Scheduling Configuration of Real-Time Component-Based Applications[*]

Patricia López Martínez, Laura Barros, and José M. Drake

Departamento de Electrónica y Computadores, Universidad de Cantabria, 39005-Santander, Spain
{lopezpa,barrosl,drakej}@unican.es

Abstract. This paper proposes a strategy to manage the scheduling of real-time component-based applications that is fully compatible with the concept of component viewed as a reusable and opaque software module. The strategy is used on top of the RT-CCM technology, which extends the OMG's LwCCM technology with the purpose of building real-time distributed component-based applications that can be executed on embedded platforms and with heterogeneous communication services. The strategy is based on three services included in the RT-CCM framework, which are implemented by the containers of the components, and are in charge of supplying the threads and the synchronization artifacts that the business code of a component requires to implement its functionality. During the components configuration process, these services are used to assign the values that lead to a schedulable application to the scheduling parameters of these threads and synchronization mechanisms, without having to know the internal code of the components. The assigned values are obtained from the analysis of the real-time model of the application, which is built based on metadata provided by the components and the elements of the platform.

1 Introduction

The aim of component-based development is to build applications as assemblies of reusable software components that satisfy three characteristics: isolation (components are atomic units of deployment), composability (they should be composable with other components) and opacity (neither the environment nor other components or a third party can modify their codes) [1]. Several strategies have been proposed to apply this component-based paradigm to the development of real-time systems. This work relies on the RT-CCM (*Real-Time Container Component Model*) technology [2], which was initiated in former european projects [3][4], and which results from applying two extensions to the LwCCM (*Lightweight CORBA Component Model*) specification [5]:

- The interactions between components are generalized by specialized components, called connectors. They provide the communication mechanisms in their code,

[*] This work has been funded by the EU under contract FP7/NoE/214373 (ArtistDesign); and by the Spanish Government under grant TIN2008-06766-C03-03 (RT-MODEL). 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.

allowing the components to implement only the business logic. They do not have to be based on CORBA, as it is required in LwCCM.

- The interface and the implementations of a component include metadata related to their temporal behaviour, which are used to predict the temporal behaviour of the applications in which the component takes part.

RT-CCM presents two outstanding features: 1) The internal architecture of the components can be arbitrarily complex, and 2) the components incorporate all the informa-tion that is required to generate automatically the code of the containers and the connectors that adapt their codes to the corresponding platforms. As Figure 1 shows, an RT-CCM component is delivered as a package that includes, together with the code, the metadata that describe its functionality (described through the set of required and provided ports), its temporal behaviour, its configuration properties and its instantiation requirements. These metadata must be enough to generate the component container and the required connectors without modifying or accessing the internal code.

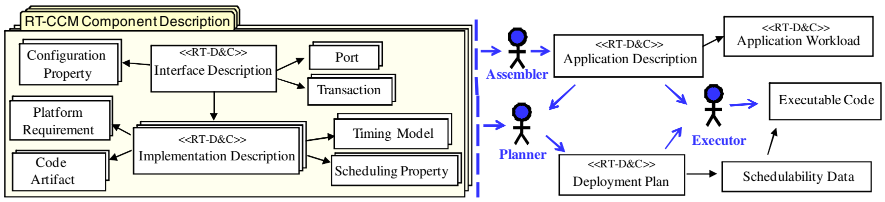

RT-CCM Component Description
Configuration Property
<<RT-D&C>> Interface Description
Port
Transaction
Platform Requirement
<<RT-D&C>> Implementation Description
Timing Model
Code Artifact
Scheduling Property
Assembler
Planner
<<RT-D&C>> Application Description
<<RT-D&C>> Application Workload
Executor
Executable Code
<<RT-D&C>> Deployment Plan
Schedulability Data

Fig. 1. Information provided by an RT-CCM component on parameters

For defining these metadata, RT-CCM relies on the RT-D&C extension [6]. The OMG's D&C specification [7] formalizes the formats and the contents of the documents that are used to describe a component and a component-based application. These documents are used by the application designers (*Assembler* and *Planner*) as a guide for accessing to the different pieces of information that are provided by the components and managed during the design and deployment process. RT-D&C extends D&C with the real-time metadata that are required to predict, analyse and configure the temporal behaviour of a component-based application during its design process.

The purpose of the real-time design of an application is to schedule the execution of its activities in order to satisfy the temporal requirements imposed on its specification. In traditional real-time design strategies, the designer has several means of getting a suitable scheduling for the application: (i) defining the concurrency level, i.e. the number of threads available for the execution of the application activities; (ii) assigning these activities to the threads in which they are scheduled; (iii) choosing the synchronization mechanisms required to coordinate the execution of the different threads; or (iv) assigning the policies and the scheduling parameters which are used as the basis to decide which thread accesses to the processing, synchronization or communication mechanisms when several threads compete for them.

