© British Computer Society 2003

An Object-oriented Methodology for Embedded Real-time Systems

J. M. ALVAREZ, M. DIAZ, L. LLOPIS, E. PIMENTEL AND J. M. TROYA

Departamento Lenguajes y Ciencias de la Computacion, Complejo Politecnico, University of Malaga, 29071 Spain
Email: alvarezp@lcc.uma.es

The usage of object-oriented methodologies in conjunction with formal description techniques has arisen as a promising way of dealing with the increasing complexity of embedded real-time systems. These methodologies are currently well supported by a set of tools that allow the specification, simulation and validation of the functional aspects of these systems. However, most of these methodologies do not take into account non-functional aspects such as hardware interaction and real-time constraints, which are especially important in the context of this kind of system. Based on our experiences in developing embedded real-time systems, we present a new methodology to design them. This methodology is based on a combination of ideas from different existing methodologies (UML, OCTOPUS, etc.) together with the integration of rate-monotonic analysis in the context of the SDL formal description technique development cycle. Additionally, in order to get this integration, a real-time execution model for SDL is presented to allow us to express hard real-time constraints. The methodology pays special attention to the transition from the object model to the task model, taking into account real-time and hardware integration issues. We also illustrate our proposal by applying it to the development of a multi-handset cordless telephone.

Received 9 January 2001; revised 16 October 2002

Downloaded from http://comjnl.oxfordjournals.org/ at Pennsylvania State University on September 15, 2016

1. INTRODUCTION

Nowadays, real-time embedded computers or controllers can be found everywhere, both in very simple devices used in everyday life, such as microwave ovens, wristwatches or washing machines and in professional environments, such as medical life support systems, aircraft controllers or nuclear power plant controllers. These real-time embedded systems are spreading to more and more new fields and their scope, complexity and criticality have grown dramatically in the last few years.

As well as the functional requirements, real-time embedded systems have to meet some non-functional requirements that make them more complex than other systems. Real-time embedded systems have to take into account robustness, safety and timeliness. Most of these systems must respond in a certain period of time. A lost deadline can mean a wrong computation and a system failure. Another feature that makes real-time embedded systems different from general-purpose systems is that they have to interact directly with hardware devices, such as sensors and actuators. Besides that, the environment where these systems work is highly unpredictable, and there is not an *a priori* knowledge about when events are happening, since they can be periodic or sporadic with very different workloads.

Though the simplest systems can be developed by a single person with basic programming skills, the increasing complexity of the requirements of real-time embedded systems are making them difficult to manage with the traditional methodologies used by real-time developers. The joint use of object-oriented technologies and formal description techniques (FDTs) has been proposed as a promising alternative for the development of this kind of system [1]. Object-oriented methodologies are widely used to cope with complexity in any kind of system, but most of them lack a formal foundation that allows for the analysis and verification of designs, which is one of main requirements to deal with the complexity of concurrent and reactive systems. Also, one of the most important advantages is the improved consistency between the models of the different phases. On the other hand, FDTs provide the basis for an automated design process, allowing simulation, validation and automatic code generation from the specifications. These formal techniques were originally developed for the design of telecommunication systems and, for this reason, they were designed to cope with the characteristics of this kind of systems (concurrency, reactivity, etc.) that are common to embedded real-time systems.

One of the most widely used FDTs is SDL (Specification and Description Language) [2]. SDL is an ITU standard and is currently well supported by commercial tools like TAU [3]. SDL is based on extended finite-state communicating machines and can be used throughout the development cycle, from specification to implementation, although it is better suited for design purposes. Current object-oriented methodologies based on SDL, such as SOMT [3], use UML [4] in the requirements phase, jointly with some other formalisms to express external dynamic

behaviour, like message sequence charts (MSCs) [5]. MSCs are another ITU standard that can be used in combination with SDL to simulate and verify the system design. A serious problem of these methodologies, as in other object-oriented methodologies not based on SDL, is the gap between the object model obtained from the requirement analysis and the final SDL process model. This step cannot be automated and its success depends heavily on the experience of the designer. Another relevant drawback is the absence of real-time analysis.

Rate-monotonic analysis (RMA) [6, 7, 8] provides a collection of quantitative methods that enables us to analyse and predict the timing behaviour of real-time systems. This analysis can help us to organize processes and resources in our design to make it possible to predict the timing behaviour of the final system. In this sense, RMA will help us to incorporate real-time analysis [9, 10, 11] and save the gap between the object model and the task model.

In this paper we present a new methodology based on the integration of RMA in the context of the SDL development cycle [12] for the design of single processor systems. In order to achieve this integration, we have defined a new predictable execution model for SDL. The semantics of SDL includes non-determinism, like unpredictable ordering of messages or unpredictable process activation and cannot be directly used to model real-time behaviour. In addition, interactions with the system (including hardware devices) are always asynchronous and are not included in the design model. Our new model allows the specification of hard real-time constraints and interaction with hardware devices is also included as an important model feature. Our methodology based on SDL has an important advantage: SDL has formal semantics clearly established and there exists the possibility of using existing SDL-based simulation and validation tools. These tools allow us to detect some functional and concurrency errors in the early phases of the design process. However, the modification of the SDL execution model can affect some of the properties checked by these tools that have to be modified to take the real-time execution model into account. This is especially important in the case of model checking tools, since the transitions to be explored are highly reduced due to the elimination of non-determinism.

The rest of the paper is organized as follows. In Sections 1.1–1.3 we survey some related work, a background on object-oriented methodologies for real-time systems and we give a short description of SDL. In Section 2 we present the real-time execution model for SDL and the integration of the real-time analysis. The methodology based on this model is presented in Section 3. An application of the defined methodology to the development of a cordless telephone system is presented in Section 4, including the schedulability analysis of the system. Finally, some conclusions and future work are discussed in Section 5.

1.1. Object-oriented methodologies

Object-oriented methodologies have some important advantages in developing large-scale complex systems. They allow us to model the real system in a more realistic way, the hierarchical definition of objects lets the developer split the system into more simple and manageable pieces and concurrency is a fully and naturally integrated aspect. However, these methodologies were formerly thought of for general aim systems and hence they do not consider some relevant aspects in the development of real-time systems. For example, they do not take into account whether they have integrated the capability of schedulability analysis to know the response time of the system and if they have a formal foundation that allows us to analyse and to validate the modelled systems.

Hard real-time hierarchical object-oriented design (HRT-HOOD) [13] is an extension of HOOD that includes particular abstractions to real-time systems. It tries to incorporate the temporal requirements as soon as possible in the design. It is based on results from scheduling theory and the programming language Ada95. The schedulability aspects are seriously taken into account but other aspects of the design are left out. It is not a formal method except in schedulability analysis.

Octopus [14] is another methodology without a formal basis that tries to incorporate specific aspects of real-time systems such as reactivity, concurrency and event handling. Concurrency is considered as an inherent object attribute. During the analysis phase it is implicit and becomes more and more explicit in the following phases. Octopus divides the development of a system into four phases: analysis requirements, architecture design, subsystem design and performance analysis. The dynamic model is represented by a set of threads with interaction among objects. Groups of objects are derived from these threads and the processes and messages interchanged among them are derived from the group of objects. When the processes have been defined the performance analysis can be done. The methodology does not specify how to assign priorities nor the schedulability schema, though some schedulability theory could be integrated. However, it does not seem easy to carry out other analysis or automated code generation.

UML-RT [15] is another real-time object-oriented methodology that is becoming very popular in the real-time community. It has the advantage of the success of UML as a general-purpose modelling language. Basically, UML comes from three different object-oriented methodologies: Booch, OMT and OOSE. UML is a third-generation modelling language that tries to define the semantics of the object model and provides notation to capture the structure and behaviour of the system. UML is a graphical notation for object-oriented analysis and design. It is used for specifying, visualizing and documenting the phases of an object-oriented system. UML provides a number of graphical diagrams that provide different perspectives of the system under development.

UML-RT has adopted ideas from the ROOM methodology [16] to represent a special kind of active object with a collection of well-formed rules that specify particular semantics. These objects are called capsules. Capsules interact with each other through a message-passing mechanism

Downloaded from http://comjnl.oxfordjournals.org/ at Pennsylvania State University on September 15, 2016

known as ports. Each port has an associated protocol that defines the valid flow of information among connected ports. Unfortunately, most of the UML-RT tools do not incorporate validation and there is a limited number of automatic tools supporting ROOM.

There are a number of commercial tools that implement these methodologies, or support a part of them. Real-Time Studio [17] from Artisan Software and Rhapsody [18] from i-Logix offers the whole UML life cycle, from requirement analysis to code generation, from object specification. However they do not offer tools for the analysis of hard real-time properties, such as scheduling, or for system validation. Rational Rose [19], from Rational Software, is also based on UML. It has implemented a real-time design pattern called the Active Object, to better specify objects that fulfill the real-time profile. The Active Object solution has come from collaboration with ObjecTime Limited. ObjecTime has its own tool, ObjecTime Developer, based on the ROOM methodology. These tools also lack real-time analysis or a formal foundation for verification.

There are other tools based on UML and SDL. For example, Tau, from Telelogic, uses UML in the object design and MSCs to express the dynamic behaviour of the system and SDL for the design phase. It includes automatic mapping between UML statecharts and SDL diagrams. It offers automatic code generation and simulation and validation tools. Validation allows us to verify dynamic behaviour specified by means of MSCs, detecting possible deadlocks, livelocks, etc. The simulation tools allow us to easily reproduce the behaviour that leads to this situation.

1.2. Related work

We present some related work paying attention to the integration of real-time analysis in SDL and the specific aspects that distinguish our proposal with respect to the main methodologies for the design of real-time systems.

Some work is being carried out to try to integrate this kind of analysis in an SDL model. For example, ObjectGeode [20] includes a performance analyser in its simulator that makes use of some directives extending SDL to denote process priorities or timing delays. In [21] an earliest deadline semantic is given for SDL, allowing a mapping from a SDL specification to an analysable task network. However, they do not integrate this analysis in the rest of the development cycle, nor take into account hardware interaction or real-time anomalies such as priority inversion. Another working line in this context is the one of supplementing SDL with load and machine models, such as those described in [22], that use queuing theory to calculate job and message queuing times and processor peak and average workloads. In [23] a new approach for early performance prediction based on MSC specified systems in the context of SDL is presented. We think that these works can be complementary and useful in the first phases of design, but a final schedulability analysis has to be done for real-time systems and, in order to achieve this, it is necessary to provide SDL with a predictable execution model. In [24] and [25] we can see how real-time scheduling theory may be applied to ROOM. These works have been very useful for the proposal that we present. However, ROOM is not a formal description technique and it cannot be used to include an automatic validation phase directly from the design. We think that the validation is an important issue for the design of real-time systems. In [26] an approach towards automatic synthesis of implementations from real-time object-oriented design models is presented. This work is very interesting and it includes the issue of the real-time and timeliness requirements in the design process. Also, [27] completes the previous work and shows how schedulability analysis can be integrated with object-oriented design.

How to include the timing behaviour of the communication medium for the protocols for medium access in SDL is addressed in [28]. The authors propose a design pattern to allow the specification of time critical functionality such as multiplexers or quality-of-service (QoS) schedulers. Also, [29] presents another extension on SDL to describe non-functional requirements.

Finally, [30] studies a methodology to construct scheduled systems restricting the behaviour of the processes to guarantee two types of constraints: schedulability constraints and constraints on scheduling algorithms such as process priorities and preemption. Although the methodology is illustrated on periodic processes, it can be applied to arbitrary systems. Another contribution of this work is the decomposition of scheduling requirements into classes of requirements that can be expressed as safety constraints.

With respect to the related work on the mapping from an object model to a process model, UML/RT proposes for each active object of the object model a dynamic view with state charts. We propose to pass from the object notion in the analysis to the process notion in SDL. SDL is a formalism with standard semantics that has tools for validation that are very important in the design of real-time systems. In this paper we do not deal with the implementation of the SDL process on the operating system but it can be seen in [31]. Octopus [14] proposes the design model as an extension of the object model although it can need a further decomposition in the implementation phase. In order to design the concurrency it proposes the concept of an *object interaction thread* that represents the objects that take part in an event. Octopus uses state charts to describe the behaviour in terms of state changes caused by events. However, they do not use them formally. It uses *class outlines* to record the details of the design. It is the start point of the implementation phase. Octopus combines two concepts, *object* and *operating system process*, to design the concurrency. The main difference with our proposal is that we include, between these two concepts, the concept of the SDL process. The mapping from the object to operating system process is not easy and our proposal makes this mapping easier. Additionally, there are tools to generate code from SDL.

