Ada-CCM: Component-based Technology for Distributed Real-Time Systems

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

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

Abstract: This paper proposes a technology for the development of distributed real-time component-based applications, which takes advantage of the features that Ada offers for the development of applications with predictable temporal behaviour, and which can be executed in embedded platforms with limited resources. The technology uses the Deployment and Configuration of Component-based Distributed Applications Specification of the OMG for describing the components, the execution platforms and the applications. The framework defined in the Lightweight CCM standard of the OMG is taken as the basis of the internal architecture of the components and the applications. It has been extended with a number of features to make the temporal behaviour of the applications predictable. Among these extensions, the usage of CORBA has been replaced by special distributed components, called connectors, which implement the interaction between components by means of predictable and customizable communication services. Besides, special mechanisms have been introduced in the environment to make the threading characteristics of the components configurable. The technology fixes the responsibilities and the knowledge required by each actor involved in the component-based development process, and for each of them it defines the input and output artifacts that they have to manage.

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

1 Introduction[1]

The design of real-time software for embedded systems has a strategic interest in the industry nowadays. In many application areas, like robotics, industrial control, automotive, etc., systems are built by assembling subsystems (controllers, vision systems, carburation systems, etc.). A subsystem may be equipped with its own embedded processor, or deployed in a number of them, each of which is in charge of controlling its own hardware, and the communication among them is achieved by means of a dedicated network (Ethernet, CAN bus, firewire, etc.). This architecture provides considerable modularity and reconfigurability, and it minimizes and standardizes the wiring.

1. This work has been funded by the European Union's FP6 under contracts FP6/2005/IST/5-034026 (FRESCOR), FP7/224330 (ADAMS) and ArtistDesign, EU FP7 NoE 214373 and by the Spanish Government under grant TIC2005-08665-C03 (THREAD) and EVOLVE. 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.

The increasing capacity and memory provided by the processors, and the subsequent rise in the amount of functionality that they must support, together with the distributed nature of the execution platforms, and the real-time requirements of the final applications involved, make the software for this kind of systems very complex. Applying component-based design strategies to this domain offers several advantages:

- It provides a simple architecture based on interfaces instead of protocols between subsystems.
- The reconfiguration of a system can be achieved by modifying the deployment plan, without requiring any hand-made code modification. It also simplifies the evolution and versioning of systems since it is only required to replace or add new components with well-defined functionalities.

Conventional component technologies are not easily adaptable to embedded systems, since they require a large amount of services from the operating systems, file systems, middleware or networks, which are not compatible with the limitation of resources suffered by them. Various proposals dealing with the adaptation of CBSE to real-time systems have appeared in the last years. Some companies have developed their own solutions, adapted to their corresponding domains. Examples of that kind of technologies are Koala [1], developed by Philips, or Rubus [2], 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 [3] is based on Koala and adds some features to support analysis of real-time properties. Similarly, Rubus has been used as the starting point of the SaveCCT technology [4], which is focused on control systems for the automotive domain; and under appropriate assumptions for concurrency, simple RMA analysis can be applied and the resulting timing properties introduced as quality attributes of the assemblies. From the Ada language perspective, even though it 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 paper proposes a component-based technology, denominated Ada-CCM, which is specifically conceived for embedded, distributed and real-time systems. The key aspects of the technology are:

- It follows the components specification style and the programming model proposed in the Lightweight CCM (LwCCM) [5] specification of the OMG. Therefore, a container/component pattern is used, though with an essential difference: CORBA is not the communication mechanism used. The connection between a facet and a receptacle is always local, and the communication between remote components is achieved by means of special distributed components called connectors. Besides, special mechanisms have been included in the containers to make the temporal behaviour of the application execution predictable.
- Both the business component implementations and the containers code are written in Ada 2005 [6]. Making use of the Ada's native support for concurrency, scheduling policies and synchronization mechanisms, it is possible to generate code with predictable temporal behaviour. Ada is intended for embedded systems, and there are small foot-print run-time libraries for Ada that can be executed on

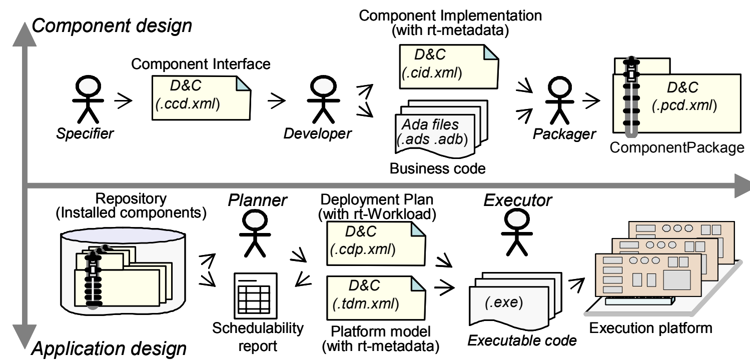

Fig. 1. Actors and main artifacts in components and applications design

bare embedded computers. The new version of the language is essential to this work, since it introduces support for multiple inheritance based on interfaces, which are key aspects in the development of component-based technologies.