When a component-based strategy is applied, the designer manages the application at a higher level of abstraction. All the previous aspects are related with the internal code of the components, so they are unknown and inaccessible for the designer due to the opacity required for managing the components. In this case, the designers can control or configure the execution only by means of the information included in the deployment plan. Through the deployment plan, the planner chooses concrete implementations for each instance that forms the application, assigns concrete values to their configuration parameters, maps the instances to the nodes and selects the communication mechanisms used for each connection between components. In a real-time application, the deployment plan must also include all the information required to configure the execution of the application so that the specified timing requirements are met.

This work presents the design process of a real-time component-based application according to the RT-CCM technology, which requires the explanation of:

- The mechanisms that have been introduced in the framework to control the scheduling of the applications.
- The models that describe the temporal behaviour of the individual components, which are used to generate the reactive model of the complete application. This final model serves as the basis to evaluate the scheduling configuration.
- The timing metadata that are associated to the components and to the deployment plans in order to configure the application scheduling in an opaque way, and based on the results extracted from the analysis.

Related work. Several works aim to achieve temporal predictability for the applications based on the metadata provided by the components. The temporal models and the associated composition process are well defined in [8], however the applied analysis is focused on performance, and it does not offer a way of configuring the temporal behaviour of the components based on the analysis results. Its concept of component is similar to ours: a reusable module that can offer a complex functionality by implementing different services and responses to events. Other approaches, specially focused on control systems [9][10], are based on lower granularity components that implement basic passive functions. In this case, the real-time design consists in composing and mapping them to the environment threads and assigning priorities to each thread. Similar to our approach, CIAO [11] implements the LwCCM specification and uses D&C to configure and deploy applications. It offers capacity for configuring real-time characteristics of the applications by using RT-CORBA features. However, they do not follow any strategy to obtain this information from the analysis. The configuration is made directly on the deployment plan based on the designer expertise.

2 An RT-CCM Application Example

Figure 2 shows the components architecture and a possible deployment of the ScadaDemo application: an RT-CCM application that is used along the paper as an example to introduce the proposed concepts[1]. Its functionality consists in supervising a set of analog environment magnitudes, and storing statistical data about them in a logger. The operator can choose the magnitudes to supervise through the keyboard, and he can also choose one of them to be periodically refreshed in the monitor.

[1] Since this paper deals with the real-time design strategy and the schedulability configuration mechanisms, for reasons of space, the used example is monoprocessor, although RT-CCM is specially focused on distributed applications and the same strategy can be applied to them.

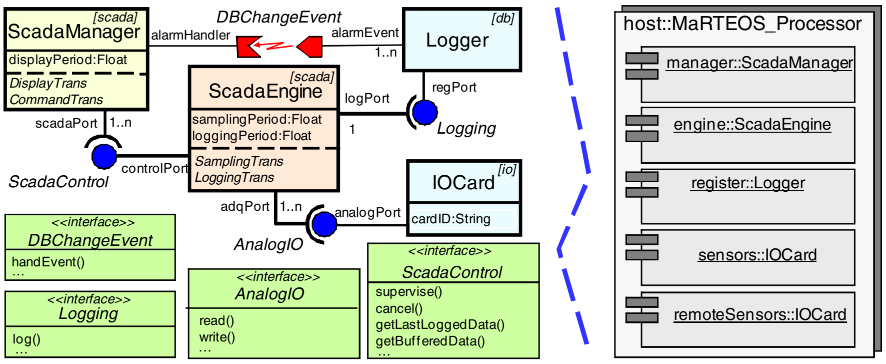


Fig. 2. Architecture (left side) and deployment (right side) of the ScadaDemo application

The application is built by assembling instances of four different types of components, organized in a three-tier architecture. The data tier is composed by leaf components: *IOCard*, which manages the acquisition cards used to read the analog signals, and *Logger*, which allows registering data with timing marks in a permanent data base. The business logic tier is formed by a component of *ScadaEngine* type, which implements a standard SCADA (Supervisory Control and Data Acquisition) functionality. It supervises periodically (with a configurable period) a set of magnitudes, processes statistically the values read through the *adqPort* port, and registers the obtained results through the *logPort* port. Finally, the presentation tier is implemented by the *ScadaManager* component, which implements the specific functionality of this application, processing the commands introduced by the operator and displaying periodically the results of one of the magnitudes. Figure 2 shows also the basic functionality of each component type, by defining their provided and required ports, called facets and receptacles respectively, together with the interfaces that these ports implement.