The mapping from the analysis model to the design model in ROPES [15] can be done using an elaborative or translative approach. The translative approach proposes

Downloaded from http://comjnl.oxfordjournals.org/ at Pennsylvania State University on September 15, 2016

to define translators applied against the object model to produce a executable system. The translator has two parts: a real-time framework and a code generator. In this case the object model needs to be more elaborate than we propose in our methodology. The elaborative method is more traditional. The analysis model is elaborated with design detail and it can be maintained as a distinct entity from the analysis model. This coincides with our proposal.

HRT-HOOD maps the object model to ADA. For each object, two packages are generated: the first simply contains a collection of data types and variables defining the object's real time attributes; the second contains the code for the object itself. In this sense this methodology proposes a translative approach.

These different ways of mapping the object model to the design model are very interesting, but we think that it is important to include in the design model a formalism to include a validation phase to detect important situations in the design of real-time systems as starvations, deadlocks, etc.

1.3. SDL

SDL is an object-oriented, graphical specification language intended for the description of the structure, behaviour and data of distributed communicating systems. As SDL is a language with a formal foundation, every symbol has precise semantics associated with it; and then it eliminates any ambiguity and guarantees system integrity.

The new SDL semantics are based on ASMs. Finite versions of this can be seen as finite machines extended with variables and timers—they run in parallel. Going from one state to another is called a transition. A transition between two states is only made after a stimulus has been received. These machines are independent in the sense that every process has its own data space and there are no shared data. The main communication mechanism is asynchronous message passing. An event is a stimulus (signal sending or time expiration) that activates a sequence of transitions to respond to it.

In SDL, systems are described hierarchically in different levels of detail: system, blocks, processes and procedures. The system description is refined progressively in the successive levels. A system is decomposed into blocks that communicate with each other by means of channels. This is the static view of the system. The dynamic view of the system is described by means of processes and procedures. Each block is split into a set of processes that run in parallel. All data have to be local to processes; that is, there are no shared data to increase the robustness of the system and to reduce deficiencies.

Processes are state machines that react to external stimuli. These stimuli can come from the environment or from other processes. Processes can communicate with each other by asynchronous (called signals) or synchronous mechanisms (remote procedure calls). Both mechanisms can carry parameters to exchange information between processes and with the environment (see Figure 1). Timers are the SDL

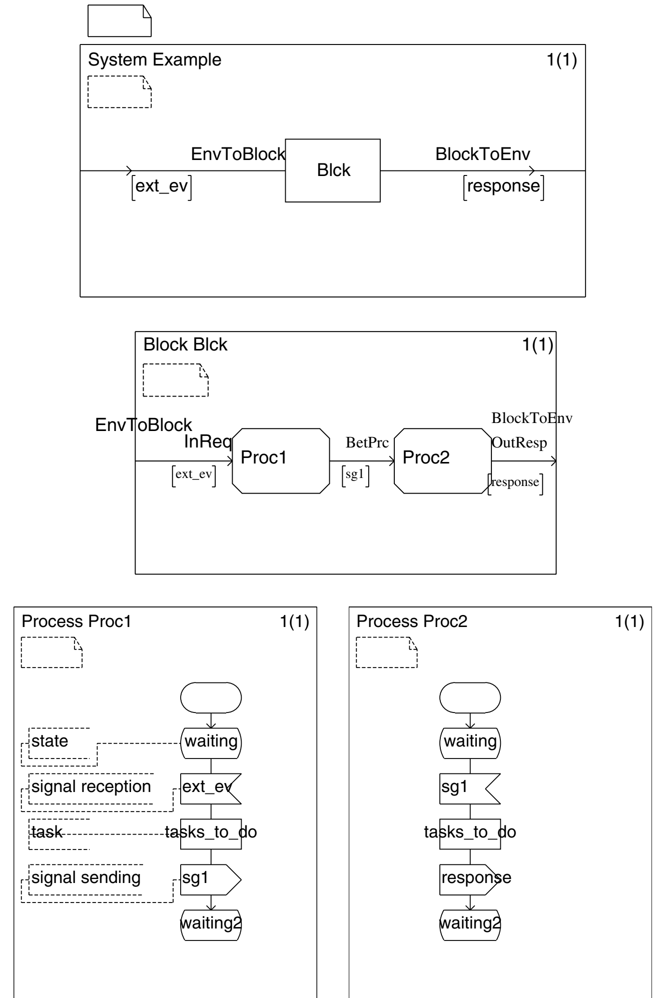


FIGURE 1. SDL blocks and processes.

instruments used to deal with time. Timers refer to a global, abstract notion of time. Timers can be set and reset and expire. When a timer expires, a signal is sent to the process that sets the timer. This timeout signal is received and managed by the process as any other signal. Timers allow us to simulate time passing before target implementation and can be mapped to operating system timers or hardware timers.

SDL in not concerned with the priority assignment to processes or signals. The designer has to assign them in the implementation phase.

2. A REAL-TIME MODEL FOR SDL

As commented on in the introduction, we define a new object-oriented methodology using SDL in the design phase. However, we have detected some real-time anomalies in SDL that have to be solved if we want to incorporate real-time analysis. In this section we present some real-time anomalies that the SDL standard model introduces and we propose some solutions so as to be able to express real-time constraints in SDL. Finally, we define a new and predictable execution model for SDL. With this model we try

Downloaded from http://comjnl.oxfordjournals.org/ at Pennsylvania State University on September 15, 2016

to eliminate non-determinism from the SDL semantics and make possible the real-time analysis of the implementations derived directly from a SDL design. This implementation will be automatically generated and will run on the final target either with an autonomous SDL run-time library (without operating system support, such as the SDT Cmicro library [3]) or a middleware that implements the execution model on a real-time operating system. In both cases a predictable SDL execution model will be necessary if we want to be able to analyse real-time behaviour.

When defining an execution model for embedded real-time systems we have to take into account at least the following aspects.

- Process scheduling. In real-time systems processes have to be scheduled in the appropriate way in order to meet real-time constraints. In our model we use preemptive scheduling with fixed priorities that is the base of RMA.
- Process communication mechanisms. SDL uses asynchronous message passing as the basic mechanism for process communication, although in SDL-92 remote procedure call has been integrated as an alternative communication mechanism.
- Hardware interaction model. In SDL hardware interaction is not considered explicitly. All the interactions with the system environment have to be achieved by means of asynchronous signal interchange.

In addition, we need to be able to express real-time constraints in the language context (SDL). Although SDL has mechanisms to express real-time constraints (timers) it is necessary to analyse if they are powerful enough and if they can be easily managed in the context of the execution model. Our objective is to introduce as few modifications as possible to the language and to maintain the semantics of timers as close as possible to the standard semantics.

In the rest of this section we first analyse the problems in expressing real-time constraints and the real-time anomalies that the SDL standard execution model introduces, then, we describe our proposal and analysis method.

2.1. Expressing real-time constraints

In SDL there are two main mechanisms to express real-time constraints.

- A global clock that can be accessed by means of the `now` function. This function returns a `TIME`-type value that represents the time in seconds from system initialization.
- Timers, which are special objects that support two operations.
  - `SET(time, Timer)`—initializes a timer that will expire in time seconds. When timer expires, a signal Timer is sent to the process that owns the timer.
  - `RESET(Timer)`—resets the timer.

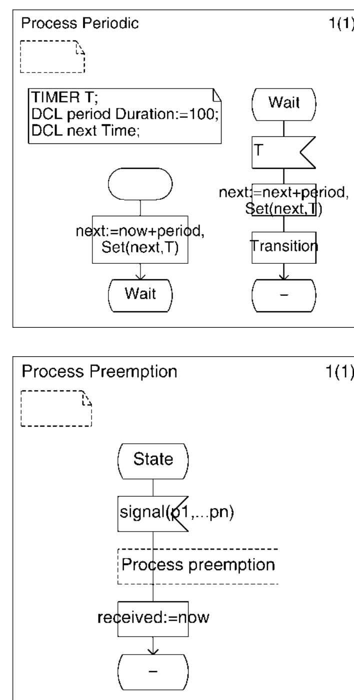


FIGURE 2. Real-time requirements in SDL.

In Figure 2 we show an example of how a periodic process can be specified with these operations. The expressiveness of these constructions can be enough, but it depends on the underlying execution model. For example, with the standard FIFO semantics of signal reception, time-out signals are received after all the signals that were in the process queue when the timer expired. With these semantics it is impossible to activate a process exactly when the timer expires and henceforth it is impossible, for example, to specify a periodic process or an exact delay.

Another problem with the SDL time semantics is the difficulty of expressing real-time constraints involving the sending and reception of signals. There is no way of knowing when a signal was really sent or attended. For example, in Figure 2, if we want to know when a signal was attended we cannot use the `now` function. Since process scheduling can be preemptive, a lot of time could have elapsed between the signal reception and the invocation of the `now` function.

In order to avoid these difficulties we have included two slight but relevant extensions to SDL.

- Process and transition priorities. We will assign priorities to process transitions, depending on the events that are involved and following RMA criteria. Process priorities are calculated taking into account the signals in the process queue and the possible transition from the current state (see Section 2.3). Processes can have a default priority that is specified in the process declaration. In Figure 3 we show how these priorities can be specified.
- Signal time-stamps. Every signal instance will have two time-stamps: one recording when the signal was

Downloaded from http://comjnl.oxfordjournals.org/ at Pennsylvania State University on September 15, 2016

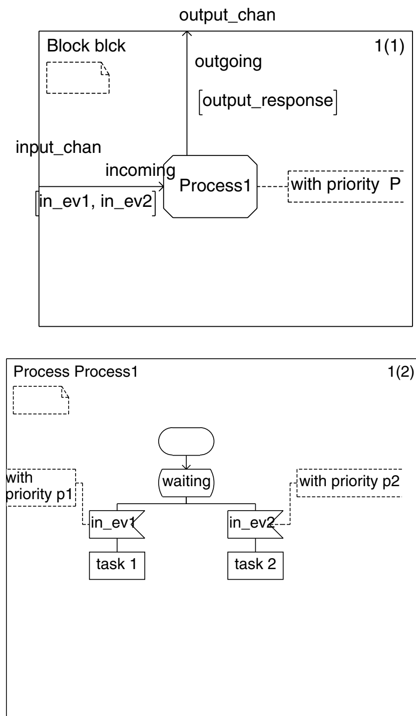


FIGURE 3. Priority specification.

sent and another one recording when the signal was attended. These time-stamps can be accessed by means of two predefined functions: `time_sent` and `time_attend`. Both functions are referred to the signal that activated the transition. With these time-stamps it is easy to specify real-time constraints relative to signal sending and reception, as input or output jitter.

2.2. Real-time limitations

In addition to real-time expressiveness, other SDL characteristics have to be analysed in order to define an analysable execution model. Some of these characteristics are determined by the underlying communication mechanism: asynchronous message passing. In this section we will analyse some real-time limitations that have to be overcome in order to make the SDL execution model predictable.

These limitations are presented in the context of the TAU design tool, which defines different priority based execution models.

2.2.1. Priority inversion

Priority inversion occurs when a higher-priority process is forced to wait to obtain a resource that is being used by a lower-priority process. This situation is unavoidable, but it is necessary to limit the delay and to make it as short as possible. The main source of priority inversion is data or resource sharing. In SDL, data cannot be directly shared among processes and they have to be encapsulated in a server process.

In Figure 4, we can see a typical priority inversion scenario. In this example we will suppose that all the transitions in a process have the same priority. Let us suppose the `client` process with high priority has to access a low-priority server (Process `server`). This server can be interrupted by another medium priority process (Process `OtherProc`) that does not need to access this server. Priority inheritance protocols have to be used in order to avoid these situations.