- The external interface and the internal implementations of components, execution platforms and deployment plans are described following the Deployment and Configuration of Component-based Distributed Applications Specification of the OMG (D&C) [7]. It has been extended to include metadata about the temporal behaviour of components, platforms and applications. This information is used to analyse the schedulability of the application as part of the development process.

The responsibilities and the artifacts that serve as inputs and outputs for the different actors that take part in the development process of an application are precisely defined in the proposed technology; they are briefly sketched in Figure 1. A detailed explanation of the development process and the involved actors and artifacts is given along the paper. Due to the real-time nature of the developed applications, this process adds a number of aspects to the standard one. The developer must formulate, together with the business code, the description of the temporal behaviour of the component. Real-time models describing the capacity provided by the elements of the execution platforms are also required. After defining the structure of an application by means of the deployment plan, the planner can build its temporal behaviour model. Based on the real-time requirements established in the specification of the application, he defines the workloads, which are the basis for the schedulability analysis. The results of the analysis are the set of scheduling parameters with which the component instances are configured, as well as the specification of the platform resources that shall be reserved in order to schedule their execution timely.

The paper is organized as follows, Section 2 describes the process of generation of a deliverable component in the technology. In Section 3, the reference model of the technology is explained, together with the structure of Ada packages to which a component is mapped. Section 4 details the process of development of an application built as an assembly of components. Section 5 describes the development suite. Section 6 details the features of the execution platform and an application example. Section 7 presents the way in which real-time models are added to the description of components and platforms, and introduces the modelling and analysis suite used in the technology. Finally, Section 8 summarizes our conclusions and future work.

2 Component development process

Figure 2 shows the process that is followed to generate a deliverable component, which will be able to be automatically assembled and executed in future applications.

When the *specifier*, who is an expert in the application domain, finds out that a certain functionality is demanded, he creates the specification of a new component that satisfies it. The component specification is formulated according to the D&C specification, by means of a Component Interface Description (.ccd file). As it is explained in Section 7, the D&C specification has been extended to incorporate special real-time composability requirements, and the metadata related to the temporal behaviour of the component. Besides, with the purpose of controlling the number of threads managed by a component, and making their scheduling parameters configurable (i.e. the priority), an important aspect has been introduced in the technology. Each thread needed by the business implementation of a component to implement its functionality, is required by means of an special activation port declared on its specification (the way in which the container manages these activation ports is explained in Section 3).

As an example, using the LwCCM graphical notation, Figure 3 shows the specification of the *SoundGenerator* component. This component is part of an application we have developed to test the technology. It offers a facet, *playerPort*, which implements the *I_Player* interface and is used by the client components to generate different sounds. It is an active component, since it requires a thread to play the sound without forcing the client to be blocked until the sound is completed. The thread is demanded by means of the declaration of the *soundThread* activation port. The component declares a configurable property, *soundThreadPeriod,* which represents the period with which the thread provided by the container will invoke the update() procedure corresponding to the soundThread port (see Section 3).

The *developer* writes the business code of the component as a set of Ada packages (.ads and .adb files). The code has to implement the *Component Business Interface*. This interface is generated using the *ComponentTemplateGenerator* tool, which takes the specification of the component, and the IDL descriptions of the related functional interfaces as inputs. It defines the set of methods that the business code must implement in order to be managed by the container in an automatic way. It has no dependencies with the technology, so the component developer is free to design the business code without having to be aware of any internal detail of the technology. The