In RT-CCM, the functionality of an application is specified in a reactive way, as the set of transactions that it executes concurrently in response to external or timed events. In this case, there are four transactions, three of them with timing requirements:

1. Sampling transaction: Represents the periodic (*samplingPeriod* period) activity through which the value of each magnitude is read and statistically processed.
2. Logging transaction: Every *loggingPeriod* period (higher than *samplingPeriod*) the information of all the supervised magnitudes is registered in the logger.
3. Display transaction: The information about one of the magnitudes is updated in the monitor every *displayPeriod* period.
4. Command transaction: The commands introduced by the operator are attended and executed, without timing requirements.

3 Threads and Components

The business code of a component implements the set of services offered through the component facets, and the responses to the events that the component attends. In the

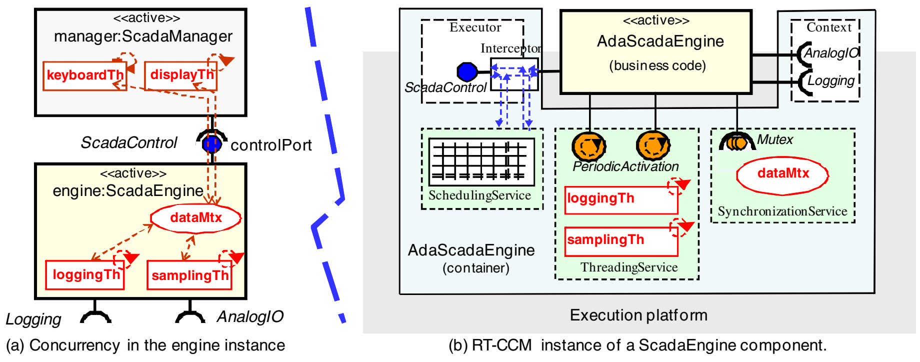


Fig. 3. Threads that concur in the engine instance of the ScadaDemo application

general case, this code requires to be concurrently executed by multiple threads, which are created either by the component itself or by other components that invoke its services. As a consequence, the code of a component must include internal synchronization mechanisms to guarantee mutually exclusive access to the shared resources, and to synchronize the threads that concur in it.

As an example, Figure 3 (a) shows the four threads that concur in the *engine* instance (of ScadaEngine component type) during the execution of the ScadaDemo application. Two of them are internal to the component: *samplingTh*, which reads periodically the supervised magnitudes, and *loggingTh*, which registers periodically the statistical values in the logger. The other two threads come from invocations made by the *manager* instance (of ScadaManager component type) in the *controlPort* facet: *keyboardTh* modifies the list of variables to supervise, and *displayTh* requires the information about the magnitude to display. The four threads require access to some internal data hold by the component, so they are synchronized by a mutex called *dataMtx*.

The real-time design and schedulability analysis of component-based applications require knowledge about these threads, their associated synchronization mechanisms and the activities that they execute. Besides, configuring the scheduling of an application requires that the parameters that control the scheduling of the different elements can be assigned by the configuration and launching tools. To make the inherent opacity of components compatible with the real-time design process, in RT-CCM the creation and management of threads and synchronization artifacts have been extracted from the code of the components. They are implemented by the containers, whose codes are known and accessible by the tools.

RT-CCM uses three elements to facilitate the scheduling of the threads in that opaque way, without knowing the code of the components:

- Four new port types have been defined. They are used by the business code of the components to access to the threads and the synchronization mechanisms provided by the container.
- The activities assigned to each thread and the points in which these threads synchronize their flows are described by the real-time model of the component.

- Interceptors are used to establish the scheduling parameters with which each service invoked in a component is executed, based on the concrete transaction and the point inside the transaction in which the invocation is made.

The new types of ports defined in the RT-CCM technology are:

- *PeriodicActivation* ports: For each port of this type declared by a component, the container (by means of the *ThreadingService*) creates a thread that periodically executes the *update()* method offered by the port. The activity executed by this method must have a finite duration (lower than the activation period).
- *OneShotActivation* port: For each port of this type declared by a component, the container creates a thread that executes the *run()* method offered by the port once, when the component is activated. The method execution can have an arbitrary duration, which can last the complete active life of the component.
- *Mutex* port: For each declared port of this type, the container (by means of the *SynchronizationService*) creates a mutex, which the component can manage through the *lock()* and *unlock()* methods invoked on it through the port.
- *ConditionVariable* port: For each port of this type the container (by means of the *SynchronizationService*) creates a condition variable, which the component can use to suspend and activate the internal threads, through the *wait*() and *notify*() methods invoked on it through the port.

Figure 3(b) shows the ports through which the *AdaScadaEngine* implementation, the Ada 2005 implementation of the ScadaEngine component used in the ScadaDemo example, requires the two internal threads, *samplingTh* and *loggingTh* ports, and the mutex, *dataMtx* port, which it needs to implement its functionality.