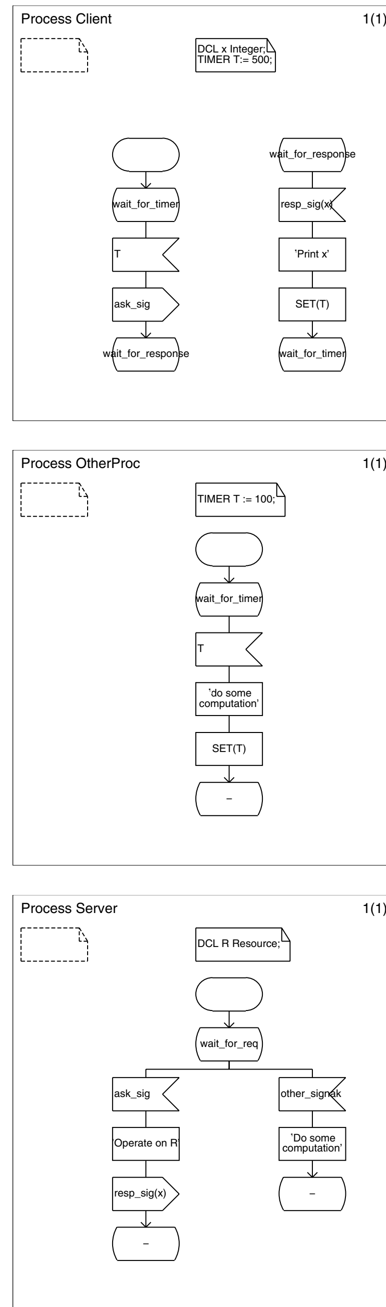


FIGURE 4. Priority inversion example.

Priority inversion can also occur when different clients with different priorities try to access the same server at the same time. In this case, if the standard FIFO semantics are maintained, the process with higher priority can be starved. In the next section we show how these problems are solved in our model.

Downloaded from http://comjnl.oxfordjournals.org/ at Pennsylvania State University on September 15, 2016

2.2.2. Precedence constraints

In message-based systems, the treatment of any event may involve different related processes in some precedence order. For example, consider two simple external events ($Senv_1$ and $Senv_2$) that are processed by a chain of three processes $Process_H$, $Process_M$ and $Process_L$. In Figure 5 we show the definition of the system and the three processes, respectively.

The external event $Senv_1$ is initially processed by $Process_H$ that sends a signal to $Process_M$ that, in turn, sends another signal to $Process_L$. The other event is processed in a similar way, but the precedence relation is the reverse one. Let us suppose that we assign priorities to processes and that priority($Process_H$) > priority($Process_M$) > priority($Process_L$). When $Senv_1$ is processed the behaviour of the processes is right, but when $Senv_2$ is received its processing can be starved, since $Process_M$ and $Process_H$ could also be involved in the processing of other events.

This problem occurs due to assigning fixed priorities to processes. If a process can receive signals caused by events with different timing requirements, its priority cannot remain the same when the different signals are consumed.

2.2.3. Hardware interaction

In the SDL model, all interactions with the environment are achieved by means of signals to and from the environment. For this reason hardware drivers have to be represented by an active process independent of the SDL design. In embedded systems, the hardware interaction is an important part of the system and should be considered in the design. In addition, if we want to obtain an analysable model, hardware timing requirements and the driver's timing behaviour have to be considered.

2.3. An analysable execution model

The execution model is based on fixed-priority preemptive scheduling; however, we do not assign fixed priorities directly to processes but to process transitions. Process priorities can vary from one state to another depending on the transitions that it can carry out in the current state (taking into account the queued signals). Processes are scheduled according to these dynamic priorities, although the schedulability analysis is based on the transition priorities, which are fixed. Transitions can be preempted by higher-priority ready transitions of other processes, but never by a transition of the same process, i.e. if a process transition with higher priority becomes ready while another transition of the same process is executing, that transition is delayed until the current one has finished. This may cause an increment of the response time of events, but this constraint is necessary in order to maintain SDL process execution semantics. Assuming this, processes are preemptively scheduled according to their dynamic priority.

In order to show how process priorities are calculated we first define some technical issues. We will denote the sets of system processes, process states and signals by *Process*, *State* and *Signal*, respectively.

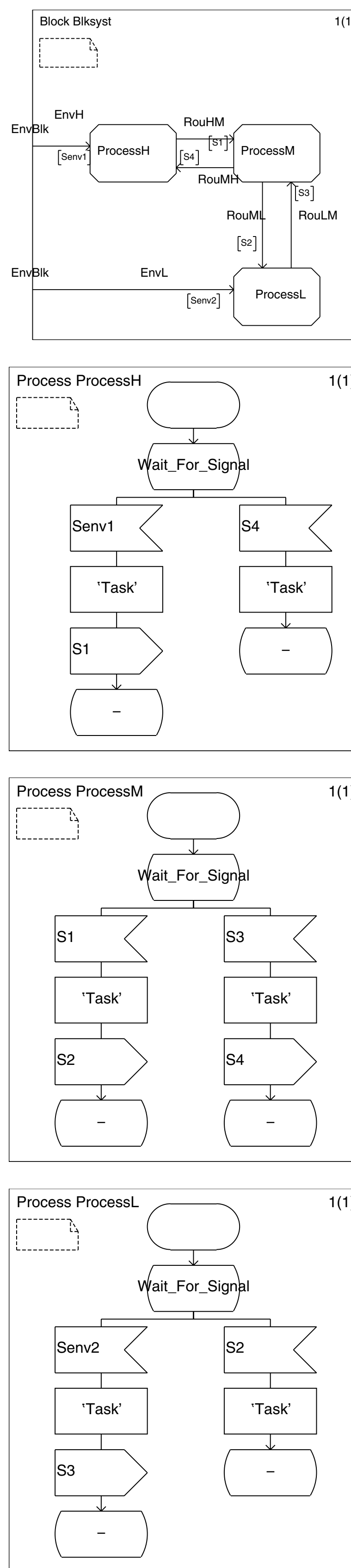


FIGURE 5. Process precedence example.

Downloaded from http://comjnl.oxfordjournals.org/ at Pennsylvania State University on September 15, 2016

The function *sig*, defined as

$$sig : Process \times State \rightarrow \wp(Signal \times N)$$

returns, for each state *st* of a process *P*, the set of pairs of the form (sg, pr), where *sg* is a signal that can be received by the process *P* in the state *st* and *pr* is the priority assigned to the transition associated to that signal. The definition of the mapping *sig* is independent of the system configuration; in other words, its definition can only be fixed by observing the specification of each process.

Additionally, the function

$$nextst : Process \times State \times Signal \rightarrow \wp(State)$$

is defined as follows. Given a process *P*, for each state *st* and a signal $sg \in sig(P, st)$, $nextst(P, st, sg)$ returns the set of next states that could be reached by the transition consuming *sg*. It is worth noting that the definition of function *nextst* can also be statically fixed.

The specified system is travelling through different configurations. Each configuration can be identified with a function mapping processes to queues of signals. We will denote by *Config* the set of configurations. Thus, for each configuration *C*, we consider a mapping defined as follows:

$$received : Config \times Process \longrightarrow \wp(Signal),$$

where, given a configuration *C*, $received(C, P)$ is the set of signals stored in the queue of process *P*. The definition of the *received* function is dynamic, in the sense that it depends on the configuration.

Once all these elements have been defined, priorities can be calculated as follows. We will assume a function $pri : Config \times Process \longrightarrow N$ which will change every time the configuration system changes; that is, every time a process receives a new signal or when a process proceeds to a new state. Function *pri* assigns priorities to processes considering the algorithm below. The priority changes are made by the run-time system (scheduler). An implementation of this model can be seen in [31].

- Initially, every process has its default priority or the priority assigned to its *START* transition.
- If process *P* is dealing with signal sg_0 in state *st*, and it receives a new signal *sg*, the priority changes according to the following rule:

$$pri(C, P) \leftarrow \begin{cases} \max\{pri(C, P), p\} & \\ & \text{if } nextst(P, st, sg_0) = \{st'\} \\ & \text{and } (sg, p) \in sig(P, st') \\ pri(C, P) & \text{otherwise.} \end{cases} \quad (1)$$

  The new priority is the maximum between the current priority and the priority of the transition enabled by the received signal *sg*, if it can be executed in the state reached by the current transition and if we can ensure that this next state is unique. In this case, this priority change can be considered a kind of priority inheritance in the SDL process between the transition currently under execution and the next transition to be achieved. If the executing transition can reach different states depending on some condition, then we cannot apply the previous rule, because we cannot ensure what will occur in the future and it is possible that the current transition reaches a state where the signal *sg* cannot be attended.
- When (in a given configuration *C*) process *P* changes to state *st* or being inactive in a state *st* receives a new signal, its priority changes as follows:

$$pri(C, P) \leftarrow \max \left\{ \begin{array}{l} p \mid \exists sg \in received(C, P), \\ (sg, p) \in sig(P, st) \end{array} \right\}. \quad (2)$$

  Its priority is recalculated to be equal to the priority of the higher priority enabled transition. If no enable signal is contained in the queue then we consider a priority of 0 until set *received* is not empty.

Note that rule (1) can be applied when the next state of the current transition is unique. However, the priority assignment may be improved when a transition reaches different states and all of them can attend to the new received signal. In this case, we can raise the priority to the minimum of the transitions enabled by signal *sg* as follows:

$$pri(C, P) \leftarrow \max \left\{ \begin{array}{l} pri(C, P), \\ \min\{p : \forall st' \in nextst(P, st, sg_0), \\ (sg, p) \in sig(P, st')\} \end{array} \right\}. \quad (3)$$

We could consider other alternatives, but not all of them would be fair. For example, if we raise the priority to the maximum priority of the transitions enabled in the next state (rule (4)), then the activated transition in the next state could have less priority than the raised priority and we would be interrupting other transitions with an intermediate priority, possibly provoking an inversion priority. Nevertheless, a good alternative for future work may be to apply the priority changes by introducing some probabilistic study on the transitions. However, this would imply a new proposal for response time calculation, different from that presented in Section 2.4:

$$pri(C, P) \leftarrow \max \left\{ \begin{array}{l} pri(C, P), \\ \max\{p : \forall st' \in nextst(P, st, sg_0), \\ (sg, p) \in sig(P, st')\} \end{array} \right\}. \quad (4)$$

2.3.1. *Assigning priorities to transitions*

Timing requirements are specified with respect to external events, also considering timer expirations as external events. For each event in the system we consider the sequence of signals exchanged among the SDL processes. This sequence is constructed from the external event by adding to the sequence the signals sent by subsequent processes. If every process only sends a single signal to another process, we obtain a sequential string of transitions, corresponding to actions that have to be carried out on the event occurrence.

Downloaded from http://comjnl.oxfordjournals.org/ at Pennsylvania State University on September 15, 2016

For instance, for the example of Figure 5 we obtain a sequential sequence string for both external events:

$$Senv1 \longrightarrow S1 \longrightarrow S2$$
$$Senv2 \longrightarrow S3 \longrightarrow S4.$$

The actions achieved by the different processes can be considered equivalent to actions achieved by a single sequential task and, initially, all the transitions involved in the same sequential string will have the same priority. This priority will be determined by rate-monotonic or deadline-monotonic assignment based on the timing requirements of the external event.

If a process can send more than one signal in a single transition, each of the signals is considered a new (internal) event that has to be taken into account in the timing analysis. The priority of the transitions involved in the treatment of this new event will be determined by those of the external event.

Additionally, if we have transitions that can reach different states depending on a condition, then we must include in the real time analysis all the possible sequences of signals in the response to the event and we will select the worst-case response time.

Another aspect to take into account is transition sharing. In some systems, transitions may be shared by two or more events. However, a transition can only be executing on behalf of one event at a time. In this situation priority inversion can occur if we assign the transition priority arbitrarily. A possible solution to this synchronization problem is to use the highest locker protocol [10]. In this protocol all shared resources have a ceiling priority, which is the highest priority among the processes that can access the resource. Any process trying to access the resource changes its priority to one level higher than the resource priority ceiling. In this way, if the process has access to the resource then it is not preempted by other processes that also want to use it. This protocol can be used directly to solve the problem of shared transitions, considering the transition itself as a shared resource. In this sense, the transition priority will be one level higher than the maximum priority corresponding to the events in which it is involved, avoiding possible priority inversion problems.