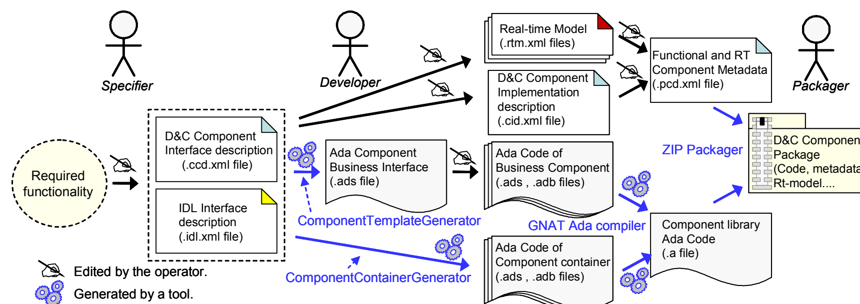

Fig. 2. Actors and artifacts involved in component development with AdaCCM

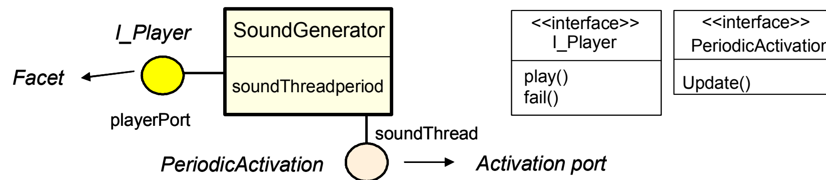

Fig. 3. SoundGenerator component declaration

developer has complete knowledge about the internal behaviour of the component, so he has to create, together with the code, the real-time model that describes the temporal behaviour of the component. Besides, the developer has to specify the requirements that the component imposes on the platform to be able to execute. All this information is described by means of a D&C's Component Implementation Description (.cid file).

The *packager* carries out the last phase of the process, which consists in building the package that constitutes the deliverable component. Taking the specification of the component as input, a new code generation tool, called *Component ContainerGenerator*, generates the set of Ada source files (.ads and .adb files) that implement the container of the component. The container groups all the resources that are used to adapt the business code implementation to the execution environment (its structure is explained in the next section). These files are compiled together with the business code using the standard GNAT Ada compiler and a library is generated (.a file). This library is the only artifact that is required to execute the component on the target platform. Finally, the packager gathers all the information available about the component, and creates and publishes the package that describe the component. This package includes both the binary code of the component and the metadata (both functional and non-functional) that allow a future user to decide about the suitability of the component in an application, and also describe the way in which the component can be instantiated and executed. This package constitutes the deliverable component and the corresponding metadata is defined according to the Package Configuration Description element of the D&C (.pcd file).

3 Component Architecture

A full component implementation must address two complementary aspects:

- It has to implement the business functionality that it offers through its facets, making use of its own business logic and the services of other components accessed through its receptacles. This aspect concerns the application domain in which the component functionality is required.
- It must include the mechanisms that are required to instantiate, connect and execute the component in the corresponding platform and framework. This aspect is addressed by implementing the appropriate interfaces that allow managing the component in a standard way. This aspect is related to the component technology used, in our case LwCCM.

The architecture of a component proposed in this technology follows an structural pattern that achieves independency of the Ada packages that implement each aspect.

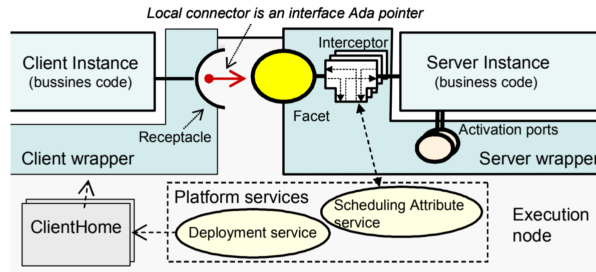

Fig. 4. Reference model of the technology

The packages that implement the technology related aspects are completely generated by automatic tools, taking the specification of the component as the only input. The component developer only has to design and implement the business code of the component, without having to have any knowledge about the underlying technology.

The architecture of a component is generated according to the reference model of the technology, which is shown in Figure 4. It is based in the container-component framework proposed in LwCCM, but it has been extended with some new features required to make the behaviour of the application execution predictable:

- In order to make the threading and scheduling characteristics of an application configurable, and therefore, to control its schedulability, the business code of a component has not internal threads. The internal activity of a component is defined through the set of activation ports declared in its specification. These ports are recognized by the container, which creates and activates the corresponding controlled threads to execute the activity of the component once it is instantiated, connected and configured. These activation ports can implement one of the predefined interfaces: PeriodicActivation or OneShotActivation. The OneShotActivation interface declares a `run()` procedure, which will be executed once by the created thread, while the PeriodicActivation interface declares an `update()` procedure, which will be invoked periodically. A component can declare several activation ports, each of them representing an independent entity of concurrency. Activation ports are declared in the component specification, and all the elements required for their execution are created automatically by the container generation tool. Their configuration parameters, which include the thread priorities as well as the activation periods (in case of PeriodicActivation ports), are assigned to each component instance in the deployment plan.
- The connection established by each receptacle in a component is always local and it is implemented by an Ada pointer to the corresponding interface. If the connection between components is local, the pointer access directly to the facet of the server component. If the connection is remote, it is implemented by an specialized component called connector. As it is shown in Figure 5, a connector is a distributed component, composed by two parts: the *proxy* side which is instantiated in the client node, and the *servant* side, which is instantiated in the server node. The proxy offers a local facet to the client component and it includes the synchronization mechanisms for the invoking thread. The servant side has a receptacle which connects with the server facet and it includes and manages the

