Real-Time Modelling of Distributed Component-based Applications

Patricia López, Julio L. Medina* and José M. Drake

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

Abstract

This paper presents a modular modelling methodology to formulate the timing behaviour of real-time distributed component-based applications. It allows to build real-time models of the platform resources and software components, which are reusable and independent of the applications that use them. The proposed methodology satisfies the completeness, opacity and composability properties, required to ensure that the complete real-time model of an application, able to predict its temporal behaviour by schedulability analysis or simulation, may be assembled by composition of the real-time models of its constituent parts. These real-time models present a dual descriptor/instance based nature. A class of component, independent of any application, is modelled as a parameterized class-type descriptor, which describes its inherent temporal behaviour and includes references to the real-time models of other hardware/software modules that it requires. An instance of the component in a concrete application context is modelled by an instance-type model, which is generated by assigning concrete values to the parameters and unsolved references of its corresponding descriptor. Instances are formed and combined by automatic tools to build complete analysis models for each specific real-time situation.

1. Introduction[1]

The real-time model of a software application is an abstraction that holds all the qualitative and quantitative information needed to predict/evaluate its timing behaviour. It is used by designers to annotate timing requirements in the specification phase, to reason about the prospective architecture during design phases and to certify its schedulability when the solution is to be validated. This work elaborates a modelling methodology founded on the conceptual model known as the "Transactional Model" [1][2], used for analysis, specification, and design of real-time systems. A number of tools and techniques for schedulability analysis and response time calculation have been proposed for it [3][4], and its fundaments are referenced by the synchronization protocols and scheduling policies used in the vast majority of real-time as well as general purposes operating systems [5]. Besides, the importance of the transactional model as a reference for analysis may also be noted by looking at the underlying model of the “UML profile for Schedulability, Performance and Time” (SPT), current OMG standard for modelling and analysis of real-time systems [6]. The transactional methodology models a real-time application by two complementary descriptions:

a) Control flow (transactional) model: It is a reactive model, which describes the application as a set of concurrent real-time transactions, which are sequences of activities that are triggered in response to external or timed events. A transaction is described by its causal flow of activities, the generation pattern of the triggering events, and the timing requirements that must be met. An activity describes the amount of processing capacity that is required to perform the duty that it has associated. There is no direct activation or execution flow dependency between activities in different transactions; they only interact by sharing the processing-resources and the mutually exclusive passive resources.

b) Resources contention model: It describes the active and passive resources that are used by the activities in a mutually exclusive way, showing characteristics like their capacity, overheads, access protocols or scheduling policies. It is used to evaluate the blocking time in the access to passive synchronization resources or while contending for active resources like processors or networks.

The information in the transactional model of a real-time application tend to be complex, therefore, tool support is required for its processing and management, and useful to exploit it effectively for analysis and design.

Traditionally, the architecture of real-time applications has followed straightforwardly the transactional model, and they have been programmed with bare RTOS services. Currently, the increasing complexity and evolution of real-time applications domains, and the necessity for managing the software production, are pushing the introduction of component-based strategies in the construction of OS, middleware, and applications with real-time constraints.

1. This work has been funded by the *Comisión Interministerial de Ciencia y Tecnología* of the Spanish Government under grant TIC2002-04123-C03-02

*Post-doctoral internship in the *Commissariat à l’Energie Atomic*, CEA Saclay, DRT/DTSI/SOL/L-LSP, F-91191, Gif-sur-Yvette Cedex, France

The "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 methodology, it interferes with the methods used in the real-time design. Traditional real-time design methodologies [7][8] conceive applications as concurrent sets of transactions in a reactive paradigm, and it is in a later phase when the code is organized in modules following some domain criteria, such as objects, tasks or subsystems. On the contrary, component-based systems are design using reusable modules selected from catalogues in accordance with the required functionality. It is later, in refinement phases, when control flow lines are identified and associated to threads, being not unique the concurrency model that can be obtained. Hence, an issue to consider in the component-based design strategies is the compatibility 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 organized in threads, tasks or processes [9][10].

A modelling methodology used to describe real-time applications that are designed using component-based techniques still needs to be oriented to describe its reactive transactional model, but it must also use modelling container elements that may be identified with its components structure. For this reason, it has to bring elements to formulate the real-time model of a component as a self-contained set of abstractions and data that describe the timing and synchronization characteristics inherent to its own code and nature, and complete enough to build its corresponding part in the real-time model of any application that may use it. The modelling methodology must also offer the composability properties required to build the complete real-time model of the application using the models of its constituent components, in analogy to the composition of the application code with the code of the components used.