2.3.2. *Sharing resources*

As we analysed in Section 2.2, resource sharing in SDL can lead to priority inversion situations. This situation can be avoided by using the execution model described above, but data and resource sharing can end up very inefficient and difficult to discuss, since it may involve several message exchanges and process context switches. In our model, shared data and resources will be encapsulated into a special kind of process. These processes are externally normal SDL processes, but their behaviour is limited in the following ways:

- they act as passive server processes, i.e. they do not initiate any action by themselves;

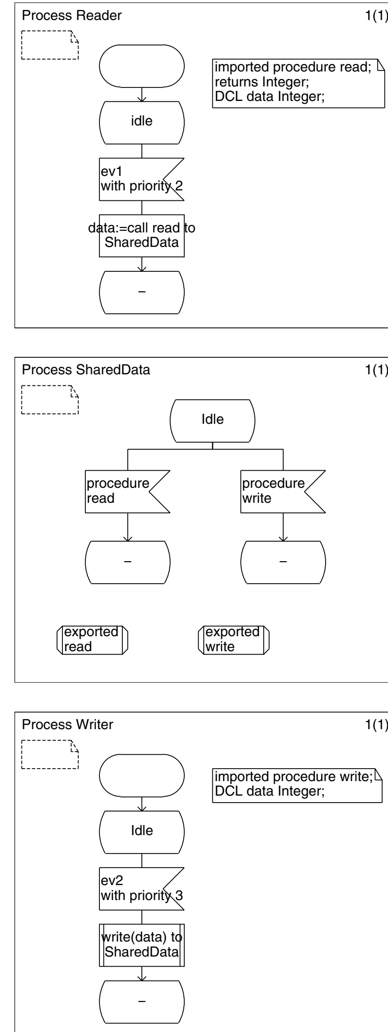


FIGURE 6. Resource sharing example

- they only use remote procedure calls (RPCs) as their communication mechanism, i.e. they are always waiting to receive RPCs from other processes;
- blocking during transition execution must be bounded.

Each of these processes has assigned a priority ceiling, which is the maximum of the priorities among all the other process transitions where the resource is accessed. In this way we avoid possible priority inversion in data access and blocking time in shared data access is predictable. Mutual exclusion is also guaranteed, since all the process transitions will be executed at the highest priority among all the processes sharing the resource. In Figure 6 we show an example of how a `reader` and a `writer` process access shared data using this schema. The priority ceiling of the `SharedData` process will be `4` since it is accessed from two different transitions, one with priority `2` and the other with priority `3`.

In addition, using this schema for modelling data sharing has another important advantage: it can be implemented very efficiently. Although in the SDL model data are

Downloaded from http://comjnl.oxfordjournals.org/ at Pennsylvania State University on September 15, 2016

encapsulated in a process, this process is not really translated to a real process in the implementation. Each of these processes can be translated into a set of procedures, one for each transition, inside a module in the target language. These procedures will be called by the processes that share the data after changing their priority to the priority ceiling of the resource. This implementation technique is similar to how Ada95 implements protected types [32].

2.3.3. Hardware integration

As we discussed in Section 2.2, hardware control software must be included in the SDL design. We provide a simple general model for the integration of interrupt-based hardware in an SDL design. Every hardware component is modelled by two different processes.

- A passive process, such as those introduced for data sharing. These passive processes execute transitions only as a result of a hardware interruption or when their associated driver processes call them. Their priority ceiling will be a hardware-level priority and will be system dependent. Only critical hardware operations will be achieved inside this process and with this high hardware priority.
- An active driver process that will interact with the passive one and that will provide the access interface to the rest of the processes in the system. Interaction with this driver process can be asynchronous or synchronous. Its priority will be determined by the same rules as the rest of the processes in the system (i.e. depending on the external events they deal with). Hardware registers are described in terms of SDL data types and they will be mapped onto the physical ones during the final implementation phase. These registers are all encapsulated in the passive process.

Hardware interruptions will be modelled by external SDL signals that will be received by the passive processes. The interrupt priority will be indicated in the transition that will attend to that interruption. This transition will be the only one in the state in which the process is waiting for the interruption (in order to prevent the possible interrupt blocking that could occur if the process is attending any call from the driver). Interrupt notifications are sent asynchronously to the driver process (so they can be buffered). With this interrupt-handling model, we have three different priority levels.

- The highest which attends to the interruption.
- The one indicated by the passive process ceiling, which indicates the priority for hardware operations.
- The driver priority, which will be determined by the priority of the events it deals with. In this way the amount of computation carried out at high priority can be minimized, increasing system schedulability. In the examples of Figures 7 and 8 we show the structure of a passive hardware process and a standard hardware configuration.

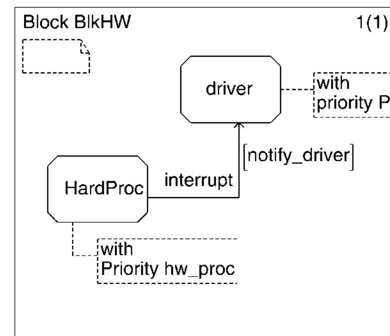


FIGURE 7. Hardware interaction processes.

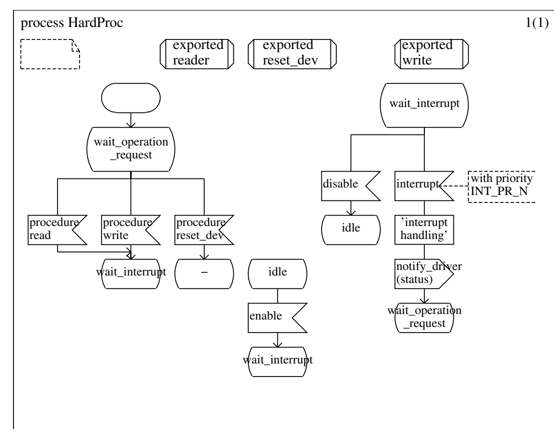


FIGURE 8. Hardware access process.

2.4. Timing analysis

In order to integrate the timing analysis of the system in the proposed methodology, we illustrate how the timing behaviour analysis of a complete system can be achieved. This analysis considers that the deadlines of the events in the system are lower than the periods and that the system is executed in a single processor architecture. An initial study for the integration of the schedulability analysis in SDL was proposed in [33]. Based on these previous works and [34] we propose a more complete analysis.

2.4.1. Basic SDL set

In this section we describe the characteristics that the systems specified in SDL have to carry out to integrate the schedulability analysis. These characteristics do not reduce the expressiveness of SDL, but indicate the way to specify the system to be able to integrate the real-time analysis. The initial considered set is composed by n SDL processes where each process state has one or more signal receptions and zero or more signal sendings, that is, a transition in a SDL process can activate several transitions in the system. In Figure 9 we can see two possible examples of transitions to carry out the schedulability analysis.

As we commented in Section 2.3, every transition in the SDL system is labelled with a transition priority and, in order to make the timing analysis, all the transitions will have an associated worst-case execution time.

Downloaded from http://comjnl.oxfordjournals.org/ at Pennsylvania State University on September 15, 2016

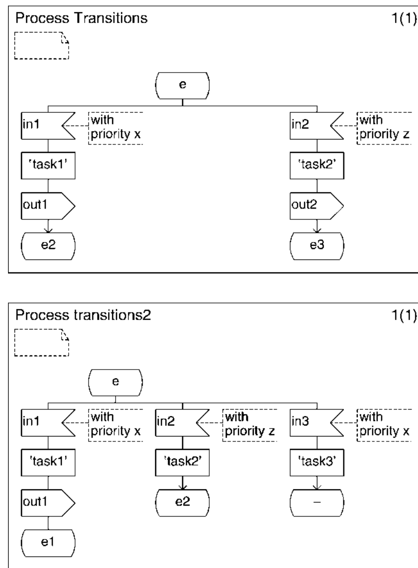


FIGURE 9. SDL basic set.

We consider the response time of an external event in the system as the sum of the response times of the transitions that take part in the response to the event. If we observe the example of Figure 5, there are two external events whose response sequence is composed of the transitions `senv1`, `s1`, `s2` for the external event `ev1` and `senv2`, `s3`, `s4` for the external event `ev2`. For instance, we obtain the response time of the event `ev1` by summing up the response times of the transitions activated by the signals `senv1`, `s1` and `s2`.

2.4.2. *Response-time calculation*

To calculate the response time of a transition in a SDL system, we take into account the interference of the higher-priority transitions and the blocking time of the lower-priority transitions. In order to get more accurate results, we propose to add the following considerations.

- Transitions responding to an external event are related by precedence relationships. This can reduce the number of transitions that participate in the interference calculation.
- The SDL semantics influence the response time.

Now we consider a set of events (external and internal) $S_1, \ldots, S_n$ and the sequence of transitions $t_{i1}, \ldots, t_{in}$ that respond to every event S_i. All these transitions belong to the SDL processes in the system. Finally, we apply this analysis to SDL systems with a finite (relatively small) number of events.

2.4.2.1. Precedence relationships. Let us suppose an external event S_i composed of the sequence of transitions $t_{i1}, \ldots, t_{in}$ that respond to this event. If we want to calculate the interference of the tasks of the event S_i over transition t_{ab} that belongs to event S_a, we take into account all the transitions with priority higher than or equal to the priority of transition t_{ab} that belongs to event S_i.

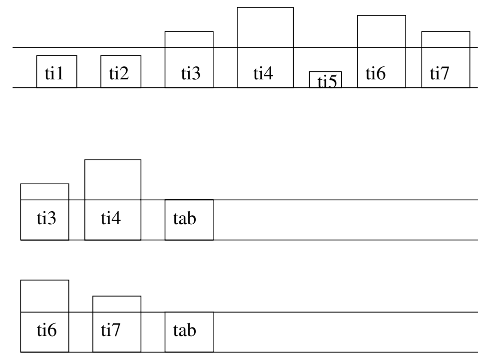


FIGURE 10. Interference for task t_{ab}.

However, the execution of the transitions in the event S_i, has an order in the execution due to the precedence relations between the transitions. When the execution of transition t_{i1} finishes, then transition t_{i2} begins and so on with the rest of the transitions in the event. Let us suppose that there exists a transition, t_{ij}, in event S_i that has a lower priority than task t_{ab}. In this situation t_{ab} can only be interrupted by sequence $t_{i1}, \ldots, t_{ij-1}$ or sequence $t_{ij+1}, \ldots, t_{in}$, but never by t_{ij}. We will have to analyse the sequence of transitions that give the worst case interference time. Now consider the situation shown in Figure 10, where we want to calculate the interference of event S_i with respect to transition t_{ab}. The height of the box indicates the priorities of the transitions and the horizontal line indicates the priority of the transition we want to analyse (t_{ab}). There exist two sequences of transitions that can potentially interrupt task t_{ab}, but at most only one of them will really be able to do it. We will include the sequence with the worst execution time in the interference of transition t_{ab}.

2.4.2.2. Including SDL semantics. The SDL semantics do not allow two transitions belonging to the same SDL process to be executed concurrently. Let us suppose that there are two transitions, t_{ij} and t_{kj}, that belong to the same SDL process but that take part in different responses to external events, S_i and S_k, and the priority of t_{ij} is greater than that of t_{kj}. If we are calculating the response time of transition t_{kj}, then transition t_{ij} will be included in the interference expression. However, t_{ij} will never be able to interrupt the execution of t_{kj} and it should not be considered. It reduces the number of transitions that can interrupt other transitions but, as a consequence of this, we have to consider an additional blocking time called *run to completion blocking*. In Figure 11, we have an external event S_i and the sequence of transitions t_{i1}, $t_{i2}, \ldots, t_{i5}$ that respond to this event and we want to calculate the interference for t_{ab}. As we can see in the figure, every transition has its fixed priority. In Figure 12 we can see the relationship between the priorities of the transitions of event S_i and transition t_{ab}. The dashed horizontal line indicates the priority of t_{ab}.

Transitions t_{i3} and t_{ab} belong to the same SDL process and so t_{i3} cannot preempt t_{ab}. If we take into account

Downloaded from http://comjnl.oxfordjournals.org/ at Pennsylvania State University on September 15, 2016

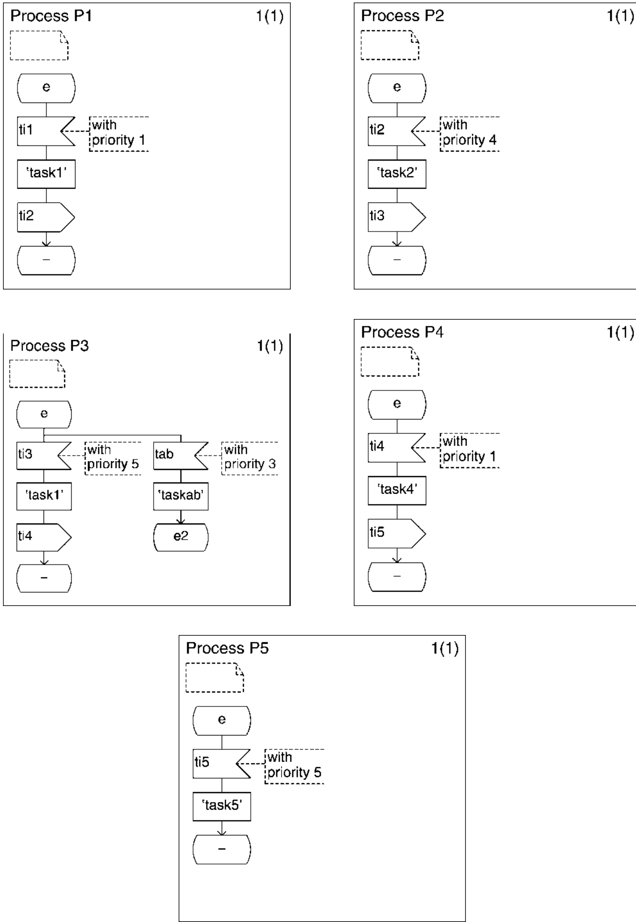


FIGURE 11. Computing t_{ab} interference in SDL.