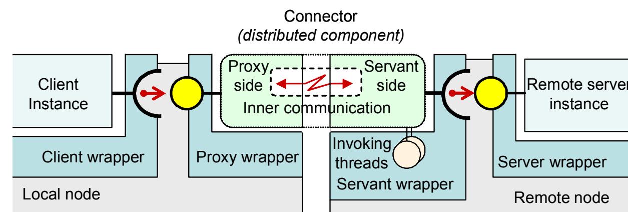

Fig. 5. Connector component

threads that carry out the remote invocations. The communication mechanisms between the two parts (marshalling and unmarshalling of the invocation and return parameters, and transmission and dispatching of messages) are internal to the connector and depend on the communication service chosen for its implementation. The code of the connector is completely generated by automatic tools according to the interface of the connected ports, the location of the components, and the communication service used for the connection. We have developed connectors which use directly the RT-EP protocol [8], which is a real-time protocol implemented over Ethernet. This kind of connectors are suitable for connections with real-time requirements. An alternate implementation with no prioritized messages has been made using GLADE [9], an implementation of the Ada Distributed Systems Annex (DSA).

- Interception mechanisms [10] and a special internal service are introduced in the container to control non functional features of the component service executions. In our technology they are specifically used to control the scheduling parameters with which each invocation received in a component operation is executed. Based on the configuration parameters assigned to each instance in the deployment plan, each interceptor knows the scheduling parameter which corresponds to the current invocation, and uses the *SchedulingAttributeService* 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 [11], 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 [12].

For each component specification, four Ada packages are generated. The first package represents the adapter of the component and includes all the resources to adapt the business code of the component to the platform, following the interaction rules imposed by the technology. The wrapper class of the component is defined in this package. This class implements the equivalent interface of the component, which LwCCM establishes as the only interface that can be used by clients or by the deployment tools to access to the component. With that 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

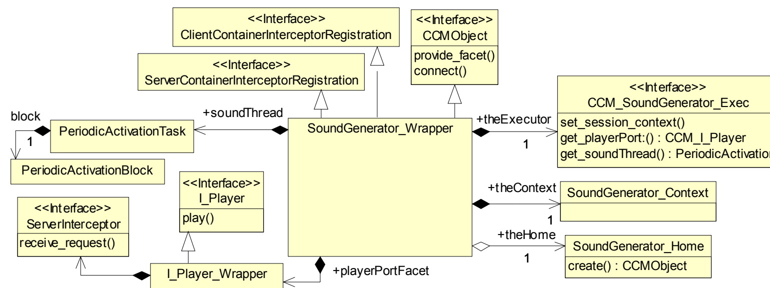

Fig. 6. Component wrapper structure

modified version of the homonymous interfaces defined in QoSCCM [10]. As it is shown in Figure 6 for the SoundGenerator component, this class is a container which aggregates or references all the elements that form the component:

- The component context, which includes all the resources required by the component to access to the components that are connected to its receptacles.
- The home, which represents the factory used to create the component instances.
- The executor of the component, which represents the link to the real business code implementation and whose structure is explained below.
- An instance of a facet wrapper class 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.
- Each activation port defined in the specification of the component represents a thread that is required by the component to implement its functionality. To implement those threads two kinds of Ada task types have been defined. The OneShotActivationTask executes the corresponding run() procedure of the port once, while the PeriodicActivationTask executes the update() procedure of the corresponding port periodically. Both types of task receive as a discriminant during its instantiation, a reference to the data structure that qualify their execution, which includes scheduling parameters, period, and the procedure to call. 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 the standard procedures that LwCCM specifies in the CCMObject interface to control the lifecycle of the component.

The rest of generated Ada packages represent the executor of the component. LwCCM defines a set of abstract classes and interfaces which have to be implemented, either automatically or by the user, to develop the executor 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 class for the component implementation which inherits directly from the interfaces defined in the previous package, and therefore, includes dependencies on the technology. As it is shown in Figure 7, this class contains as an aggregated object, the

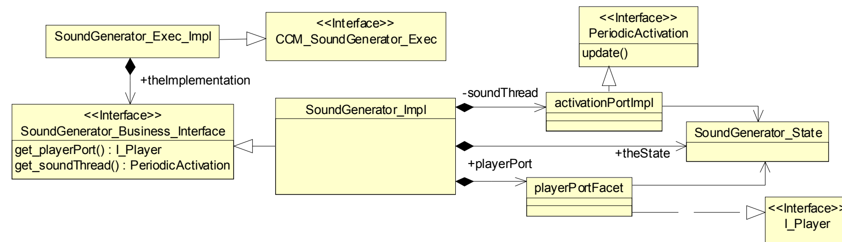

Fig. 7. Component executor structure

business code implementation of the component, which must implement the *{ComponentName}_Business_Interface* interface. As it has been said before, this interface includes all the methods that the business implementation must implement in order to be managed by the environment in an automatic way. By using this interface together with the aggregation pattern, the environment internals are hidden to the code developer, who is completely free to implement the business code of the component. The only requirement to meet is that the implementation must offer the {ComponentName}_Business_Interface interface, however, relevant aspects that should be included in a correct implementation are:

- For each facet offered by the component, a facet implementation object should be aggregated. In the case of simple components, the class itself can implement the interfaces supported by the facets.
- For each activation port defined in the component, the corresponding implementation object should be aggregated.
- All the implementation elements (facet implementations, activation ports, etc.) operate according to the state of the component, which is unique for each instance. Based on that, the state can be implemented as an independent aggregated class, which can be accessed by the rest of the elements, avoiding cyclic dependencies.

The current available Ada mapping for IDL [13] is based in Ada95, so for the development of the code generation tool, new mappings for some IDL types have been defined in order to get benefit of the new concepts introduced in Ada 2005. The main change concerns the usage of interfaces. The old mapping for the IDL "interface" type led to a complex structure, now, it can be directly mapped to an Ada interface.

4 Application development process

The development process of component-based applications, as it is shown in Figure 8, includes the design, configuration, deployment and launching of applications built as assemblies of components previously installed in the development environment.

The *assembler* describes the application as an assembly of component instances, selecting them among those stored in the repository of the design environment, and connecting them according to their requirements. The description is made by means of a Component Assembly Description (.cad file), as it is defined in the D&C specification. For real-time applications, this structural description must be complemented with the description of their workload. The workload of an application

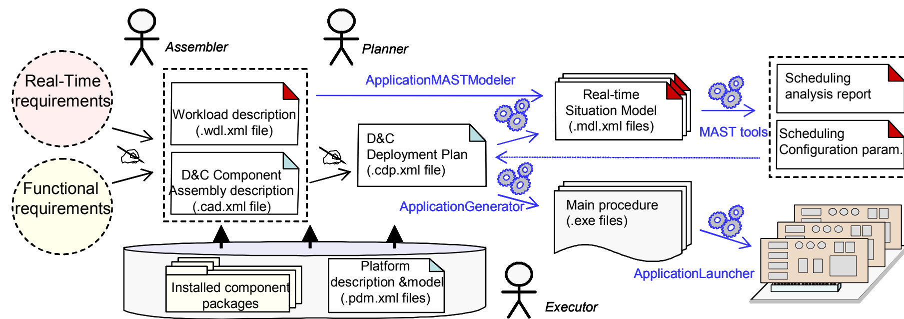

Fig. 8. Component-based application development process in AdaCCM

is defined as the set of real-time end-to-end flow transactions concurrently executed on it [14]. For each operational mode of the application with real-time requirements to meet (real-time situation), a workload model must be defined. Schedulability analysis tools can then be applied to each real-time situation. The real-time extension of the D&C includes also the definition of special metadata to describe the workload of an application.

In the next phase of the process, the *planner* takes the assembly description, and designs a deployment planning for the application. This process consists in assigning component instances to nodes, and deciding the mechanisms used for the communication between instances. The result of this stage is the deployment plan (.cdp file), which completely describes the application and the way in which it is planned to be executed.

At this point, a real-time design specific task is included in the process. The deployment plan defines the nature of the communication between component instances, assigning to each connection between component ports, the communication service to use and its corresponding configuration parameters (D&C has been extended to include this kind of information). These data will be used by the deployment tool to generate the corresponding connectors between components, but at this moment it is used to generate the real-time models of those connectors, whose templates should be stored in the repository or must be developed together with the deployment plan. Obviously, the communication services used for the connections must hold predictable behaviour. So, the deployment plan includes all the information required to generate the real time model of the complete application by composition of the real-time models of the components that form it, the platform resources (which must be also stored in the repository) and the connectors used for the interaction between components. This final model is used to calculate the optimal values for the scheduling parameters and to analyse the schedulability of the application under each workload.

Finally, in the last stage of the process, the *executor* makes use of a launching tool, which performs the following sequence of tasks:

- Using the deployment plan as input, it generates the code of the connectors involved in the application and the code of the main Ada procedures that have to be executed on each node in order to launch the application. These procedures instantiate, connect and configure the components and the connectors according to

the information defined in the deployment plan. They also include the configuration of the internal service of the execution environment (*SchedulingAttributeService)* which, together with the interceptors, manage in an automated way the scheduling parameters of the threads during the application execution. The configuration parameters of this service, whose values may be obtained by schedulability analysis or other verification techniques, are also specified in the deployment plan.

