of the usage of connectors. When two connected components are installed in different nodes, the client component interacts only with the proxy fragment of the connector, while the server component interacts only with the servant fragment of the connector. Therefore, all the interactions between components are local, since it is the connector who hides the communications mechanisms used for the interaction.

For each component, four Ada packages are generated. Three of them are completely generated by the tool, while the last package leaves the “blank” spaces in which the

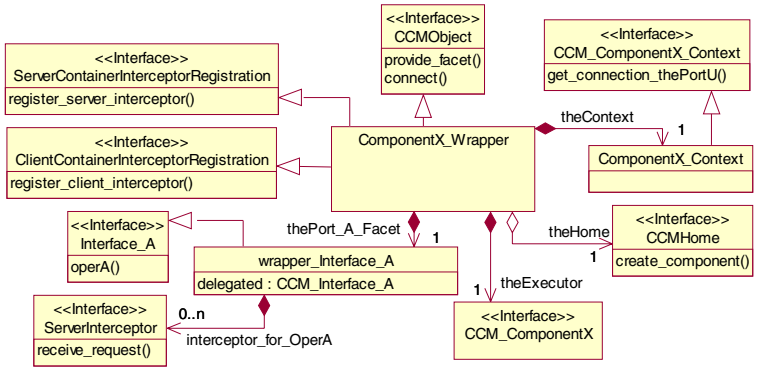


Fig. 4. Example of Component Wrapper Structure for ComponentX

component developer must include the business code of the component. The first module represents the adapter (or container) of the component. It includes the set of resources that adapt the business code of the component to the platform, following the interaction rules imposed by the technology. It defines three classes:

- The wrapper class of the component, called {ComponentName}_Wrapper, which represents the most external class of the component. It offers the equivalent interface of the component, which LwCCM establishes as the only interface that can be used by clients or by the deployment tool to access to the component. With this purpose, the class implements the CCMObject interface, which, among others, offers operations to access to the component facets, or to connect the corresponding server components to the receptacles. Besides, the capacity to incorporate interceptors is achieved by implementing the Client/ServerContainerInterceptorRegistration interfaces, a modified version of the interfaces with the same name defined in QoSCCM [17]. As it is shown in Figure 4, this class is a container which aggregates or references all the elements that form the component:
  - The component context, through which components access to their receptacles.
  - The home, which represents the factory used to create the component instance.
  - The executor of the component, which represents its real business code implementation. Its structure is explained below.
  - An instance of a facet wrapper class that is aggregated for each facet of the component. They capture the invocations received in the component and transfer them to the corresponding facet implementations, which are defined in the executor. The facet wrappers are the place in which the interceptors for managing non-functional features are included.
- The class that represents the context implementation, called {ComponentName}_Context. It includes all the information and resources required by the component to access to the components which are connected to its receptacles.
- The {ComponentName}_Home_Wrapper, which implements the equivalent interface of the home of the component. It includes the class procedures (static) that are used as factories for component instantiation.

The rest of generated Ada packages contain the classes that represent the implementation of the business code of the component (the executor). The LwCCM standard fixes a set of rules that define the programming model to follow in order to develop a component implementation. Taking the IDL3 specification of a component, LwCCM defines a set of abstract classes and interfaces which have to be implemented, either automatically or by the user, to develop the functionality of the component. This set of root classes and interfaces are grouped in the generated package {ComponentName}_Exec. The {ComponentName}_Exec_Impl package includes the concrete classes for the component implementation which inherit from the classes defined in the previous package. The class that represents the component implementation, {ComponentName}_Exec_Impl, which is shown in Figure 5, has the following attributes:

- A reference to the component context. It is set by the environment through the `set_session_context()` operation, and it is used to access to the receptacles.
- An aggregated object of the {ComponentName}_Impl class, whose skeleton is generated by the tool and has to be completed by the developer.
- Each activation port defined in the specification of the component, represents a thread that is required by the component to implement its functionality. For implementing those threads two kinds of Ada task types have been defined. The OneShotActivationTask executes once the corresponding run() procedure of the port, while the PeriodicActivationTask executes periodically the update() procedure of the corresponding port. Both types of task receive as a discriminant during its instantiation, a reference to the data structure that qualify their execution, including scheduling parameters, period, state of the component and the procedure to execute. For each activation port defined in the component, a thread of the corresponding type is declared. They will be activated and terminated by the environment by means of standard procedures that LwCCM includes in the CCMObject interface to control the lifecycle of the component.