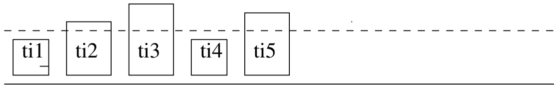


FIGURE 12. Relationship between t_{ab} priority and priorities of tasks in S_i.

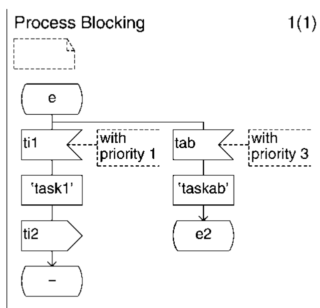


FIGURE 13. Computing the t_{ab} blocking time.

the precedence relationships, then we will select the worst execution time between the sequences $\{t_{i2}, t_{i3}\}$ and $\{t_{i5}\}$. However, if we take into account the SDL semantics we will select between the sequences $\{t_{i2}\}$ and $\{t_{i5}\}$ because t_{i3} cannot interrupt t_{ab} as they share the same SDL process.

Although the previous situation can reduce the number of transitions that can interrupt other transitions, the blocking time can be increased by adding the run to completion blocking. Let us suppose that we are calculating the blocking time of transition t_{ab} and, as we can see in Figure 13, t_{i1} and t_{ab} belong to the same SDL process and the priority of t_{i1} is lower than the priority of t_{ab}. Initially, t_{i1} does not take part in the response time of t_{ab}, but if we integrate this type of analysis in SDL, the execution time of t_{i1} has to be considered like the run to completion blocking time of transition t_{ab}.

In the following we detail the steps to develop the worst case interference time and the blocking time.

2.4.2.3. Worst case interference time. In order to calculate the worst case interference time of transition t_{ab} we take the following steps.

- Select the consecutive transition sequences that have higher priority than t_{ab} and do not share the same SDL process as for each event in the system. Several sequences can appear in every event.
- If there are several sequences for the same event then we select the worst execution time sequence.
- Calculate the interference using the expression proposed in [33]. We call this expression I_{ab}.

2.4.2.4. Blocking time. In order to include the blocking time in the schedulability analysis we have to consider two possible blocking sources. The first is related to the access to shared resources. We encapsulate these shared resources in passive processes accessed by means of RPCs. Using the highest locker protocol, this blocking time is bounded (see Section 2.3). We call this blocking source B_{sh}.

The second possible source is due to the run to completion blocking, B_{rtc}. If we define the function $psdl$: *Transition* $\longrightarrow$ *Process* returning the SDL process that this transition belongs to, the expression of B_{rtc} for transition t_{ab} is the following:

$$B_{rtc}(t_{ab}) = \max\{c_{ij} : p_{sdl}(t_{ij}) = p_{sdl}(t_{ab}) \text{ and } t_{ij} \neq t_{ab}\}. \quad (5)$$

The blocking time is the maximum worst execution time among the transitions that belong to the same SDL process of the analysed transition t_{ab}. We do not include transition t_{ab} if it is not shared by more than one event.

If transition t_{ab} takes part in the response to two external events, e_1 and e_2, the same transition cannot execute concurrently answering both events. In this case the transition itself has to be considered as a possible blocking source. In this case the blocking expression is

$$B_{rtc}(t_{ab}) = \max\{c_{ij} : p_{sdl}(t_{ij}) = p_{sdl}(t_{ab})\}. \quad (6)$$

The expression of the response time of transition t_{ab}, R_{ab}, that participates in the response of an external event S_a is the following:

$$R_{ab} = I_{ab} + \max\{B_{sh}, B_{rtc}\} + c_{ab} \quad (7)$$

where c_{ab} is the worst execution time of transition t_{ab}.

Downloaded from http://comjnl.oxfordjournals.org/ at Pennsylvania State University on September 15, 2016

The response-time calculation is outside of the scope of this paper. In [31] a complete response-time study is shown. In this paper we show that the modifications to the standard techniques applied to SDL allow us to obtain response times more similar to the real execution.

3. THE METHODOLOGY

Our proposal is based on the experiences in the design of embedded systems and is conceptually based on the approach already used in other methodologies (SOMT, Octopus, UML). However, we provide partial solutions to some of the problems that are present in the methodologies mentioned above and also in other object-oriented development techniques. In fact, we deal with the existing gap between the object model and the process model, which presents a special interest in the context of real-time embedded systems. We try to fill this gap by considering two orthogonal strategies. Thus, we provide an approach to move the object model into a SDL process model, in such a way that the real-time constraints can be predictable. The design criteria to be applied when a class (or object) in the object model has to be mapped to one (or more) processes (e.g. described in SDL) are summarized in Section 3.2. Additionally, we propose a way to map associations in the object model to signal routes in the design model. Each method associated to each object will have an associated signal in the design model. This way of mapping the object model into the process model is an important aspect of our proposal and it provides a clear strategy to convert objects into processes, giving the possibility of analysing real-time properties. However, we have learned from our experience in developing real-time embedded systems two other lessons. Firstly, the object model can be refined more than other methodologies recommend and in some occasions this finer object-oriented design is encouraged. Secondly, this refinement can be better exploited if we obtain a good correspondence between elements in the object model (UML) and elements in the process model (SDL). Thus, we think that this detailed mapping between both models presents some novelties with respect to the suggestions made in other proposals.

The primary focus of this method is on system development using object-oriented analysis and design, but providing real-time analysis based on SDL. The proposed methodology is basically divided into four phases: `Analysis`, `Design`, `Validation` and `Implementation`. In every phase, both the functional and the non-functional aspects are considered in two different, but complementary, views of the system. This distinction allows us to capture the real-time features and the hardware peculiarities of the system, with (relative) independence from the functional requirements (object and dynamic models), but keeping track of the level where the specific requirements have arisen.

The different phases for each aspect are shown in Figure 14. As has been mentioned, this figure also shows how each phase is vertically divided into two views, giving the functional and non-functional perspectives. Note that the expressions used to denote each phase are different depending on what view we want to focus on. The first phase is called `Analysis` from a functional perspective, whereas we use `Real-time requirements specification` for the non-functional view. Each of these views is subdivided into two other views (called models) distinguishing the object model from the dynamic model in the functional perspective and the real-time model from the hardware constraints in the non-functional perspective.

In the *analysis* phase the problem domain and the user requirements on the system are captured and analysed. Basically, during this phase we establish a first object model and the preliminary use cases of the system. We use three graphic notations to support both activities: UML class diagrams to capture the object-oriented description and use case diagrams and MSCs to describe the dynamic model. A MSC is a kind of sequence diagram and we use it because it can be validated with the design in SDL.

During the *design* phase the functional aspects of the system are discussed at two different levels: system and object. The system design is basically made using class and object diagrams (UML), whereas the object design is made by means of SDL specifications.

In the *simulation, validation and performance evaluation* phase the results of the analysis and design phases are contrasted. The simulation and validation of the system is made using the SDL specification. There are some tools with these capabilities (i.e. SDT), which apply a number of known techniques to validate SDL specifications: verification use cases, validation traces, model checking, etc. These techniques allow us to detect possible general design errors such as deadlock, implicit signal consumption, etc. In addition, some particular scenarios of the system can also be studied by means of the verification of MSC behaviour specifications. These MSCs can describe required or forbidden system behaviour that can be automatically verified by existing tools. In this sense, it is possible to use the refined MSC in the previous phase to verify part of the functionality of the system.

Finally, in the *implementation* phase we obtain the code corresponding to the application-specific part of the system. This application-specific code typically relies on generic services provided by an underlying layer (operating system or SDL run-time system). This layer must provide certain characteristics in order to support our real-time execution model described in Section 2.3. At the very least, it must provide fully preemptive scheduling based on fixed priorities and some form of priority inheritance. In addition, the hardware model should be adapted to the one provided (if any). Current commercial code generators can be divided into two groups: those that use an existing operating system and which adapt the SDL execution model to it and those that are based in a customizable run-time system. For our approach to embedded systems, we think that the second option is more suitable, since it allows a more direct translation of the execution model and a tighter integration of hardware devices in the SDL design. We have developed

Downloaded from http://comjnl.oxfordjournals.org/ at Pennsylvania State University on September 15, 2016

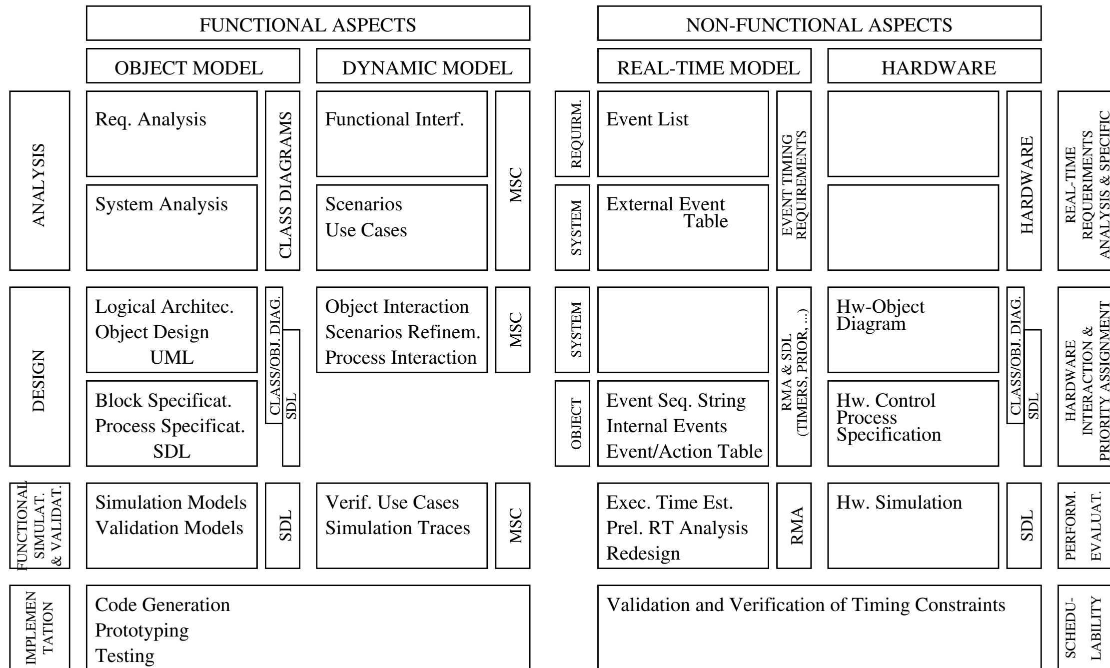


FIGURE 14. Methodology phases.

a prototype of a run-time system support for a subset of SDL that implements the real-time execution model. With this support, the implementation phase is mainly devoted to completing the functions for the hardware access and a successive design refinement. In the schedulability analysis stage, we complete the timing analysis carried out in the previous phase. To achieve more accurate results, several factors must be included in the analysis model. For example, aspects such as timer implementation, process and signal queue management, interrupt handling, etc. must be taken into account. All these factors are very system dependent and they should be measured and adjusted in order to obtain a reliable model.

In the following we describe the new aspects of the methodology that we present in detail, showing the stages where we make some relevant contribution. In order to maintain the temporal relation between functional and non-functional aspects, we will present every phase giving the functional and non-functional views at the same time.