In RT-CCM a component is a complex module whose temporal response is not defined by its own code in the general case, since this temporal behaviour depends on the temporal behaviour of other components that it uses, and on the final execution platform. Therefore, the aim of the real-time model of a component is not to describe the temporal behaviour of the component as an isolated module, but to provide the information about the component that is required to predict the temporal behaviour of any application in which it may be used. Each component implementation has its own real-time model, which basically contains the declaration of the processing and synchronization resources that the component uses to implement its offered services and its responses to external or timed events. The execution of a component service or the response to a received event are described as the execution of a set of activities ordered by control flow relations (sequential, fork, branch, merge and join). Each activity is described by enumerating the resources that it uses and the corresponding usage times.

In RT-CCM these models are formulated according to CBSE-MAST [12], an extension of the MAST [13] modelling methodology. It incorporates parameterization and composability to the components models, which facilitate their composition into full reactive models able to be analysed with the MAST set of tools. The modelling concepts used in MAST are almost the same as the ones proposed in the SAM (*Schedulability Analysis Modelling*) chapter of the OMG's MARTE [14] profile. Figure 4 represents the real-time model of the AdaScadaEngine implementation. The model includes the SchedulingServers that describe the *samplingTh* and *loggingTh* threads that

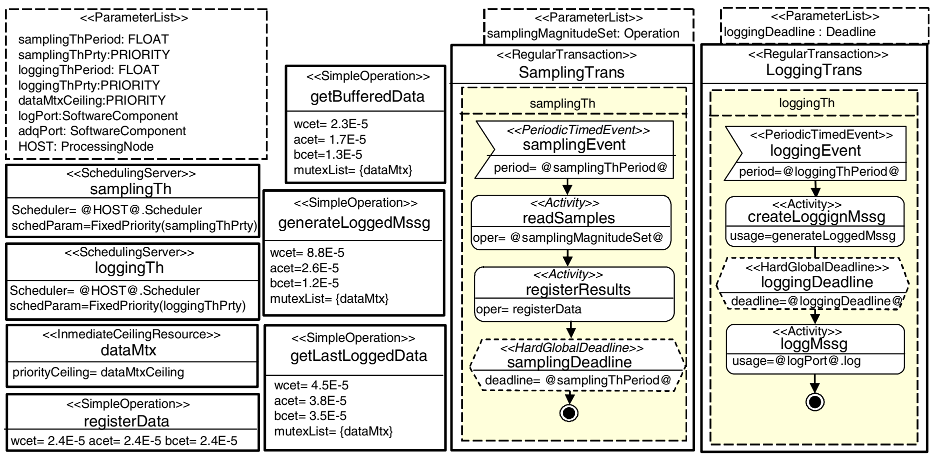


Fig. 4. Elements of the real-time model of the AdaScadaEngine implementation

the component uses internally. The model states that they are scheduled by the scheduler of the processor in which the component is installed (HOST reference) and they are managed according to a fixed priority preemptive scheduling policy. The model declares a SharedResource to describe the *dataMtx* mutex, which uses a priority ceiling policy. The models of the *getBufferedData* and *getLastLoggedData* real-time services, offered through the controlPort facet, are also included. Besides, the model describes as transactions, the two periodic activities executed internally in the component, *SamplingTrans* and *LoggingTrans*. They correspond to the code executed in the two update() methods of the two declared PeriodicActivation ports. Each transaction defines the SchedulingServer in which it is scheduled (*samplingTh* and *loggingTh* respectively), the generation pattern of the triggering events, the sequence of activities that are executed in response to those triggering events, and the timing requirements that must be met in each execution. Finally, the model includes some internal operations that are used to describe the transactions activities, as for example *registerData*.

The real-time model of a component is parameterized in three different aspects:

- The real-time model of the processor in which the component is executed is not known at development time, so it is referenced through the HOST predefined parameter. The execution times of the internal activities of the component are formulated as normalized execution times, so only in the context of an application deployment, when the processor and its processing capacity are known, the actual physical execution times can be obtained.
- The resources usages generated by those component activities that consist in invoking services on other components (via receptacles) can not be specified at component development time. The model includes only a reference to the port and the service invoked. Only in an application, when the concrete connected component is known, the actual resource usages can be evaluated. This is the case e.g. of the invocation of the *log* method (in *logPort*) in the *LoggingTrans* model.

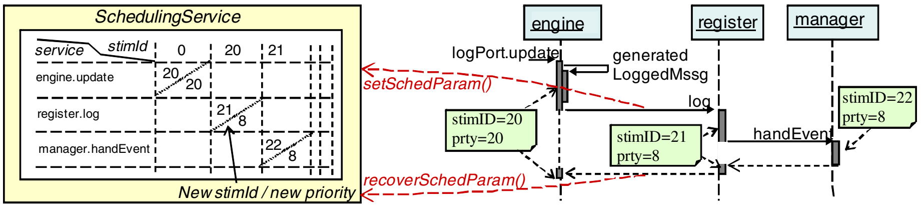


Fig. 5. Management of scheduling parameters in RT-CCM

- There are some characteristics of the real-time model that are declared as parameters to adapt them to the specific situation in which the component is instantiated in a concrete application context. In the example, the *samplingThPeriod* and *loggingThPeriod* are parameters of the real-time model since different values can be assigned to them according to the concrete application context.