The {ComponentName}_Impl class, represented in Figure 5, is defined in a new package, in order to hide the environment internals to the code developer. It represents the reference frame in which the developer introduces the business code. Relevant elements of this class are:

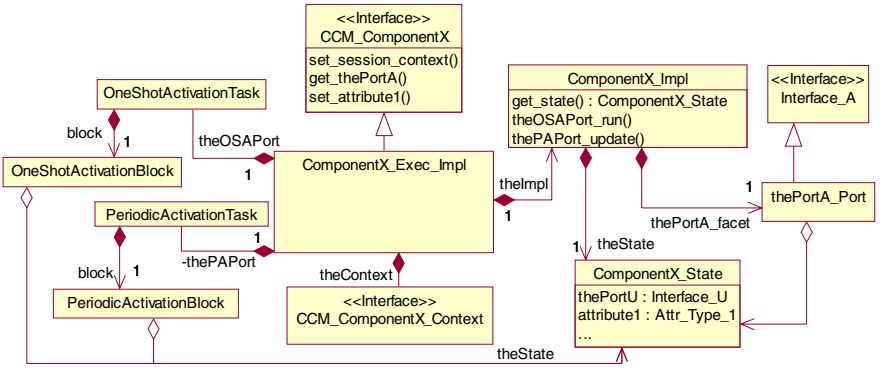


Fig. 5. Example of Component Implementation Structure for ComponentX

- For each facet offered by the component, a facet implementation object is aggregated. However, in the case of simple components, the class itself can implement the interfaces supported by the facets.
- All the implementation elements (facet implementations, activation tasks, etc.) operate according to the state of the component, which is unique for each instance. Based on that, the state has been implemented as an independent aggregated class, which can be accessed by the rest of the elements, avoiding cyclic dependencies.
- For each activation port defined in the component specification, the corresponding {PortName}_run() or {PortName}_update procedures are declared.

Most of the code of this class is generated automatically, the component developer only has to write the body of the activation port procedures (run or update), and the body of the operations offered by each of the facets implementations. The developer, who knows the temporal behaviour of the code, must also elaborate the real-time model of the component. In the case of a connector, the structure generated is exactly the same, but the "business" code, which in that case consists in the code required to implement remote invocations, is also automatically generated by the deployment tool.

The current available Ada mapping for IDL [20] is based in Ada95, so for the development of the code generation tool, it has been necessary to define new mappings for some IDL types in order to get benefit of the new concepts introduced in Ada 2005. The main change concerns to the usage of interfaces. The old mapping for the IDL "interface" type led to a complex Ada structure while now can be directly mapped to an Ada interface. Besides, some data structures defined in IDL, as for example the "sequence" type, can be implemented now with the new Ada 2005 containers.

5 Practical Experience

At the time of the first attempts made to validate the proposed technology, there was no real-time operating system with support for Ada 2005 applications, so the tests were run on a Linux platform, using the GNAT (GAP) 2007 compiler. The construction of the connectors for the communication between remote components, was made using the native Ada Distributed System Annex (DSA), Annex E of the Ada specification. The implementation of DSA used was GLADE [21]. Distributed test applications were developed and executed successfully. The platforms used in this evaluation were sufficient for the conceptual validation of the technology, since from the point of view of the software architecture the final code is equivalent, but of course, it is not appropriate for the validation of the timing properties of real-time applications.

The recently released new version of MaRTE_OS [22] provides now support for the execution of Ada 2005 applications, and allows to test the technology over a hard real-time environment. Still there is a lack for a real-time communication middleware. An enhanced version of GLADE that enables messages priority assignment exists for MaRTE_OS & GNAT [23], but it has not been ported to the new versions. To overcome this limitation, we have developed simpler connectors using a link layer real-time protocol. Our first tests on a real-time platform have been done with connectors that use directly the RT-EP [24] protocol for the communication between remote components.

The same application tested in the linux platform was used in MaRTE_OS, and as expectable, the code of the components did not require any modification, the only necessary change was the development of the new connectors suitable for the new communication service (RT-EP) used.

6 Conclusions and Future Work

This paper proposes a model based technology for the development of real-time component-based applications. The usage of the Ada language for its implementation, makes it particularly suitable for applications that run in embedded nodes with limited resources and strict timing requirements. The technology is based on the D&C and LwCCM standard specifications, which have been extended in order to support the development of applications with a predictable and analysable behaviour.

The key features of this technology have been specified and tested successfully. Nevertheless some challenges arise for this community to face. The most rewarding of them is the availability of an Ada native communication middleware, here used in the development of connectors, which must hold predictable behaviour, and allow a priority assigment for the messages based on the transactional (or so called end-to-end flow) model. Our aim is to develop the connectors using the Ada Distributed System Annex so that applications rely only on the Ada run-time infrastructure with no additional middleware, which is highly desirable to target small embedded systems.

As future work, some more tests have to be applied in order to quantify the concrete overheads introduced by the technology. A planned enhancement for the technology is the construction of a graphical environment to integrate all the stages of development of an application: design, code generation, analysis, and finally, execution. Another effort that has been started in the OMG and arise from this work is the elaboration of an updated version of the mapping from IDL to Ada 2005 [25].