3.1. Real-time requirements analysis and specification

This phase coincides with the *analysis* phase, but it is devoted to describing the non-functional aspects. A real-time system responds simultaneously to random occurrences of events and gives the desired responses within the required time limits. Thus, this phase will focus on the detection of the timing requirements of the external events (specially input events) produced by the hardware interaction and other subsystems. The result of this activity is a table with the external events, where the following information is given for each event: event name, involved classes, and real-time requirements (rate, deadline, jitter, etc).

3.2. Design, hardware interaction and priority assignment

In this phase, an important difference of our proposal with respect to other methodologies (e.g. SOMT) is the possibility of narrowing the existing gap between the object design and the SDL design. This is possible due to the activities (hardware interaction and priority assignment) which are being developed during this phase (see Section 3.2.3) to capture the non-functional aspects.

3.2.1. Object level design

The object design is made mainly in SDL. However, the class/object diagrams are still used until the last stages of this phase. Thus, during this design level, blocks and processes are specified in SDL, taking as a starting point a rather refined object design. Perhaps this is the activity mostly influenced by the real-time and hardware requirements, which are analysed in the following section. Nevertheless, independently of these non-functional requirements, we also propose a number of new strategies and criteria to map objects into processes, which we learned from our experience in developing real-time embedded systems. In particular, we propose a sophisticated way (more elaborated than in other SDL-based methodologies) to map elements from the object model into SDL entities. Some of these strategies correspond to the mapping between associations (in the object model) and signal routes (in SDL), or the mapping of active and passive objects into different kinds of processes in SDL. Concerning the first issue, we establish a mechanism to associate subsets of methods (of a class) with associations. This makes the translation to signal routes and signal lists easier.

Downloaded from http://comjnl.oxfordjournals.org/ at Pennsylvania State University on September 15, 2016

On the other hand, like in other methodologies, objects are mapped into processes by following clear criteria such as: active objects are always moved to one process at least; in some situations several processes have to be considered (e.g. when the object is involved in two parallel events which cannot fulfil some time constraints—in this case, it is necessary to define one process for each event). Passive objects not shared by different active objects are modelled as internal data types defined in the client process. If the passive object is shared we consider a process with the limitations established in Section 2.3.2. Finally, we also deal with the classes which model the hardware behaviour and propose the distinction between active processes (hardware drivers) and passive processes (directly modelling the hardware), as was discussed in Section 2.3.3.

During the design of the functional aspects of the system, the non-functional features are also taken into account. In this stage, *hardware interaction and priority assignment*, two levels are still maintained in this activity: the system level and the object level.

3.2.2. *System level hardware interaction and priority assignment*

Concerning the non-functional issues, the design has to capture how the hardware interacts with the system. In this sense, each hardware component must be modelled as an object; therefore, a class has to be defined in order to represent the corresponding physical component. The activities to be developed in this phase are similar to those proposed in other methodologies, e.g. Octopus. However, the guidelines to group the classes modelling the hardware are different with respect to the approach promoted by Octopus. In fact, as was mentioned in Section 2.3.3, instead of grouping all these classes in only one subsystem, we propose to distribute the hardware classes among the different subsystems that have been generated during the analysis phase. This promotes a more natural and reasonable design, because the classes representing every hardware component are located where they will be needed. Basically, the notation we propose at this level is UML (class-object diagrams).

3.2.3. *Object level hardware interaction and priority assignment*

The object level of this phase is devoted to designing the hardware interaction, as was discussed in Section 2.3.3, distinguishing for each hardware component a passive process and a driver process. The first models the accessing of the hardware component and the second implements the protocol demanded by the system requirements. All this is specified in SDL. This way of designing the hardware helps one to decide what is hardware and software and to simulate the system in the design phase. The hardware processes will disappear in the implementation phase.

During this phase other activities are carried out. On the one hand, the event sequence string is completed, associating objects to events or fixing the priorities of the transitions (see Section 2.3.1). On the other hand, the passive process ceilings are assigned upon it, as described earlier. Finally, a table must be constructed showing the correspondence between every event and the related sequence of transitions. In addition, every transition has some associated information, such as the blocking delay time (if any), whether it is atomic or not, the worst case execution time of the transition and the relative priority.

3.3. Performance evaluation

In this phase we use the simulation models to analyse quantitative measures like throughput and response time. This analysis can also be carried out with existing commercial tools, although the integration of performance evaluation in SDL has not been fully established. Concerning the hardware interaction, hardware must be simulated by SDL processes. When the system is implemented, the hardware will be accessed by means of C functions embedded in the code generated from the design. However, during this phase, we have to complete the SDL specification with SDL processes modelling the hardware behaviour. This will allow us to simulate and validate the whole system, also taking into account the interaction with the hardware. Another relevant aspect corresponds to the real-time requirements. After the design, a preliminary real-time analysis is necessary, which will be based on the analysis proposed in Section 2.4. Using this analysis it is possible to know the timing behaviour of the system before the implementation phase and make the necessary changes to improve the response time of the system. In this sense, the *performance evaluation* phase also includes a redesign stage to refine the system specified in SDL. Based on our experiences, this stage proposes a set of heuristics to change the design and improve the response times of the events in the system. This stage will be especially useful when the system does not meet the deadlines. The changes in the design will affect the parameters in the response time equation (see Section 2.4) like blocking and interference. We propose a set of heuristics that allow us to:

- redesign the system if it does not meet the deadlines;
- take into account the real-time requirements from the first stages of the design.

The heuristics can be summarized as follows:

- task transference, reducing the interference with other transitions;
- process creation, reducing the blocking time;
- intermediate transition elimination, eliminating transitions in the SDL design.

3.3.1. *Task transference*

This heuristic may be applied for transitions that are not shared for more than one event. In this case, the technique modifies the design in two steps.

- It looks for consecutive transitions that take part in the response to an event that does not meet its deadlines

Downloaded from http://comjnl.oxfordjournals.org/ at Pennsylvania State University on September 15, 2016

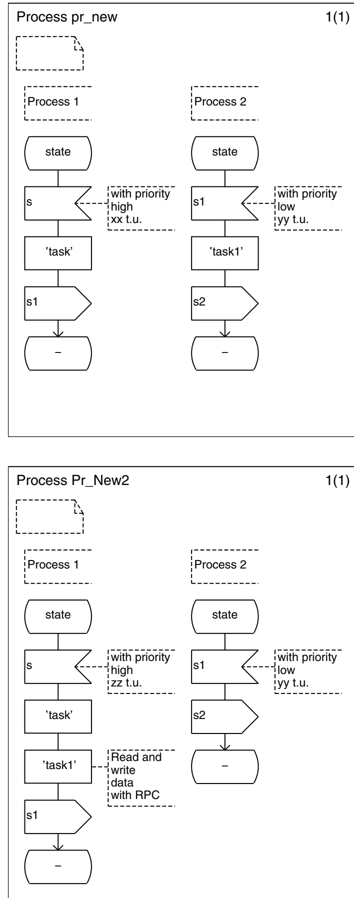


FIGURE 15. Task transference in SDL processes.

where the priority of the first is higher than the priority of the second, or *vice versa*.

- It transfers the task from the lower-priority transition to that with the higher priority. The response time will decrease in most of the cases since the lower-priority transition will execute with higher priority.

This way we reduce the interference of the lower priority transition since there will be less computation time where this transition could be preempted. Although this heuristic can increase the response time of other events, it can make all the events in the system meet the deadlines. If we take into account the SDL design, it is not an easy solution because two consecutive transitions do not belong to the same SDL process. If we have two consecutive transitions, t and t′, that belong to SDL processes p and p′, respectively, and the priority of t is higher than t′, then we transfer the t′ computation to t. Due to the SDL semantics, a SDL process cannot have access to the variables of other process, so t′ computation cannot have access to its variables. We consider these variables like shared resources and they have to be accessed by means of RPC (see Section 2.3). This can increase the response time since the blocking time (B_{sh}) of both processes can be affected. However, as we use the priority ceiling protocol, the blocking time is bounded and we have to select the maximum possible blocking time

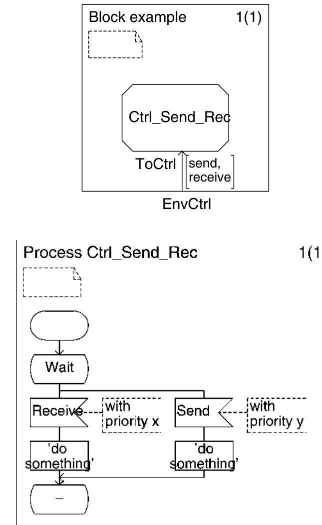


FIGURE 16. Initial design.

between B_{rtc} and B_{sh} for all the transitions. In most of the cases B_{rtc} is longer than B_{sh} and then the last blocking source does not affect the response time. As we can see in Figure 15, process 2 maintains the signal reception and sending, but these operations are atomic. This transition will disappear by applying *intermediate transition elimination*.

3.3.2. *Creating SDL processes*

Many processes in the design of a system with SDL can be designed as in Figure 16 where there exists a state in a process with more than one signal reception and we consider that both transitions would be able to be executed concurrently. If we look at the figure, the process activation can occur due to the reception of signals `receive` or `send`. However, we have to take into account this situation if we want to design real-time systems in SDL, because it increases the blocking time in the applications and reduces the concurrency, which is very important in this kind of system. In the figure it is not possible to execute transition `send` until the completion of transition `receive` and *vice versa*. As commented on in Section 2.4, it produces a run to completion blocking. The solution is to create two different processes to send and receive data concurrently as shown in Figure 17. We have to have two considerations. First, the system is executed in an uniprocessor system, so the representation proposed to access shared resources (RPCs) is implemented efficiently. Second, although we make some changes in the design, the original design does not disappear. It is a new phase of our methodology and these changes are carried out when the system does not meet the deadlines. It is possible that the changes affect functional cohesion, but on the other hand the timing requirements can be met.

3.3.3. *Intermediate transition elimination*

This design technique allows us to reduce the concurrency in the systems because it can eliminate processes in the

Downloaded from http://comjnl.oxfordjournals.org/ at Pennsylvania State University on September 15, 2016

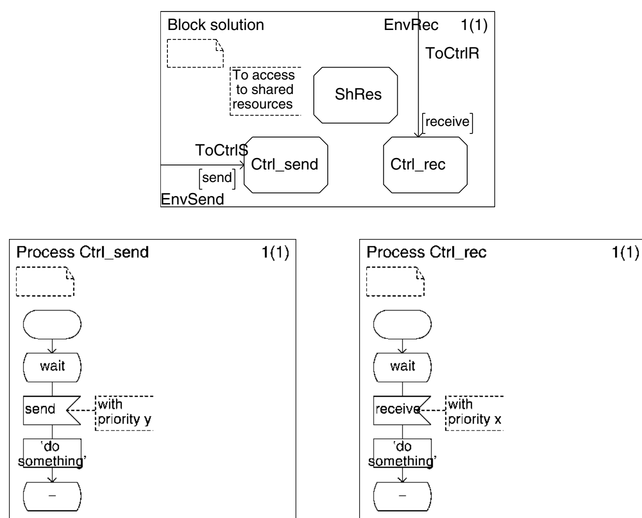


FIGURE 17. Final design.

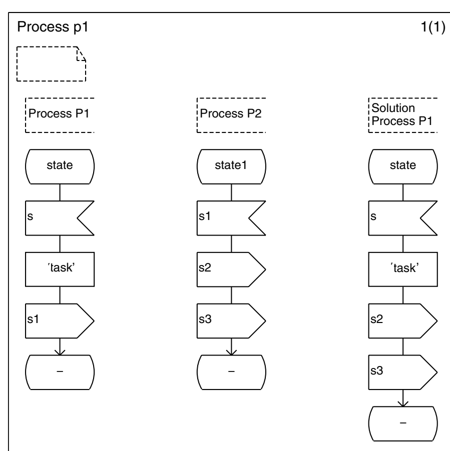


FIGURE 18. Intermediate transition elimination.

design without affecting system responsiveness. This can be seen in Figure 18. It basically consists of eliminating not-shared intermediate transitions that have no real computation to achieve. It is necessary to take the signal sendings of the eliminated transition to the predecessor. In this way, there are fewer transitions participating in the schedulability analysis, but the responsiveness in the system is maintained. If the transition that is going to be eliminated is shared for more than one event then this transition will have to be replicated to be able to apply this heuristic.