To get a more flexible scheduling of the applications, the execution of the services and the responses to events that the components manage can be scheduled according to parameters (priorities, preemption levels, etc.) that depend on the concrete transaction and the internal state of the transaction in which they are executed [15]. To make this scheduling easier, each of the transactions executed in the application are associated with an independent thread when they are sequential, or with a set of threads when they are concurrent. When the timing requirements are assigned to the finalization of the last activity of the transaction, the scheduling can be associated to the threads, making all the activities to execute with the same scheduling parameters. However, when the requirements are associated to intermediate points of the transaction, the scheduling parameters must be independently assigned for each activity.

In RT-CCM, this kind of management of the scheduling parameters in an activity-based manner, is managed by means of interceptors. This mechanism [16] is used to manage non-functional aspects, by invoking environment services before and after the execution of a service in a component. In RT-CCM, before a component service is executed, the interceptor calls the *setSchedParam()* method in the *SchedulingService* of the container. This method receives as argument an identifier (called stimId) of the transaction and the transaction state, which is used to modify the scheduling parameters of the invoking thread and the value of stimId. When the service execution ends, the interceptor invokes the *recoverSchedParam()* method, which recovers the value of stimId and the scheduling parameters that the thread had when the invocation was made. To clarify this concept, Figure 5 shows the scheduling management in the case of the Logging transaction of the ScadaDemo example. The *update()* method of the loggingTh port of the *engine* instance (which maps the Logging transaction) is started with stimId = 20 and priority = 20. The invocation of the *log* service in the *register* instance is made with that value of stimId. When the interceptor associated to this service invokes *setSchedParam()*, according to the configuration of the SchedulingService established for the application, the priority of the invoking thread is set to 8 and the stimId value is changed to 21 (in order to distinguish other invocations made inside the *log* service, as the *handEvent* invocation on the *manager* instance). When the execution of *log* ends, the initial values for the stimId and the priority are recovered, by invoking the *recoverSchedParam()* method.

4 Real-Time Components Description

An RT-CCM component is described by means of the descriptors formalized in the RT-D&C specification. According to it, the information that describes a component is split up in two parts: the component interface and the component implementations.

The RT-D&C's ComponentInterfaceDescription contains the information about the functionality, connectivity and configurability of the component that is common to all its possible implementations. It provides the information required by the assembler to decide if a component is useful in the application that is being designed. In real-time applications, some specific aspects of this description must be remarked:

- The functionality of a real-time application is described by means of a reactive specification, which defines the external or timed events to which the application responds, the activities that constitute these responses and the timing requirements that must be met. As a consequence, the RT-D&C descriptor for a component interface must include metadata declaring the kind of responses to the events that the component can attend, i.e. the transactions generated in the component.
- The connectivity of the components must be described not only from the functional point of view (based on interfaces compatibility) but also from the real-time models composability point of view. Some metadata must be included to guarantee that an assembly of components leads to a real-time composed component, i.e. a component for which a temporal behaviour model can be generated. With that aim, in RT-D&C, each facet declares the operations for which the component provides a real-time model, and each receptacle declares the operations whose real-time models are needed by the component. In an application, a facet of a component can be connected to a receptacle of another component only if all the real-time models of operations required by the facet are provided by the receptacle.

The RT-D&C's ComponentImplementationDescription contains the specific information of an implementation of the component interface. The planner obtains from it the information that he requires to decide the deployment of an instance of the component in a concrete node of the platform. In case of a real-time component, it has to include metadata that allow the planner to configure the application scheduling:

- Declaration of the threads required by the component to implement its functionality, formulated as activation ports.
- Declaration of the synchronization artifacts required by the component to manage concurrent accesses to its internal data, formulated as synchronization ports.
- Declaration of the scheduling configuration parameters. Each implementation can declare the priorities, preemption levels, priority ceilings, etc. that must be configured to control the scheduling of the threads and the synchronization mechanisms required by the component.

Besides, the implementation description must include the reactive model that describes the temporal behaviour of the component. This model must include:

- Temporal behaviour models of the services offered through the facets. Each model describes the sequence of activities executed when the service is invoked.