- The code of the generated main procedures is compiled and linked with the libraries corresponding to the components, and the code of the connectors involved in the application.
- The resulting executables are moved to and executed in the corresponding nodes.

5 Design environment and tools for components development

The design of a new component or the deployment of an application are processes which are performed in the development environment. They are to be assisted by tools to guarantee the correctness "by construction" of the generated artifacts. A design environment based in Eclipse has been defined for the Ada-CCM technology. It provides a set of frameworks and services which simplify resource management and tools development. The environment is composed of two key elements: the *repository*, in which the intermediate and final products are organized, and the tools which carry out the transformations between those products.

The information is organized in three Eclipse projects:

- The *repository* project is a general project. It stores and organizes the information relative to the registered elements. It is divided in five main sections: *applications*, *components*, *interfaces, platforms* and *technology.* Inside each section, the information is divided in domains which define different namespaces. Each element inside the repository is identified by the chain *<section>/<domain>/<name>/<extension>*.

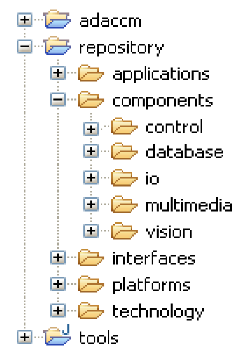

Fig. 9. Repository Structure

- The *adaccm* project is an Ada project. It stores the source or compiled Ada code which is required to build a deliverable component or execute an application. Its internal structure corresponds to the structure of Ada packages suitable for compiling and linking with the tools provided by the Ada development plug-in for Eclipse.
- The *tools* project is a Java project. It includes the code of the tools developed for the transformation processes. The currently developed tools are:
  - Tools for importing and exporting deliverable elements: *ComponentImport*, *ComponentExport, InterfaceImport* e *InterfaceExport*.

- Tools for components development: *ComponentTemplateGenerator* and *ComponentContainerGenerator*.
- Tools for application management: *ConnectorGenerator*, *ApplicationGenerator* *y ApplicationLauncher*.
- The tool which generates the final real-time model of an application: *ApplicationMastModeler*.

The Eclipse environment is provided with specialized editors, so it has not been necessary to develop specific tools for editing Ada source code, or the XML files corresponding to the D&C descriptors. For the latter, W3G-Schemas have been defined to facilitate the elaboration of this kind of files.

6 Execution Platform

Applications developed with Ada-CCM can be executed in distributed platforms which provide a run-time library with support for Ada applications. If the applications have hard real-time requirements, all the services of the run-time library and the communication mechanisms must have bounded response times. Likewise, for being able to port the applications to minimal embedded platforms, the run-time must be light, compatible with different targets (including microcontrollers), and it should not require a full file system or support for a hard disk. Figure 10a shows an example of the kind of applications that can be developed with this technology. It is an application whose purpose is to follow the trajectory of a moving object with a camera. The software architecture of the application is shown in Figure 10b. It is composed of six components which come from different application domains. The TrackFollower component plays the role of client component, since it is source of business end-to-end flow transactions. It has been specifically designed for this concrete application. The rest of the components have a broad scope of applicability, so they can be reused in different systems. The ServoController component performs the control closed loop of the n servos controllers. In the example, it controls the two degree-of-freedom of the camera orientation. The rest of the components (Tracker, Logger, IOCard and SoundGenerator) are leaves components and their function is to control different resources of the system (vision system, I/O cards, sound generators).

An application like this has been used to probe and experiment with the technology. It has been run on a MaRTE OS (Minimal Real Time Operating System for Embedded Applications) [15] target. MaRTE OS is a real-time kernel which follows the Minimal

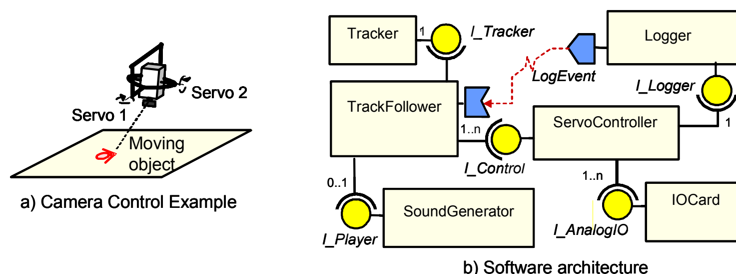

Fig. 10. Application example

Real-Time POSIX subset, defined in the IEEE 1003.13 standard. Besides, it offers support for hierarchical scheduling. The target hardware platform is any 386 PC or higher, with at least 512KByte of memory and with a device for booting the application (such as a floppy disk, flash memory, etc.), but not requiring a hard disk. Real-time communication mechanisms currently supported by MaRTE OS include CAN bus, and ethernet with the RT-EP protocol (Real-Time Ethernet Protocol) [8].