4. A CASE STUDY: THE DESIGN OF A CORDLESS TELEPHONE

Eole 400 is a multihandset cordless telephone of the CT0 class, with a solid state answering machine integrated at the base. It is based on two previous CT0 products from Alcatel, the France Telecom X1 (for the handset and the radio circuit at the base) and the EOLE 300 (for the base). The system will be made of a fixed part (the base) and one or more handsets. The main software characteristics implemented in the telephone allow the following features:

- intercommunication between base and handsets;
- automatic recall to the last dialled number;
- loudspeaker activation during communication;
- encryption of the communication base-handset;
- decimal/multifrequency dial;
- three programmable melodies at the handset and ring-off option;
- answering machine with simple and recorder mode and remote access to the answering machine through a security code.

4.1. Hardware description

The Eole 400 base is made of several electronic circuits, some of them programmable.

- The 4-bit microcomputer NEC μPD75116, with 8192 $\times$ 8 bits of program (ROM) memory, 512 $\times$ 4 bits of data (RAM) memory, 34 I/O pins, an 8-bit serial interface and a clock at a frequency of 4.19 Mhz. The minimum instruction execution time is 0.95 μs. The I/O pins connect the LEDs and the keyboard to the microcomputer.
- A single-chip EEPROM memory, 24c01, with a 1 Kbyte storage capacity.
- A radio circuit, known as COMBO, based on the Motorola MC13110 chip.
- A conversational circuit with loudspeaker based on the Motorola MC34216A chip.
- An answering module, known as ATISAM, based on the Texas Instruments MSP58C80 chip.

In order to begin a call, the user has to press the `line` key at the handset and a line request message is sent to the base. The radio circuit detects the message and informs the system. If the base is able to process it, i.e. it is not already in a communication state, a radio link is established between the base and the handset and the telephone line hook is enabled. To go back to the idle state, the user has to press the `line` key again. If, for example, the `intercom` key is pressed on the base, it begins the actions to communicate the request to all the handsets by broadcasting a message through the radio circuit. If a handset answers the request, by pressing the `line` key at the handset, an intercommunication is established between the handset and the base, thanks to the integrated microphone and loudspeaker. Otherwise, if there is not an answer within the fifteen seconds, it is assumed that nobody is interested in the intercommunication and the base goes back to the idle state. Although the system is distributed (fixed part and handsets) we only model the fixed part of the system; that is, we apply our methodology to a monoprocessor system.

4.2. Applying the methodology

In this section we deal with the main models of our proposed methodology, applying them to the telephonic system developed.

Downloaded from http://comjnl.oxfordjournals.org/ at Pennsylvania State University on September 15, 2016

TABLE 1. External event table.

External event	Objects involved	Time requirements
Incoming call	System, LEDs, COMBO, conversational circuit, phone line	Scan line and no signal present. Period: 0.122 ms. Deadline: — Detect signal in line. Period: — Deadline: 700 ms
Outgoing call	System, LEDs, COMBO, phone line	Scan channels and no carrier. Period: 0.122 ms. Deadline: — Carrier detected. Period: — . Deadline: 1100 ms
Intercommunication	System, LEDs, keyboard, COMBO	Scan keyboard and no key pressed. Period: 7.82 ms. Deadline: — Intercom key pressed. Period: — . Deadline: 1200 ms
Change volume	System, keyboard, EEPROM, conversational circuit	Scan keyboard and no key pressed. Period: 7.82 ms. Deadline: — Volume key pressed. Period: — . Deadline: 2000 ms
Change melody	System, keyboard, EEPROM, conversational circuit	Scan keyboard and no key pressed. Period: 7.82 ms. Deadline: — Intercom key pressed. Period: — . Deadline: 2000 ms

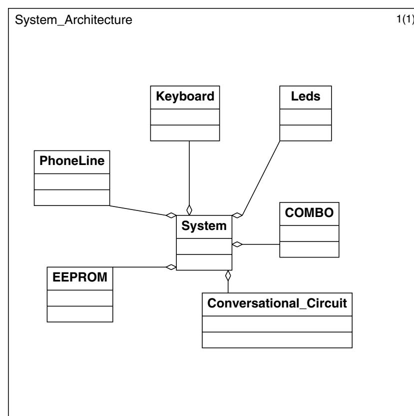


FIGURE 19. System object architecture.

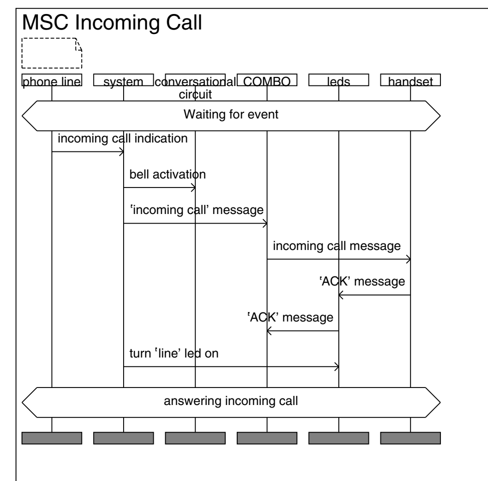


FIGURE 20. MSC for incoming call scenario.

The first part of our methodology is the analysis phase. The functional aspects of this phase can be divided into the requirement and system levels. The resulting documents of the requirement level are cases developed from the user manual, technical specification documents and interviews with the engineers involved in the project. In the system level a class diagram is developed in order to show the relations among the objects identified in the study of the requirement level documents. This diagram is shown in Figure 19. After this, we make MSCs to study the dynamic behaviour of the objects in the different possible situations. In Figure 20, we show the MSC for an `Incoming call` signal detection in the phone line. The non-functional aspects covered in this phase are those relative to the system external events. The external events are shown in Table 1 with their names, time requirements and involved objects as commented in Section 3.1.

The design, hardware interaction and priority assignment phase is divided into the functional aspects and the real-time and hardware issues. In the system level, we made a logical structure of the classes to design the system architecture which is refined to get a more precise definition of the system. These models are shown in Figures 21 and 22. It is important to point out the mapping between objects and blocks and between associations and signal routes commented in Section 3.2 (see Figure 23). In the system level of the non-functional aspects we make a more precise study of the hardware objects as commented

Downloaded from http://comjnl.oxfordjournals.org/ at Pennsylvania State University on September 15, 2016

TABLE 2. Event/transitions table.

Event name	Type	Arrival pattern	Time requirement	Response/transition
Keyboard reading	Internal	Periodic, 7.82 ms	Hard, 7.82 ms	read key → read key → read key → read key → verify key
Turn LED on	Internal	Periodic, 500 ms	Soft, 100 ms	set on → enable register → set port
Turn LED off	Internal	Periodic, 500 ms	Soft, 150 ms	set off → disable register → set port
Scan channels	Internal	Periodic, 0.122 ms	Hard, 0.122 ms	get scan ready → set squelch level → change channel
Send message to handset	Internal	Periodic, 1000 ms	Hard, 500 ms	search free channel → set head → set data
Check line	Internal	Periodic, 0.122 ms	Hard, 0.01 ms	check line
Incoming call	External	Periodic, 700 ms	Soft, 700 ms	check line → (search free channel → set head → set data → read header → read data → verify channel → verify security code) → (set on → enable register → set port) ‖ programming speaker registers
Change security code	External	Periodic, 600 ms	Soft, 400 ms	New Code → (Modify value → (Save EEPROM ‖ (set header → set data)))
Outgoing call	External	Periodic, 1100 ms	Soft, 1100 ms	read header → read data → ask for line → search free channel → set head → set data → read header → read data → (programming line ‖ (set on → enable register → set port))
Intercommunication	External	Periodic, 1200 ms	Soft, 1200 ms	get intercom ready → ((search free channel → set header → set data → read head → read data → verify channel → verify security code) ‖ (set on → enable register → set port))

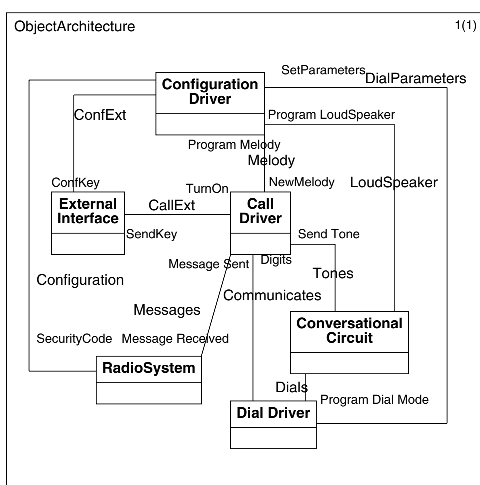


FIGURE 21. Object design.

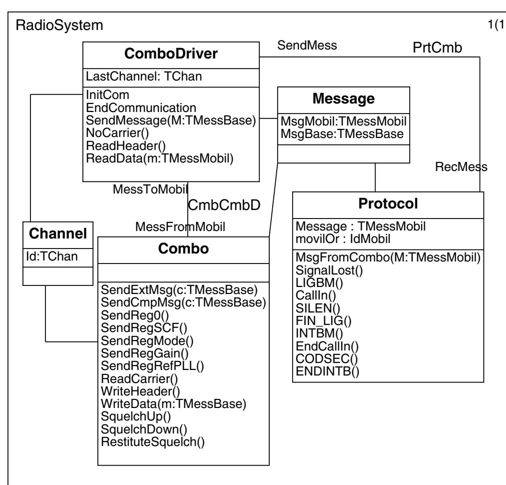


FIGURE 22. Class combo design.

Downloaded from http://comjnl.oxfordjournals.org/ at Pennsylvania State University on September 15, 2016

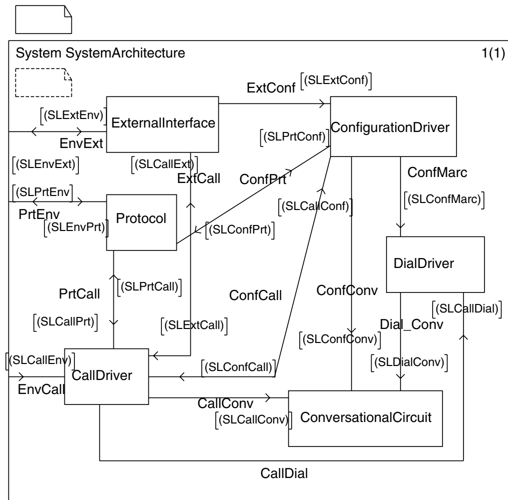


FIGURE 23. SDL system specification.

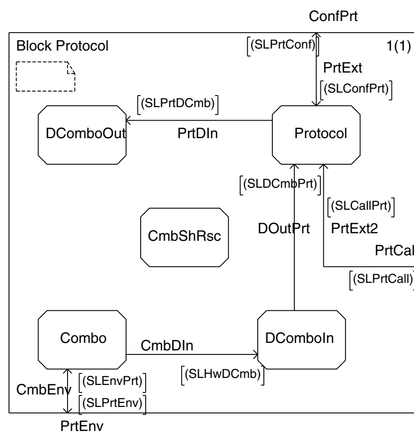
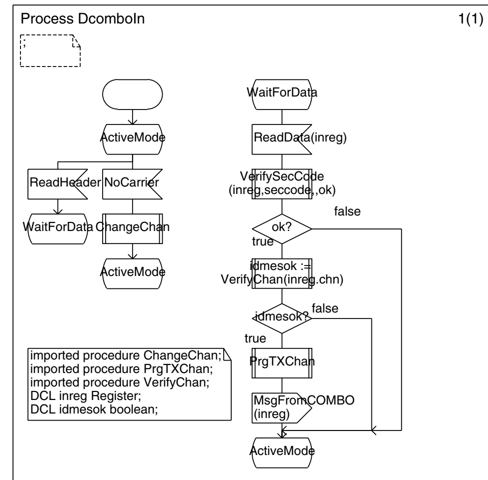


FIGURE 24. Block protocol specification.