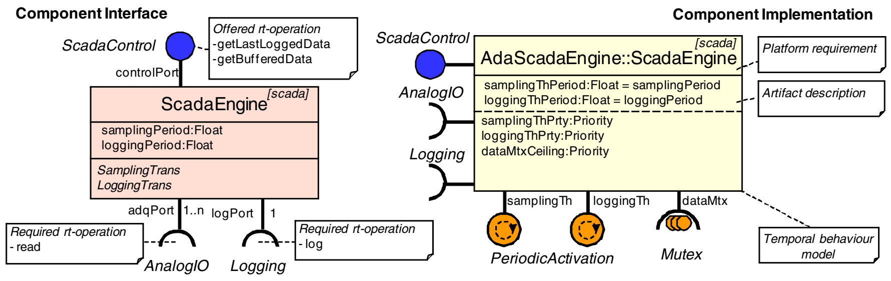


Fig. 6. ScadaEngine interface and AdaScadaEngine Implementation

- Temporal behaviour models of the transactions initiated by the component in response to external or timed events. They are described also as the sequence of activities executed in them, together with the triggering patterns and the timing requirements imposed on their execution.

Figure 6 shows the elements that constitute the interface of the ScadaEngine component, and the description of the AdaScadaEngine implementation. The ScadaEngine interface description includes the following elements:

- The *controlPort* facet that implements the *ScadaControl* interface. The component declares that it offers associated real-time models for the *getBufferedData* and *getLastLoggedData* services.
- For implementing its functionality, the component needs to be connected to at least one component implementing the *AnalogIO* interface through the *adqPort*. It requires that the connected component offers a real-time model for the *read* service. It declares another required port, *logPort*, which must be connected to a component implementing the *Logging* interface. In the case of real-time applications, the connected component must provide a temporal model for the *log* service.
- The component defines two configuration parameters, *samplingPeriod* and *loggingPeriod*, for adapting its business behaviour to the corresponding application.
- The component responds to two different events, through the *SamplingTrans* and *LoggingTrans* transactions. These transactions map the ones required by the application that were identified in section 2 (Sampling and Logging transactions).

The description of the AdaScadaEngine implementation specifies:

- The requirements that the component imposes on a processor to be instantiated.
- The description of the artifacts which hold the implementation code, and the way in which they have to be instantiated and executed.
- The description of the activation ports. As it was said in Section 3, the AdaScadaEngine implementation requires two threads, which execute the *update()* method of the corresponding *samplingTh* and *loggingTh* ports.
- The description of the synchronization ports. The AdaScadaEngine implementation requires a mutex, which is accessed through the *dataMtx* port.
- Implementation configuration parameters: due to the periodic activation ports, two new parameters appear at this level, *samplingThPeriod* and *loggingThPeriod*,

which describe the activation periods for the ports. They take values directly from the corresponding samplingPeriod and loggingPeriod parameters.

- Scheduling configuration parameters: The implementation introduces three scheduling parameters (also due to the activation and synchronization ports), *samplingThPrty*, *loggingThPrty* and *dataMtxCeiling*, which are used to contro-lytyuthe scheduling of the internal activities of the component.
- The reference to the file in which the reactive temporal model of the component is described, whose main elements were introduced in section 3. As shown in Figure 4, its parameters correspond to the implementation and scheduling configuration parameters defined in the component implementation (*sampling/loggingThPeriod, sampling/loggingThPrty and dataMtxCeiling*).

5 Real-Time Design of a Component-Based Application in RT-CCM

Figure 7 shows the artifacts and the actors involved in the process of development of an application based on the RT-CCM technology. The *assembler* is a domain expert who implements the functionality of the application by assembling instances of the available components, based on the metadata provided by their interfaces. He generates the description of the application as a composed component. The *planner* decides the processor in which each instance is going to be instantiated, selects the appropriate implementation for each instance and chooses the communication mechanism to use for each connection between instances. The planner describes the application as a deployment plan. Finally, the *executor* generates the executable codes for each node, transfers and installs them in the corresponding nodes and starts the application.

When the designed application has real-time requirements, some specific characteristics are added to this process:

- The specification of the application has a reactive nature, so the assembler must build the application choosing firstly those components that have the capacity to manage the events to which the application responds. This is the cause of selecting e.g. the ScadaEngine component in the ScadaDemo example, since it implements two of the required transactions. Then, he may have to choose other components to satisfy the connectivity requirements of the chosen components, as it happens with the IOCard and Logger components in ScadaDemo, which are chosen to fulfil the functionality of the ScadaEngine component.
- The assembler must formulate the real-time requirements of the application in the context of one or several workloads, i.e. he must specify the generation patterns for the events that lead the execution of the application, and the timing requirements that must be satisfied during the execution.
- The assembler builds the application ensuring that it implements the functional specification. However, he can not guarantee that the timing requirements are going to be met, since the timing behaviour of the application depends on the execution platform and the concrete component implementations chosen. He can only assure that a real-time model of the application can be obtained, by checking the composability of the real-time models of the components involved.