Previous work [11][12] has proposed MAST (Modelling and Analysis Suite for Real-Time Applications) as a methodology for the modelling and analysis of real-time distributed systems using the transactional model, which is compatible with the analysis approach proposed by SPT [6]. A characteristic of MAST that makes it specially useful to model component-based applications is that it formulates the model in three sections: the logical part, which model the processing capacity and synchronization elements required by the software modules; the platform used, which models the processing capacity and resources that are available to the system; and the real-time situations, which describe the way the system uses the resources in each mode of operation in response to the workload imposed. This work describes at conceptual level the strategy proposed to get the composability of real-time models. The meta-modelling approach followed is an extension of the MAST methodology, called CBSE-MAST, but the concepts and solutions brought are directly usable in any methodology derived from the SPT standard [6]. Like [17], CBSE-MAST uses a behavioural/resource decomposition, but it brings an explicit concurrency model, and uses holistic analysis techniques to target distributed systems.

The paper is organized as follows, Section 2 shows the basic concepts and the metamodel that defines the set of available modelling elements. Section 3 presents as an example a simple real-time distributed application that will serve to illustrate the usage of the methodology. Section 4 describes relevant characteristics of the structure of software component descriptors. Section 5 describes the way real-time analysis models are generated for the real-time situations of an application. Section 6 presents some of the tools used to compose, analyse and design the application using the proposed methodology. Finally, Section 7 summarizes our conclusions.

2. "RT-Model_Descriptor" and "RT-Model_Instance" concepts

RT-Model_Descriptor and RT-Model_Instance are the key concepts of the real-time modelling methodology described in this work:

- An RT-Model_Descriptor is an abstract modelling entity that represents a generic and parameterized descriptor. It is used to describe the real-time model of any type of resource or service in the system. It constitutes a parameterized template that includes the semantic and quantitative information of all the aspects that are inherent to the component and affect its real-time behaviour. The information that a descriptor provides is independent of the application in which the component is used, the platform in which it is executed, and the behaviour of other components that it requires to implement its functionality.
- An RT-Model_Instance is a modelling entity that represents a concrete final model of a single instance of a component, resource or service of the system, in a partic-

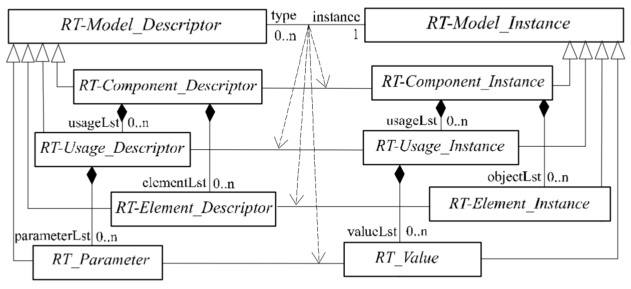

Figure 1. Core classes of the metamodel

ular application. In the model of a real-time situation, each RT-Model_Instance object is declared in reference to the corresponding RT-Model_Descriptor, which defines its nature and semantics, assigning concrete values or instances to all the parameters, attributes and references it has.

Figure 1 shows the root classes of the meta-model of the proposed real-time models. RT-Model_Descriptor and RT-Model_Instance are high level concepts used to support not only the real-time modelling of software components, and hardware/software modules of the platform (RT-Component_Descriptor and RT-Component_Instance), but also the modelling of any other basic element used to describe the internal nature of a component (RT-Element _Descriptor and RT-Element_Instance), as well as the services that it offers (RT-Usage_Descriptor and RT-Usage-_Instance). The set of concrete classes that represent the modelling contructs used to describe the temporal behaviour of all the elements that take part in the execution of an application are derived from them. A detailed description of the aspects that these elements model and the attributes that define their behaviour can be found in [12][13].

The MAST modelling methodology has defined a wide range of basic modelling primitives to model real-time applications and, as it is shown in [14], they are implementations of the entities proposed in the SPT profile [6]. These modelling primitives are classified in two groups:

- Resources models: They model the behaviour of those elements of the application that relate to the available processing capacity, either because they provide it, or reduce it, or because they modify its usage due to mutual exclusion or synchronized access. *Processors*, *Networks*, *Timers*, *Drivers*, *Schedulers*, *Scheduling Servers* and *Shared Resources* are included in this group.
- Usage models: They describe the information required for evaluating the timing behaviour of the activities executed in the application. They model the consumed processing capacity, or the resources required for execution that may generate deadlocks or delays. This group includes *Transactions*, *Jobs, Operations*, *Interfaces* and *Real-Time Situations*.