7 Real-time modelling and analysis of component-based systems

A real-time model is a timing abstraction that holds all the qualitative and quantitative information needed to predict/evaluate the timing behaviour of an application. It is used by designers to annotate timing requirements in the specification phase, to reason about the prospective architecture during design phases, and to guarantee its schedulability when the system is to be validated.

Software componentization is a structural pattern, which in principle is independent of the real-time design process, but, since it introduces deep changes in the development phases, it interferes with the traditional real-time design. An issue to consider in the component-based design strategies is the coordination between the structural (static) point of view, in which operations are identified as services of instances of components, and the reactive (dynamic) one, in which the activities (invocation of operations) are serialized in threads.

A real-time component must include metadata that allow a designer to predict its timing behaviour and analyse the schedulability of the applications that make use of it. The modelling methodology must provide two elements:

- Composable and formalized entities to hold the information about the timing and synchronization characteristics of the internal code of the component in a self-contained way and independent of any external elements (other components or platform resources).
- A systematic composition process that allows building the complete real-time model of an application using the models of its constituent parts: business components and platform resources.

An extension to the D&C specification is proposed to add real-time metadata to the descriptions of components, platform resources and applications. These metadata have been distributed according to the phase of the process in which they are required.

The D&C component interface description (.ccd file) includes the information about the temporal behaviour of a component that an application designer needs to decide the utility and the compatibility of the component inside an application. It declares:

- The set of operations with real-time behaviour offered by each facet. Any implementation of the component will include models for these operations. Likewise, for each receptacle of the component, the interface description must declare the operations whose real-time model is required to develop the real-time model of the component itself. Two components will be composable when the server component provides the real-time models of the operations that the client component requires through its receptacles.

- The parameters of the real-time model of the component. The real-time model of a component is a parameterized template which can be configured to describe the component behaviour according to the specific way in which the component is planned to be used in an application. Concrete values must be assigned to each parameter of each component instance declared in an application.
- Components with client role, i.e., components which can trigger business end-to-end flow transactions, must include the declaration of the kind of business transactions that they can initiate. The schedulability analysis of an application is performed regarding the workload of the application, and this workload is defined as the set of transactions concurrently executed in the application.

The D&C description of a component implementation (.cid file) must include the elements that describe the real-time behaviour of the internal code of the component:

- The execution time of the operations offered by the component. They are described through their worst, best, and average case values and are related to a reference processor, the one defined as having speed factor equals to 1.
- The synchronization resources that are used during the operations execution. They are necessary since they can cause blocking delays during execution.
- The scheduling entities in which the code execution is organized. These represent the execution capacity of the threads required to the environment.
- The description of the end-to-end flow transactions that are triggered in the component. Each transaction describes the set of activities that are executed in the system in response to external or timed events.

The D&C description of a platform (.tdm files) has been extended to include its real-time model. The platform model defines the models of the software resources (os, mutexes, drivers, etc.) and hardware resources (processors, networks, timers, etc.) that qualify and quantify the available processing capacity, the overheads associated to their management, the policies for the management of their access queues, etc.

The D&C description of the deployment plan that describes an application (.cdp files) incorporates two aspects regarding temporal behaviour:

- Each connection between component ports includes a reference to the corresponding real-time model of the connector used for the communication. The real-time model of a connector includes the information that describes the processes of marshalling and unmarshalling for the invocation parameters and return values, the activities involved in the transmission of messages and the processes of message dispatching.
- A deployment plan is associated with one or more declarations of the application workload. Each workload corresponds to a specific operation mode of the application that have timing requirements to meet, and for each of which schedulability analysis can be applied to verify that the requirements are met. Special metadata have been defined to declare the workload associated to an application.

Once an application is defined through a deployment plan, the planner can build its real-time model by means of the *ApplicationMastModeler* tool. As it is shown in Fig-

ure 8, this tool takes the information provided by the deployment plan, the metadata associated to the component descriptions and the metadata associated to the platform descriptions, and generates the final real-time model of the application. The real-time modelling and analysis methodology applied in Ada-CCM is MAST [14]. Specifically, an extension to MAST which incorporates the composability properties needed to generate the real-time model of a complex system by the composition of the individual real-time models of the software and hardware components that forms it [16].

MAST conceives the real-time model of an application as a description of its reactive behaviour. An application is modelled as a set of end-to-end flow transactions ("transactions" in MAST), which are sequences of activities that are triggered in response to external or timed events. A transaction is described by its set of activities, the generation pattern of the triggering events, and the timing requirements that must be met. The activities in different transactions only interact by sharing the processing-resources and the mutually exclusive passive resources.