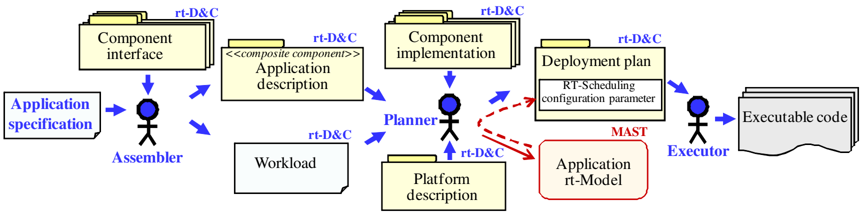


Fig. 7. Development process of an RT-CCM component-based application

- The planner is responsible of making the application schedulable. Besides assigning instances to nodes and choosing the concrete implementations, he must assign values to the scheduling parameters of the components and the platform resources so that the timing requirements of the application are met.

This last aspect is identified as the real-time design of a component-based application. It is a complex task, which requires having the temporal behaviour model of the complete application, in order to use it as the basis for applying real-time design algorithms and tools. These tools are used to obtain optimal values for the scheduling parameters and to analyse the schedulability of the application, certifying the fulfilment of the timing requirements. Figure 8 shows the final model that results from the deployment of the ScadaDemo application shown in Figure 2, with a workload that corresponds to a situation in which three magnitudes are supervised with a sampling period of 10 ms, a logging Period of 100 ms and a display period of 1s. The processor considered in this case has a speedFactor = 0.5 (i.e. with half the capacity of the processor taken as reference for specifying the normalized execution times in the real-time models of the components) [13][14] and its scheduler uses a fixed priority policy.

The real-time model of the application is automatically generated by a tool that takes as inputs the deployment plan, and the descriptor of the execution platform. The model is built by identifying the component instances that form the application and composing the temporal models that they include in their descriptions. In this final model, all the references and parameters are solved since all the connections between components are known and also the capacity of the platform. Although the generation of the real-time model of the application is guided by the metadata and the parameterized models included in the component packages, the result is a conventional real-time model, which can be analysed with standard real-time design and analysis tools. In our case, the tools provided by the MAST environment are used.

The results obtained from the real-time design process in RT-CCM are:

- The initial priority for all the threads required by the components.
- The priority ceilings that must be assigned to each required mutex.
- The priority of execution of each invocation received in a component service in the context of the transaction in which the invocation is made. The tool generates the sequences of stimId values that identify univocally each received invocation.
- The stimId values that the active components must use to identify each transaction that is triggered on them. They are given to the threads that execute the activation ports of the components.

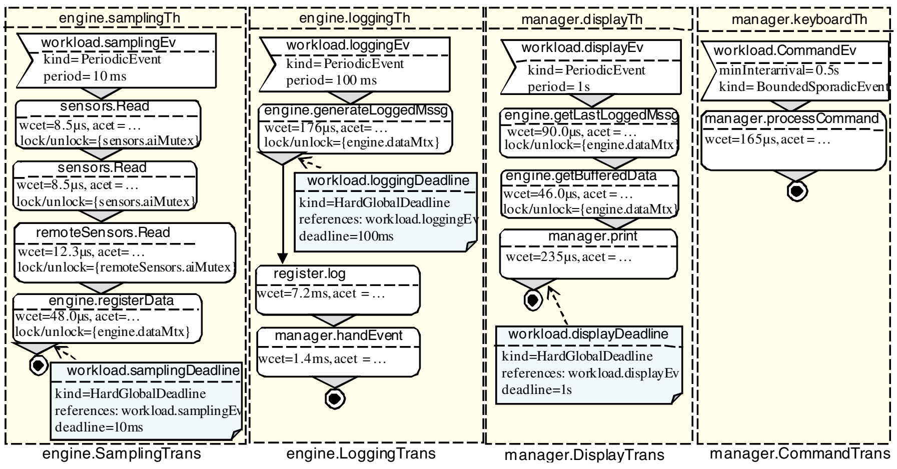
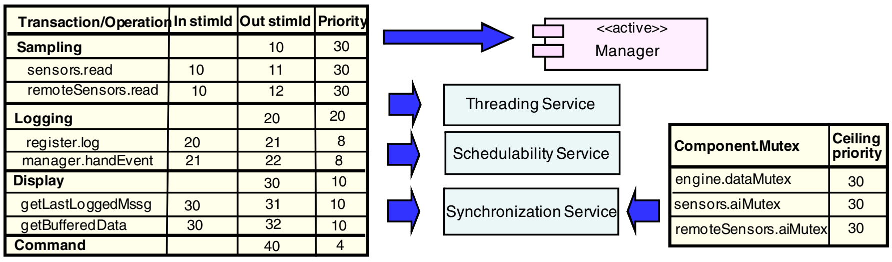