3. Application example

To illustrate the concepts that have been defined, a simple example is proposed. It is a distributed real-time application that reads a set of digital signals, generating some alarm actions like playing a sound in a speaker and setting other digital lines, when a wrong state is detected. Figure 2 shows the logical model of the application, which is based on three types of software components:

Agent: It is an active component that controls the concurrent execution of multiple alarm checking tasks, using an independent thread for each one.

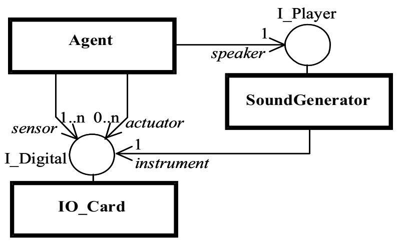

Figure 2. Software architecture of the example application

IO_Card: It is a passive component that offers the I_Digital interface, which allows clients to read an input digital line (*readState* function) or set an output digital line (*writeState* procedure).

SoundGenerator: It is a component that offers the I_Player interface, which allows clients to generate sounds (*play* procedure). It uses some digital lines to control the device that physically creates the sounds.

Figure 3 shows the deployment of the *carAlarm* application, which is the one that will be modelled. The execution platform consists of two nodes, *panelProcessor* and *engineProcessor*, both use MaRTE OS as operating system and communicate through the *localBus* CAN bus. The RT_CORBA distribution services are used as middleware in this application. The components *alarmControl* of the type Agent, *boardSpeaker* of the type SoundGenerator, and *boardPanel* of the type IO_Card, run in *panelProcessor* and the components *engineSensor* and *engineActuator*, both of the type IO_Card, are executed in *engineProcessor*.

The approach in this work proposes the elaboration of the real-time models in two phases. When the components are developed (or simply acquired) their real-time models are also elaborated and registered with their code and meta-

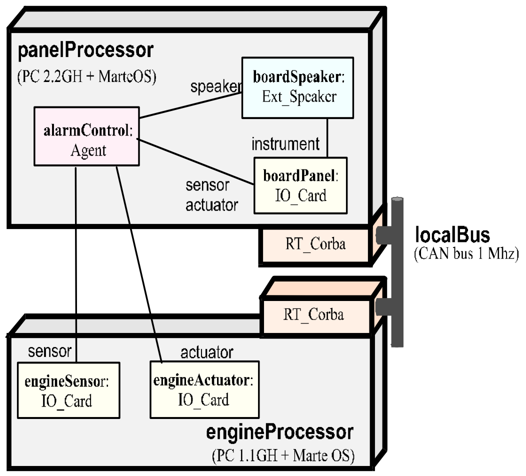

Figure 3. Platform architecture and application deployment

data. Likewise, for the hardware and software platforms that are planned to be used, their real-time models must be formulated, validated and registered. Later, when a certain application is under development, the models of its real-time situations are elaborated and composed with instance models of the software components, and the platform resources and middleware that are part of the application. Figure 4 shows the set of descriptors and instances that take part in the modelling of the *carAlarm* application, whose corresponding deployment is shown in Figure 3.

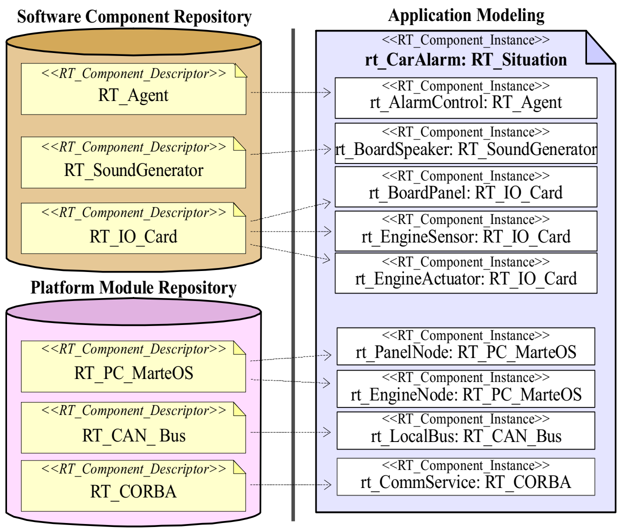
Software Component Repository
<<RT_Component_Descriptor>>
RT_Agent
<<RT_Component_Descriptor>>
RT_SoundGenerator
<<RT_Component_Descriptor>>
RT_IO_Card
Platform Module Repository
<<RT_Component_Descriptor>>
RT_PC_MarteOS
<<RT_Component_Descriptor>>
RT_CAN_Bus
<<RT_Component_Descriptor>>
RT_CORBA
Application Modeling
<<RT_Component_Instance>>
rt_CarAlarm: RT_Situation
<<RT_Component_Instance>>
rt_AlarmControl: RT_Agent
<<RT_Component_Instance>>
rt_BoardSpeaker: RT_SoundGenerator
<<RT_Component_Instance>>
rt_BoardPanel: RT_IO_Card
<<RT_Component_Instance>>
rt_EngineSensor: RT_IO_Card
<<RT_Component_Instance>>
rt_EngineActuator: RT_IO_Card
<<RT_Component_Instance>>
rt_PanelNode: RT_PC_MarteOS
<<RT_Component_Instance>>
rt_EngineNode: RT_PC_MarteOS
<<RT_Component_Instance>>
rt_LocalBus: RT_CAN_Bus
<<RT_Component_Instance>>
rt_CommService: RT_CORBA