The MAST environment includes several tools for real-time applications design:

- Schedulability analysis tools: They can be applied to both monoprocessor (RM Analysis y EDF Monoprocessor Analysis) and distributed systems (Holistic Analysis y Offset Based Analysis). They allow to certify that, in the worst case, the activities scheduled in the application meet their real-time requirements.
- Automatic priority assignment tools: Their usage is required, specially in distributed systems, when the amount of priorities or scheduling parameters to adjust makes the calculation process too complex to be developed without tool assistance. They can be applied to monoprocessor (Rate Monotonic and Deadline Monotonic) and distributed (Simulated Annealing and HOPA) platforms.
- Slack calculation tools: These tools calculate the percentage by which the execution time of the operations may be increased while keeping the system schedulable, or must be decreased to get the system schedulable.

8 Conclusions and future work

The proposed technology enables the development of hard real-time embedded component-based applications, whose temporal behaviour can be modelled and analysed by schedulability analysis tools. This is achieved by the combination and enhancement of well known technologies. (i) The usage of Ada makes the technology particularly suitable for applications that run in embedded nodes with limited resources, and interconnected with real-time communication networks. (ii) Some extensions introduced in the LwCCM container/component framework, together with the Ada's native support for concurrency and synchronization, provide the capacity of developing the code of the components with predictable temporal behaviour. (iii) The technology follows the D&C standard for the specification of components, platforms and applications. A real-time extension of D&C has been proposed to incorporate metadata describing the temporal behaviour of components and platforms. These metadata is used to analyse the schedulability of the application during the development process. The concepts and semantics added with the real-time extensions

allow the assembler or the planner to design the real-time aspects of the application without knowing the modelling methodology used by the analysis tools. The developer formulates the real-time models of the components following a concrete modelling and analysis methodology, which in the case of Ada-CCM is MAST.

Relevant future work concerns reducing the cost of real-time components design. They have to be designed so that their execution holds bounded timing behaviour, this behaviour should be modelled in detail, and all the execution times used in the model must be consciously evaluated. This latter task is currently very costly, though recent advances in techniques and tools promise to help reducing this cost in the future.

References

[1] R. Ommering, F. Linden, J. Kramer: The koala component model for con-sumer electronics software. In: IEEE Computer, IEEE (2000) 78-85.

[2] Lundbäck K-L., Lundbäck J., Lindberg M.: Component based development of dependable real-time applications Arcticus Systems, http://www.arcticus-systems.com

[3] Bondarev E., de With P., Chaudron M.: Predicting Real-Time Properties of Component-Based Applications In: Proc. of 10th RTCSA Conference, Goteborg, August 2004.

[4] M. Åkerholm et al.: The SAVE approach to component-based development of vehicular systems. In: Journal of Systems and Software, Vol. 80, 5, May 2007.

[5] OMG: Lightweight Corba Component Model, ptc/03-11-03, November 2003

[6] T. Taft et al. editors: Ada 2005 Reference Manual. Int. Standard ISO/IEC 8652/1995(E) with Technical Corrigendum 1 and Amendment 1. LNCS 43-48, Springer-Verlag 2006.

[7] OMG: Deployment and Configuration of Component-Based Distributed Applications Specification, version 4.0, Formal/06-04-02, April 2006

[8] J.M.Martínez and M. González.: RT-EP: A Fixed-Priority Real Time Communication Protocol over Standard Ethernet In: Proc. of the 10th Int. Conference on Reliable Software Technologies, Ada-Europe 2005, York(UK), June 2005

[9] L. Pautet and S. Tardieu.: 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, Newport Beach, USA, March 2000.

[10] OMG: Quality of Service for CORBA Components, ptc/06-04-05. April 2006

[11] OMG: Real-Time CORBA Specification, v1.2 formal/05-01-04. Enero 2005

[12] J.J.Gutiérrez García and M. González Harbour: 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, Junio 1999.

[13] OMG: Ada Language Mapping Specification - Version 1.2. October 2001

[14] M. González Harbour, J.J. Gutiérrez, J.C.Palencia and J.M.Drake: MAST: Modeling and Analysis Suite for Real-Time Applications. In: Proc. of the Euromicro Conference on Real-Time Systems, June 2001. http://mast.unican.es/

[15] M. Aldea and M. González.: MaRTE OS: An Ada Kernel for Real-Time Embedded Applications. In: Proc. of the International Conference on Reliable Software Technologies, Ada-Europe 2001, Leuven, Belgium, Springer LNCS 2043, May 2001. http://marte.unican.es/

[16] P. López, J.M. Drake, and J.L. Medina: Real-Time Modelling of Distributed Component-Based Applications In: Proc. of 32h Euromicro Conference on Software Engineering and Advanced Applications, Croatia, August 2006.