Fig. 8. Real-time model of the deployed ScadaDemo Application



Transaction/Operation	In stimId	Out stimId	Priority
Sampling		10	30
sensors.read	10	11	30
remoteSensors.read	10	12	30
Logging		20	20
register.log	20	21	8
manager.handEvent	21	22	8
Display		30	10
getLastLoggedMssg	30	31	10
getBufferedData	30	32	10
Command		40	4

Component.Mutex	Ceiling priority
engine.dataMutex	30
sensors.aiMutex	30
remoteSensors.aiMutex	30

Fig. 9. Schedulability configuration results in the ScadaDemo example

The tables in Figure 9 show the priority assignment obtained for the ScadaDemo example. The Sampling and Display transactions have deadlines associated to the end of the transaction, so the assigned priority is the same for all the activities executed in them. However, the Logging transaction has a deadline associated to an intermediate state, so the activity *register.log*, which is executed after it, receives a lower priority than the rest of the transaction. Based on this information, the launching tool that starts the application execution by instantiating the components, configures the platform services in each node with the following information:

- The ThreadingService receives the initial priority and stimId with which each thread provided to a component through an activation port must start its execution. Each stimId identifies univocally one transaction.
- The SynchronizationService receives the configuration parameters for each of the synchronization artifacts required by the components (e.g. mutex ceilings).
- The SchedulingService of each node receives the tables with the mappings between stimId and priority values, which are used to configure each invocation made on a component service.

With this strategy, the RT-CCM technology satisfies the objective of configuring the scheduling of an application respecting the opacity of the components.

6 Conclusions

The strategy proposed in this paper allows the designer of a real-time component-based application to configure its scheduling, satisfying the opacity requirement typical of the components paradigm. The strategy is applied on top of the RT-CCM technology, which uses a container/component model to extract the scheduling management from the business code of the components. The management of all the aspects related with the application scheduling are carried out by a set of services included in the components containers. Besides, the technology relies on an extension of the D&C specification, which incorporates metadata about the temporal behaviour of components and platforms. These metadata allow the designers of the applications to analyse their temporal behaviour, or to extract from this analysis the scheduling configuration parameters to be assigned to the component instances, through the services, in order to guarantee the fulfilment of the timing requirements of the application.

References

[1] Crnkovic, I., Larsson, M.: Building Reliable Component-Based Software Systems. Artech House Publishers (2002)

[2] López, P., Drake, J.M., Pacheco, P., Medina, J.L.: An Ada 2005 Technology for Distributed and Real-Time Component-based Applications. In: Kordon, F., Vardanega, T. (eds.) Ada-Europe 2008. LNCS, vol. 5026, pp. 254–267. Springer, Heidelberg (2008)

[3] IST project COMPARE: Component-based approach for real-time and embedded systems, http://www.ist-compare.org

[4] IST project FRESCOR: Framework for Real-time Embedded Systems based on Contracts, http://www.frescor.org

[5] OMG.: CORBA Component Model Specification. formal/06-04-01 (April 2006)

[6] López Martínez, P., et al.: Real-time Extensions to the OMG's Deployment and Configuration of Component-based Distributed Applications. OMG's 9th Work. Distributed Object Computing for Real-time and Embedded Systems, Arlington, VA, USA (2008)

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

[8] Bondarev, E., de Withy, P., Chaudron, M.: CARAT: a Toolkit for Design and Performance Analysis of Component-Based Embedded Systems. DATE Europe Conference (2007)

[9] Åkerholm, M., et al.: The SAVE approach to component-based development of vehicular systems. Journal of Systems and Software 80 (May 2007)

[10] Angelov, C., Sierszeckiy, K., Zhou, F.: A Software Framework for Hard Real-Time Distributed Embedded Systems. In: Proc. 34th Euromicro Conf. Software Engineering and Advanced Applications, Parma (August 2008)

[11] Kavimandany, A., Gokhale, A.: Automated Middleware QoS Configuration Techniques for Distributed Real-time and Embedded Systems, April 2008. Real-Time and Embedded Technology and Applications Symposium. IEEE Computer Society Press, Los Alamitos (2008)

[12] López, P., Drake, J.M., Medina, J.L.: Real-Time Modelling of Distributed Component-Based Applications. In: Proc. of 32h Euromicro Conf. on Software Engineering and Advanced Applications, Croatia (August 2006)
[13] González Harbour, M., et al.: MAST: Modeling and Analysis Suite for Real-Time Applications. In: Proc. of the Euromicro Conference on Real-Time Systems (June 2001)
[14] OMG: UML Profile for Modeling and Analysis of Real-Time and Embedded systems (MARTE), OMG doc. formal/2009-11-02 (November 2009)
[15] Gutiérrez García, J.J., González Harbour, M.: Prioritizing Remote Procedure Calls in Ada Distributed Systems. ACM Ada Letters XIX(2), 67–72 (1999)
[16] OMG: Quality of Service for CORBA Components, ptc/06-04-05 (April 2006)