Figure 4. RT-Component_Descriptors and RT-Component_Instances of the example

4. RT-Component_Descriptors of software components

As far as the real-time model is concerned, a software component is a reusable module of application code (a function library, an RT-CORBA server, a CBSE-Component, etc.). Its RT-Model_Descriptor contains all the information that describes the timing behaviour of its offered services, the synchronization mechanisms that it uses to manage concurrency, and in the case of active components, the models of the transactions that it may introduce.

In order to show the more relevant characteristics of a software component model, components are classified, in a non exclusive way, according to the following patterns:

- Server component: Software component whose offered services are directly implemented inside its code, and can be invoked locally or remotely by other components. In the application example, *SoundGenerator* and *IO_Card* components correspond to this pattern.
- Client component: Software component that makes use of other components services to implement its own offered services. *Agent* and *SoundGenerator* components match this pattern in the example.
- Active component: Software component that can trigger transactions, since it receives and must respond to timed or external events. *Agent* and *SoundGenerator* components match this pattern in the example.

The RT-Component_Descriptor of a component that implements a server pattern must declare the models of the services that it offers. Figure 5 shows a section of the RT-Component_Descriptor corresponding to an IO_Card server component. This section of the model describes the different timing behaviours that the *writeState* procedure may have:.

- The simple operation *localWriteState* describes its execution time (in the worst, best and average cases). Time is expressed normalized with respect to the speed of a certain processor, taken as a reference one. The physical execution time is calculated taking into account the relative speed of the processor in which the operation is executed. It also defines that for the procedure execution it is required exclusive access to the *mutex* resource, which uses the immediate ceiling protocol. The value AGGREGATED of the attribute *tie* indicates that there will be a *mutex* for each instance of the component. Its priority ceiling, *@theMutexCeiling,* is a parameter to which a concrete value must be assigned for each instance of the

```
<?xml version="1.0" encoding="UTF-8" ?>
<?mast fileType="CBSE-Mast-Component-Descriptor-File"
version="1.1"?>
<MAST_COMPONENTS
xmlns:xsi=http://www.w3.org/2001/XMLSchema
      instance xsi:schemaLocation="http://cbsemast/component
      Mast_Component_v2.xsd" fileName="RT_IO_Card" domain="ADQ"
      author="Patricia Lopez" version="2006-01-11">

  <!-- Real-time model of the logical component IO_Card.
       It controls the PCI-9111 card for digital/analog signals adquisition.
   <Component name="RT_IO_Card" tie="DECLARED"
              type="RmtSoftModule" roles=”Digital_IO Analog_IO”>
    <!-- ........................................................................................... -->
    <:ComponentRef name="host" value="@theHost"
      reqRoles="RCI_PROC_NODE" />
      ….
    <!-- ........................................................................................... -->
    <SimpleOperation name="localWriteState" wcet="2.2E+06"
      acet="2.2E+06" bcet="2.2E+06">
      <SharedResources>mutex</mast_u:SharedResources>
        </SimpleOperation>
    <APCOperation name="writeState" usage="localWriteState">
      <OutgoingMessage minMessageSize="32" maxMessageSize="32"/>
      <OutgoingMarshalling acet="8.9E-06" bcet="8.9E-06"
              wcet="8.9E-06"/>
      <OutgoingUnmarshalling acet="5.5E-06" bcet="5.5E-06"
              wcet="5.5E-06"/>
     </APCOperation>
….
    <!-- ........................................................................................... -->
    <!-- Model of common element declared in the component -->
      <ImmediateCeilingResource name="mutex"
      ceiling="@theMutexCeiling" tie="AGGREGATED" />
    </Component>
</MAST_COMPONENTS>
```

Figure 5. Section of the RT-Component_Descriptor of an IO_Card component

component. This model is sufficient when the procedure is invoked locally.