References

[1] IST project COMPARE: Component-based approach for real-time and embedded systems, http://www.ist-compare.org
[2] IST project FRESCOR: Framework for Real-time Embedded Systems based on Contracts, http://www.frescor.org
[3] OMG: Lightweight Corba Component Model, ptc/03-11-03 (November 2003)
[4] Tucker Taft, S., Duff, R.A., Brukardt, R.L., Plödereder, E., Leroy, P.: Ada 2005 Reference Manual. LNCS, vol. 4348, pp. 43–48. Springer, Heidelberg (2006)
[5] Aldea, M., et al.: FSF: A Real-Time Scheduling Architecture Framework. In: Proc. of 12th RTAS Conference, April 2006, San Jose, USA (2006)
[6] OMG: Deployment and Configuration of Component-Based Distributed Applications Specification, version 4.0, Formal/06-04-02 (April 2006)
[7] Möller, A., Åkerholm, M., Fredriksson, J., Nolin, M.: Evaluation of Component Technologies with Respect to Industrial Requirements. In: Proc. of 30th Euromicro Conference on Software Engineering and Advanced Applications (August 2004)
[8] Ommering, R., Linden, F., Kramer, J.: The koala component model for consumer electronics software. IEEE Computer, IEEE, 78–85 (2000)

[9] Lundbäck, K.-L., Lundbäck, J., Lindberg, M.: Component based development of dependable real-time applications Arcticus Systems, http://www.arcticus-systems.com
[10] Bondarev, E., de With, P., Chaudron, M.: Predicting Real-Time Properties of Component-Based Applications. In: Proc. of 10th RTCSA Conference, Goteborg (August 2004)
[11] Bondarev, E., et al.: CARAT: a toolkit for design and performance analysis of component-based embedded systems. In: Proc. of DATE 2007 Conference (April 2007)
[12] Åkerholm, M., et al.: The SAVE approach to component-based development of vehicular systems. Journal of Systems and Software 80(5) (May 2007)
[13] Ke, X., Sierszecki, K., Angelov, C.: COMDES-II: A Component-Based Framework for Generative Development of Distributed Real-Time Control Systems. In: Proc. of 13th RTCSA Conference (August 2007)
[14] Wallnau, K.C.: Volume III: A Technology for Predictable Assembly from Certifiable Components, Technical report, Software Engineering Institute, Carnegie Mellon University, April 2003, Pittsburgh, USA (2003)
[15] López, P., Drake, J.M., Medina, J.L.: Real-Time Modelling of Distributed Component-Based Applications. In: Proc. of 32h Euromicro Conference on Software Engineering and Advanced Applications, August 2006, Croatia (2006)
[16] González Harbour, M., Gutiérrez, J.J., Palencia, J.C., Drake, J.M.: MAST: Modeling and Analysis Suite for Real-Time Applications. In: Proc. of the Euromicro Conference on Real-Time Systems (June 2001)
[17] OMG: Quality of Service for CORBA Components, ptc/06-04-05 (April 2006)
[18] OMG: Real-Time CORBA Specification, v1.2 formal/05-01-04. Enero (2005)
[19] Gutiérrez García, J.J., González Harbour, M.: Prioritizing Remote Procedure Calls in Ada Distributed Systems. In: Proc. of the 9th Intl. Real-Time Ada Workshop, ACM Ada Letters, XIX, 2, pp. 67–72 (June 1999)
[20] OMG: Ada Language Mapping Specification - Version 1.2 (October 2001)
[21] Pautet, L., Tardieu, S.: GLADE: a Framework for Building Large Object-Oriented Real-Time Distributed Systems. In: Proc. of the 3rd IEEE Intl. Symposium on Object- Oriented Real-Time Distributed Computing, March 2000, Newport Beach, USA (2000)
[22] Aldea, M., González, M.: MaRTE OS: An Ada Kernel for Real-Time Embedded Applications. In: Strohmeier, A., Craeynest, D. (eds.) Ada-Europe 2001. LNCS, vol. 2043. Springer, Heidelberg (2001)
[23] López-Campos, J., Gutiérrez, J.-J., González-Harbour, M.: The Chance for Ada to Support Distribution and Real-Time in Embedded Systems. In: Llamosí, A., Strohmeier, A. (eds.) Ada-Europe 2004. LNCS, vol. 3063, pp. 91–105. Springer, Heidelberg (2004)
[24] Martínez, J.M., González, M.: RT-EP: A Fixed-Priority Real Time Communication Protocol over Standard Ethernet. In: Vardanega, T., Wellings, A.J. (eds.) Ada-Europe 2005. LNCS, vol. 3555, pp. 180–195. Springer, Heidelberg (2005)
[25] Medina, J.: Status report of the Ada2005 expected impact on the IDL to Ada Mapping. OMG documents mars/07-09-12 and mars/07-06-13 (2007), http://www.omg.org