- The asynchronous procedure call (APCOperation) *writeState* extends the model of the simple operation with the information required to model the timing behaviour of a remote invocation of the procedure. It describes only the information that is consequence of the procedure nature, like the messages length (due to the size of the arguments) and the overheads introduced by the marshalling and unmarshalling operations (used to serialize the arguments to send them through the network).

@theHost is a parameter that references the model of the processor node in which the instance of the component is installed and executed, and it must be present in the real-time model of any software component. It allows to access the basic characteristics of the processor (processing speed, scheduler nature, timer resolution, etc.). Access to them is required by the tools that process the model in order to evaluate the response times of the transactions that use any of the services offered by the component.

The RT-Component_Descriptor of a component that implements a client pattern, and consequently requires other components to implement its functionality, must include parametric references to the real-time models of these required server components. These references allow the tools to access the models of the server components in order to evaluate the response times of the services offered by the client component. To consider the case in which the access to the server is remote, the descriptors include additional parameters to refer to the communication network and protocol models used in the invocation of the service.

Figure 6 shows a section of the RT-Component _Descriptor of an *Agent* component type. The parameter *@usedSpeaker* is defined to make reference to the model of

```
…
 <Component name="RT_Agent">
…
   <!-- Reference to Ext_Speaker server -->
   <ComponentRef name="speaker" value="@usedSpeaker"
            reqroles="SoundGenerator"/>
   <!-- Reference to network model for remote invocation -->
   <ComponentRef name="net" value="@theCommNetwork"/>
   <!-- Reference to communication middleware to remote invocation -->
    <Component name="speakerAccess" base="@usedSpeakerAccess">
      <Job name="rmt_speakerPlay" base="call_APC">
        <AssignedParameters>
          <UsageRef name="calledUsage" value=speaker.play"/>
          <SchedulerRef name="remoteHost" value="speaker.host.scheduler"/>
          <SchedulerRef name="commScheduler" value="net.scheduler"/>
        </AssignedParameters>
      </Job>
   </Component>
 . . .
 </Component>
</MAST_COMPONENTS>
```

Figure 6. Extract of the RT-Component_Descriptor of a client type component

```
…
<Component name="RT_Agent">
…
 <SimpleOperation name="TestAlarm" wcet="7.7E-6" acet="7.2E-6"
            bcet="7.1E-6"/>
…
 <Transaction name="ControlAlarmTask">
   <RegularSchedulingServer name="transServer" scheduler="host">
     <FixedPriorityPolicy priority="@controlAlarmPriority"
            preassigned="NO"/>
   </RegularSchedulingServer>
   <PeriodicExternalEvent  name="startControlAlarm"
            period="@alarmPeriod"/>
   <RegularEvent name="ev2"/>
   <RegularEvent name="ev3"/>
    …
   <RegularEvent name="ev7"/>
   <RegularEvent name="endControlAlarm">
     <HardGlobalDeadlineReq deadline="@alarmPeriod"
            referencedEvent="startControlAlarm"/>
   </RegularEvent>
   <Activity name="ReadValue" inputEvent="ev1" outputEvent="ev2"
            activityUsage="@theSensor.sensorAccess.rmt_readSensor"
            activityServer="transactionServer">
     <AssignedParameters>
       <Priority name="outgoingMsgPrty" value="@readOutgoingMsgPrty"/>
       <Priority name="remoteAgentPrty" value="@readRemoteAgentPrty"/>
       <Priority name="incomingMsgPrty" value="@readIncomingMsgPrty"/>
     </AssignedParameters>
   </Activity>
   <Activity name="EvaluateRes" inputEvent="ev2" outputEvent="ev3"
           activityUsage="TestAlarm" activityServer="transactionServer"/>
    ...
 </Transaction>
…
</Component>
```

Figure 7. Declaration of a transaction in a RT-Component_Descripor of an active component

the component to which the *Agent* will access with the role *speaker*. For the case in which the client and the server are in different processors, the descriptor includes parameters to refer to the communication network model (*@theCommNetwork*) and the communication middleware (*@usedSpeakerAccess*) used to invoke the service. A communication network model describes its bandwidth, the messages granularity, the characteristics of the scheduler that manages messages transmission, the processing capacity required to the processors by the drivers, and the overhead due to synchronization protocol messages among nodes. The middleware model describes the sequence of activities that represents the internal code executed between the service invocation made by the client and the execution of the method in the server. This sequence has two alternative models, one for local invocations and another for remote calls.

The RT-Component_Descriptor of a software component that corresponds to the active pattern contains parameterized models of the transactions that the component will manage. Any element of a transaction can be declared as a parameter, which includes event triggering patterns, operations, timing requirements, scheduling characteristics, etc. Figure 7 shows a section of the *Agent* component model with the description of the transaction that the component can manage. Figure 8 shows its functionality.

A transaction model defines a sequence of activities. This sequence is described using a reactive event-based model that holds its external triggering events patterns, the control flow dependencies between the activities, and the timing requirements that must be met. Each activity represents the execution of a simple or composite operation defined in any of the components declared in the model, and indicates also the concurrency unit (*SchedulingServer*) in which the operation is to be performed. In the model of the generic transaction *ControlAlarmTask*, declared in component *Agent*, the external event *startControlAlarm* describes the periodicity and trigger frequency, the events *ev2, ev3*,.. represent the control flow dependencies, and the timing requirement relative to the transaction completion is indicated in the last event *endControlAlarm*. In the segment of the transaction model shown in Figure 7, two activities are modelled: *ReadValue,* whose operation is defined in the component with the role sensor (indicated with the parameter *@theSensor.SensorAccess. rmt_readSensor*), and *EvaluateRes*, whose operation, *TestAlarm*, is declared in the component itself.

The transaction model has several parameters that must have concrete values assigned in the RT-Model_Instance declaration. For example, *@theSensor* is the parameter that references the concrete instance of the server component that is used in the transaction with the role sensor, *@alarmPeriod* determines both the invocation period and the deadline that must be met, and *@controlAlarmPriority* defines the scheduling priority of the SchedulingServer in which the operations are executed.

5. Formulation of the real-time model of an application

The first phase of modelling corresponds to identify the different real-time situations in which the application can operate. Each real-time situation represents a specific operation mode of the system, and it consists of a configuration and static deployment of component instances, for which

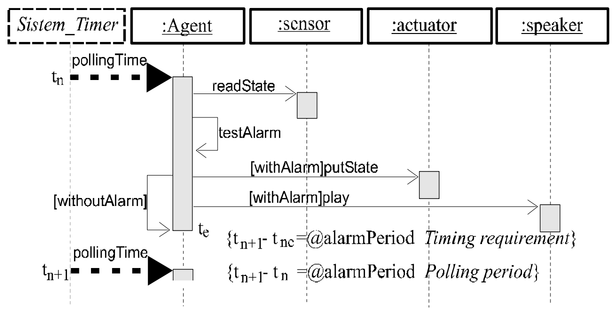

Figure 8. Alarm transaction managed by a component of the Agent type

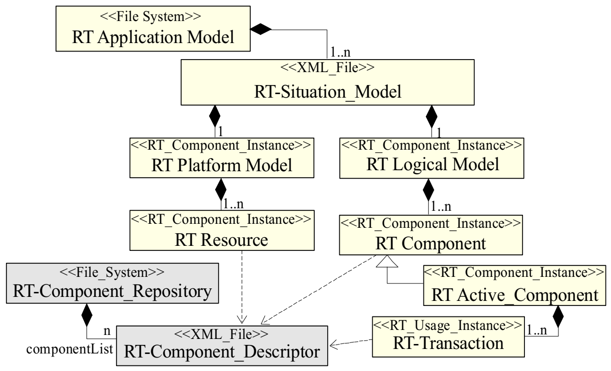

Figure 9. Elements of the real-time model of an application

real-time requirements have been defined. For each real-time situation a complete and independent real-time analysis model is generated, and schedulability analysis or response time estimation tools are applied to these final models.

Formulating the real-time model of a real-time situation implies two tasks, the first has an structural nature and involves collecting and linking all the RT-Model_Instances that describe the behaviour of the software and hardware elements that take part in the application execution. The second comprises modelling the real-time situation from the reactive point of view, for which it is necessary to define the application workload and the timing requirements that the application must meet.

From the structural point of view, the organization of instances in a real-time model follows the structure that the application has, regarding the components of which it is build up. Figure 9 shows the elements that take part in the real-time model of an application:

- For each hardware or software resource of the execution platform, the corresponding RT-Model_Instance must be instantiated, which includes the models of communication nodes (Processors, timers, schedulers), communication networks (networks, schedulers, drivers) and middleware elements (remote access resources, brokers, etc.). These instances form the RT_Platform_Model.
- For each component that takes part in the application, the RT-Model_Instance that describes its temporal behaviour is instantiated. This set of instances form the RT_Logical_Model.
- All the links that each instance requires to get access to the other instances on which its model depends, must be established according to the configuration and deployment of the application.

The process of instantiating the RT-Model_Instance of a software component or a platform resource consists of taking the reference to its corresponding RT-Model_Descriptor, and considering the application context, assigning concrete values to its parameters.

The workload in the real-time model of an application is determined by the set of transactions that its active components declare. The activation patterns that trigger them may be deterministic or defined through the statistical distribution of the interarrival times between consecutive triggering events. They also provide references for defining the global timing requirements.

There are two types of transaction instances. On the one hand, those defined as <<Aggregated>> are inherent to the model of the component and are automatically attached to its instance model, so the designer do not have to declare them explicitly in the real-time situation model. On the other hand, those transactions that depend on the application workload, as a consequence of the input data or the application configuration, are defined as <<Declared>> in the active component models. The number of instances of these transactions present in the system and the concrete values assigned to their parameters must be explicitly established by the designer in the real-time situation model.

6. Analysis and design tools

The real-time modelling methodology proposed is reusability and composability oriented, and the internal organization of the models reflects the structure of the modules which the application has been designed with. In order to structure the timing information of the system in the way that is needed for the analysis and design of the application, these models have to be transformed to a purely transactional formulation, which is compatible with the techniques used for schedulability analysis. As it is shown in Figure 10, the tool *RT_Model_Component_Compiler* has been developed to generate this transactional model of the application. It takes as inputs the RT-Model_Instance that represents a concrete and complete model of the application in a particular real-time situation, and all the RT-Model_Descriptors that are referenced in it, which are stored in the component repository. All this models are formulated as text files with XML tagged formats that follow the meta-models defined and their corresponding Schemas.

The transactional model generated by this tool follows the MAST methodology. The modelling resources currently defined in MAST are able to model most of the real-time programming features included in real-time operating systems and languages, like POSIX.13 and the real-time and distributed Annexes of Ada95.

Figure 9 shows some of the tools included in the MAST environment that help in the development of real-time applications. This set of tools allow the designer to optimize the values of the scheduling parameters in order to adapt the application to the platform characteristics, and evaluate its schedulability:

- **Tools for automatic priority assignment**: In monoprocessor systems to assign priorities to the different threads used in the transactions, the tool makes the assignment using Deadline Monotonic technique[15] or some of its extensions. In multiprocessor and distributed system, priority assignment is more complex because of the strong interconnection among the response times of different resources, and because the number of priorities that must be decided is very high. In this case, it is necessary not only to assign a priority to each concurrency unit in the application, but for each remote invocation between components of different nodes three additional priorities has to be defined: one for the message that makes de invocation, one for the remote process that executes it and another for the message that returns the results of the invocation. For multiprocessor systems the tool uses the HOPA algorithm, which is an optimization algorithm based on the distribution of the end-to-end or global deadlines of each transaction among the different actions that compose that transaction [16]
- **Tools for schedulability analysis**: These tools allow to verify if the selected platform has capacity enough to execute de system in a way that all the activities scheduled in the application meet their real-time requirements. The tools can be applied to both monoprocessor and distributed systems, and using different scheduling policies, like fixed priority, EDF, or a combination of both.
- **Tools for slack calculation**: These tools allow to calculate the percentage by which the execution time of the

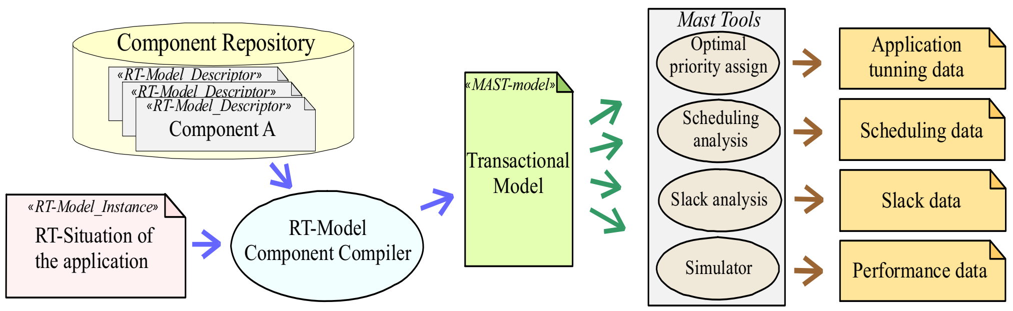

Figure 10. Analysis and design tools

operations may be increased while keeping the system schedulable, or that in which they must be decreased to get the system schedulable.

- **Tools for statistical estimation of performance**: These tools evaluate by simulation the average performance of the system and are particularly useful in applications that combine soft and hard real-time requirements.

7. Conclusions and Future Work

This work describes a methodology to formulate the real-time model of an application, which has 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.

This paper introduces the RT-Model_Descriptor concept as a general real-time behaviour model for reusable software components, like subsystems, commercial off-the-shelf software or even a hardware processing device, an operating system platform or a network. It does it as an independent entity, in a way that allows the automated construction of the final analyzable model by a tool with plasticity and composability properties similar to those used in the construction of the application.

The key in this methodology is the application of both concepts, descriptor and instance, in the real-time modelling of components. The RT-Component_Descriptor is a parameterized consistent form, which holds the complete real-time data of a component disregarding the applications in which it may be used. It includes all the internal elements of the model, and leaves the "blanks" to be filled with references to the concrete models of components that affect its behaviour. The RT-Component_Instance is the final real-time model of a component instance in the context of a concrete application. It is to be built by a tool after all the necessary components in the real-time situation are known.

The methodology has been successfully applied in the real-time modelling of a set of components designed for the development of industrial controllers and the automation of production lines using real-time POSIX OS, CAN Bus, real-time protocols on Ethernet, Ada95 and RT_CORBA software technologies.

Further work will explore the future MARTE [18] UML 2.0 based profile, for more precise constructs to hold and treat real-time models with standard UML CASE tools.

References

[1] H. Kopetz, R. Zainlinger, G. Fohler, H. Kantz, P. Puschner, and W. Schutz, ”The design of real-time systems: from specification to implementation and verification” Software Engineering Journal, Vol.6 , no. 3, pp. 72-82. May, 1991.

[2] J. Liu, Real-Time Systems. ISBN 0-13-099651-3. Prentice Hall Inc., 2000.

[3] M. Klein, T. Ralya, B. Pollak, R. Obenza, and M. González Harbour, A Practitioner's Handbook for Real-Time Systems Analysis, Kluwer Academic Pub., 1993.

[4] A. Cheng, “Real-time Systems Scheduling, Analysis and Verification”. ISBN 0-471-18406-3. J. Wiley & Sons, 2002

[5] IEEE Std 1003.13^{TM}-2003: ”IEEE Standard for Information Technology-Standarized Application Environment Profile (AEP)-POSIX^{R} Realtime and Embedded Application Support”, 2003.

[6] Object Management Group: "UML Profile for Schedulability, Performance and Time Specification", Version 1.1. OMG document formal/05-01-02, January, 2005.

[7] A. Burns and A.Wellings. "HRT-HOOD: a Structured Design Method for Hard Real-Time Ada Systems", volume 3 of Real-Time Safety Critical Systems. Elsevier, 1995.

[8] H. Gomaa. Designing Concurrent, Distributed, and Real-Time Applications with UML. Addison-Wesley, 2000

[9] H.Kopetz and N.Suri:“Compositional design of RT-Systems: A Conceptual basis for specification of linking interfaces” 6th IEEE Int. Symp. of Object-oriented Real-time Distributed Computing (ISORT03). Hakodate (Japan). May, 2003.

[10]A. Tesanovic, D. Nyström, J. Hansson, and C. Norström: "Aspects and Components in Real-Time System Development: Towards Reconfigurable and Reusable Software". Journal of Embedded Computing, February, 2004.

[11]M. González Harbour, J.J. Gutiérrez, J.C.Palencia and J.M.Drake: "MAST: Modeling and Analysis Suite for Real-Time Applications" Proc. of the Euromicro Conference on Real-Time Systems, June 2001.

[12].J.L. Medina, M.González Harbour, J.M. Drake: "MAST Real-time View: A Graphic UML Tool for Modeling Object_Oriented Real_Time Systems", RTSS, Dec, 2001.

[13]MAST: Modeling and Analysis Suite for Real-Time Applications. “http://mast.unican.es

[14]J.L. Medina: ”Metodología y herramientas UML para el modelado y análisis de sistemas de tiempo real orientados a objetos”. PhD Thesis (In Spanish), Santander (Spain), 2005.

[15]C.L. Liu and J.W. Laylan: “Scheduling Algorithms for Multiprogramming in a Hard Real-time Enviroment” J. Of the ACM, Vol. 29, No 1,pp46-61, 1973.

[16]JJ. Gutiérrez and M. González Harbour: “Optimized Priority Assignment for Task and Message in Distributed Real-Time Systems”. Proceedings of the 3rd Workshop on Parallel and Distributed Real-time Systems, Santa Barbara (USA), 1995.

[17]E. Bondarev, P. de With, M.Chaudron, and J. Muskens: “Modelling of Input-Parameter Dependency for Performance Predictions of Component-Based Embeded Systems”. Proc. of Euromicro SEAA, IEEE Computer Society Press 2005.

[18]Object Management Group: UML Profile for Modeling and Analysis of Real-Time and Embedded systems (MARTE), RFP. 2005. OMG document: realtime/05-02-06