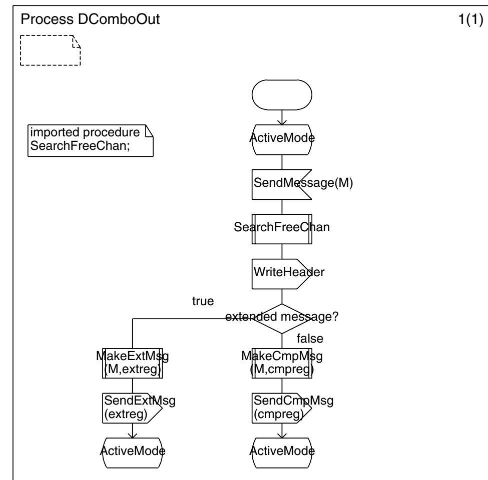


FIGURE 25. Resulting processes from object Driver Combo.

on in Section 3.2.3. This study can make changes in the refined logical architecture as is shown in Figure 24. As stated in Sections 2.3.2, 2.3.3 and 3.2, the process `Combo` is the passive process modelling the hardware device and its features will be accessed by means of RPC calls. Every interaction will be through the active process corresponding to its driver. However, we have to split the driver process in two (`DComboin` and `DComboOut`) since it deals with parallel events (simultaneous message sending and reception) and this situation introduces a blocking time making the system non-schedulable. As a consequence of this, we also have to add a passive process to encapsulate part of the object internal state as a shared resource (`CmbShRsc`). These shared resources are the table of channels and reception and transmission channels. The arrival of a message from a handset is dealt with an interruption, so we had to add a signal route between `Combo` and `DComboIn` (see Figures 24 and 25).

One of the most important tasks in developing the non-functional aspects of the design phase is the study of the internal events and the realization of the event/transition table explained in Section 3.2.3. We distinguish different

TABLE 3. Events for idle, incoming call and outgoing call operating modes.

Mode	Event name
Idle	Keyboard reading
	Turn LED on
	Turn LED off
	Scan channels
	Check line
Incoming Call	Keyboard reading
	Turn LED on
	Turn LED off
	Incoming call
Outgoing Call	Keyboard reading
	Turn LED on
	Turn LED off
	Outgoing call
	Check line

Downloaded from http://comjnl.oxfordjournals.org/ at Pennsylvania State University on September 15, 2016

TABLE 4. Transition table.

Action ID	Blocking delay	Atomic	Time used	Priority
Read key	n/a	Y	0.010 ms	50
Verify key	n/a	Y	0.008 ms	50
Set off	0.083 ms	Y	0.001 ms	40
Disable register	n/a	Y	0.017 ms	40
Set on	n/a	Y	0.001 ms	39
Enable register	n/a	Y	0.017 ms	39
Set port	n/a	Y	0.055 ms	39
Get scan ready	n/a	Y	0.001 ms	55
Set squelch level	n/a	N	0.065 ms	55
Change channel	n/a	N	0.050 ms	55
Search free channel	230 ms	N	1.770 ms	38
Set header	n/a	N	0.018 ms	38
Set data	n/a	N	0.028 ms	38
Read data	n/a	Y	230 ms	37
Read header	n/a	Y	280 ms	37
Verify channel	n/a	N	0.133 ms	37
Verify security code	n/a	N	0.400 ms	37
New code	n/a	N	0.004 ms	33
Modify value	n/a	N	0.001 ms	33
Save EEPROM	n/a	Y	120 ms	33
Check line	n/a	Y	0.002 ms	60
Programming speaker registers	n/a	N	0.040 ms	32
Ask for line	n/a	N	0.247 ms	30
Programming line	n/a	N	0.040 ms	30
Get intercom ready	n/a	Y	0.001 ms	29

operating modes and the analysis is carried out in the context of each of these operating modes. In Table 2 we can see the event/transitions table which contains all the events in our simplified application. The numbers related to the time are requirements of the system and they depend on the microprocessor where the software is installed. These times were supplied by the Alcatel company. All the actions that respond to an event are the corresponding transitions in the SDL design. For example the action `ReadHeader` in event `incoming call` is transition `ReadHeader` in the SDL process of Figure 25.

We have considered three operating modes: idle, incoming call and outgoing call. The concurrent events implied in the functional modes are shown in Table 3.

Table 4 contains all the information necessary to apply the timing techniques explained in Section 2.4.

In the simulation and validation phase we made simulation models to prepare the application to solve the errors in the developed system. The validation models helped to complete the system by detecting design failures such as deadlocks and implicit signal consumption using bit-state algorithms. Also, we verified our system by developing MSC use cases to compare with the results of the simulation of the system. These actions are commented on in Section 3. In the non-functional aspects, we obtained the preliminary real-time analysis by using the event/transition tables with the different operating modes from the design phase. Table 5 shows the preliminary real time analysis for idle, incoming call and outgoing call operating modes.

TABLE 5. Response time for operating mode actions.

Mode	Event name	Deadline	Response time
Idle	Keyboard reading	7.82 ms	5.85 ms
	Turn LED on	100 ms	76.37 ms
	Turn LED off	150 ms	77.59 ms
	Scan channels	0.122 ms	0.121 ms
	Check line	0.01ms	0.005 ms
Incoming call	Keyboard reading	7.82 ms	0.053 ms
	Turn LED on	100 ms	0.19 ms
	Turn LED off	150 ms	0.20 ms
	Incoming call	700 ms	515.92 ms
Outgoing call	Turn LED on	100 ms	0.156 ms
	Turn LED off	150 ms	0.166 ms
	Check line	0.01ms	0.005 ms
	Outgoing call	1100 ms	1067.37 ms

5. CONCLUSIONS AND FUTURE WORK

The contributions of this paper could be summarized in four main aspects. First, rate monotonic analysis is integrated in the SDL development cycle, paying special attention to

Downloaded from http://comjnl.oxfordjournals.org/ at Pennsylvania State University on September 15, 2016

the transition between the object and the process models. This integration is helpful in filling the gap existing between two models, which is particularly important in embedded real-time systems. Second, a new predictable execution model for SDL has been presented. This model is based on transition priority assignment and priority inheritance and considers subjects such as precedence constraints, priority inversion and hardware integration. Third, all these aspects are combined to propose an object-oriented methodology for embedded real-time systems, detecting the activities to be carried out in each of the four phases. Finally, to illustrate the most relevant details of our proposal, a case study is analysed, corresponding to the software development for a cordless telephone application.

The conclusions presented in this work are the result of particular experiences of the authors in developing embedded real-time software. As a future work, advantages and inconveniences of applying this methodology to a wider range of problems should be evaluated. The construction of automatic tools to support all the proposed techniques would also be interesting. To do this, there are two possibilities: to extend existing tools, which already support the functional aspects, or construct new ones. A prototype of a run-time system support for a subset of SDL has already been developed. It implements the real-time execution model described in Section 2.3. Currently, we are working to use our methodology in designs based on state charts.

ACKNOWLEDGEMENTS

This work has been partially funded by CICYT projects TIC99-1083-C02-01 and TIC98-0445-C03-03. We would like to thank the anonymous referees for their insightful comments and suggestions, which greatly helped us to improve the quality of the paper.

REFERENCES

[1] Terrier, F. and Barroca, L. (1997) Object technology and real-time: problematic and trends. In *Proc. ECOOP97 Workshop Reader*, Jyäskylä, Finland, June 9–13, pp. 417–433. Springer, Heidelberg.

[2] ITU (1994) *Specification and Description Language (SDL)*. ITU Recommendation Z.100. Geneva, Switzerland.

[3] SDT 3.4 Manuals (1998).

[4] Booch, G., Rumbaugh, J. and Jacobsson, I. (1997) Unified modeling language. *Notation Guide Version 1.0*. Rational Software Corporation.

[5] ITU (1996) *Message Sequence Chart (MSC)*. ITU Recommendation Z.120. Geneva, Switzerland.

[6] Klein, M. H. *et al*. (1993) *A Practitioners Handbook for Real-time Analysis*. Kluwer, London.

[7] Burns, A. and Wellings, A. (1997) *Real-Time Systems and Programming Languages*. Addison-Wesley.

[8] Liu, C. L. and Layland, J. W. (1973) Scheduling algorithms for multiprogramming in a hard real-time environment. *J. ACM*, **20**, 46–71.

[9] Joseph, M. and Pandya, P. (1986) Finding response time in a real-time system. *Comp. J.*, **29**, 390–395.

[10] González, M., Klein, M. and Lehoczky, J. (1991) Fixed priority scheduling of periodic tasks with varying execution priorities. In *Proc. IEEE Real Time Systems Symp.*, San Antonio, CA, December 4–6, pp. 116–128. IEEE Computer Society Press.

[11] Mathai, J. (1996) *Real-time Systems. Specification, Verification and Analysis*. Prentice-Hall, Hertfordshire.

[12] Alvarez, J. M., Diaz, M., Llopis, L., Pimentel, E. and Troya, J. M. (1999) Integrating schedulability analysis and SDL in an object-oriented methodology. In *Proc. 9th SDL Forum*, Montreal, Canada, June 21–25, pp. 241–259. Elsevier, Amsterdam.

[13] Burns, A. and Wellings, A. (1994) HRT-HOOD: A design method for hard real-time systems. *Real Time Sys.*, **6**, 73–114.

[14] Awad, M. *et al.* (1996) *Object Oriented Technology for Real-Time Systems*. Prentice-Hall, Upper Saddle River, NJ.

[15] Powel, B. (1999) *Real-Time UML*. Addison-Wesley, Reading, MA.

[16] Selic, B., Gullekson, G. and Ward, P. T. (1994). *Real-Time Object-Oriented Modelling*. Wiley.

[17] http://www.artisansw.com (2001)

[18] http://www.ilogix.com (2001)

[19] http://www.rational.com (2001)

[20] http://www.csverilog.com (2001)

[21] Kolloch, T. and Färber, G. (1998) Mapping an embedded hard real time systems SDL specification to an analysable task network—a case study. In *Proc. ACM SIGPLAN Workshop on Languages, Compilers and Tools for Embedded Systems*, Montreal, Canada, June, pp. 156–166. Springer, Berlin.

[22] Mitschele-Thiel, A. and Müller-Clostermann, B. (1997) Performance engineering of SDL/MSC systems. *Tutorial of the 8th SDL Forum*, Evry, France, September 23–26. Elsevier, Amsterdam.

[23] Dulz, W., Grughl, S., Kerber, L. and Söllner, M. (1999) Early performance prediction of SDL/MSC specified systems by automatic synthetic code generation. In *Proc. 9th SDL Forum*, Montreal, June 21–25, pp. 457–473. Elsevier, Amsterdam.

[24] Saksena, M., Freedman, P. and Rodziewick, P. (1997) Guidelines for automated implementation of executable object oriented models for real-time embedded control systems. In *Proc. 18th IEEE Real-Time System Symp.*, San Francisco, CA, December 2–5, pp. 240–251. IEEE Computer Society Press.

[25] Saksena, M., Ptak, A., Freedman, P. and Rodziewick, P. (1998) Schedulability analysis for automated implementations of real-time object-oriented methods. In *Proc. 19th IEEE Real-Time System Symp. (RTSS98)*, Madrid, Spain, December 2–4, pp. 92–102. IEEE Computer Society Press.

[26] Saksena, M., Karvelas, P. and Wang, Y. (2000) Automatic synthesis of multi-tasking implementations from real-time object oriented models. In *Proc. Int. Symp. on Object-Oriented Real-Time Distributed Computing (ISORC00)*, San Francisco, CA, March 15–17, pp. 360–368. IEEE Computer Society Press.

[27] Saksena, M. and Karvelas, P. (2000) Designing for schedulability: integrating schedulability analysis with object-oriented design. In *Proc. 12th IEEE Euromicro Conf. on Real-Time Systems*, Stockholm, Sweden, June 19–21, pp. 101–109. IEEE Computer Society Press.

[28] Münzenberger, R., Slomka, F., Dörfel, M. and Hofmann, R. (2001) A general approach for the specification of real-time

Downloaded from http://comjnl.oxfordjournals.org/ at Pennsylvania State University on September 15, 2016

systems with SDL. In *Proc. 10th SDL Forum*, Copenhagen, Denmark, June 26–29, pp. 203–223. Springer, Heidelberg.

[29] Spitz, S., Slomka, F. and Dörfel, M. (1997) SDL*—an annotated specification language for engineering multimedia communication systems. In *Proc. 6th Open Workshop on High Speed Networks*, Stuttgart, October.

[30] Altisen, K., Goesler, G. and Sifakis, J. (2000) A methodology for the construction of scheduler systems. In *Proc. Formal Techniques in Real-Time and Fault-Tolerant Systems*, Pune, India, September 20–22. Springer, Heidelberg.

[31] Alvarez, J. M., Diaz, M., Llopis, L., Pimentel, E. and Troya, J. M. (2001) Deriving hard real time system implementations directly from SDL specifications. In *Proc. 9th International Symp. on Hardware/Software Codesign*, Copenhagen, Denmark, April 25–27, pp. 128–134. ACM Press, New York.

[32] Wheeler, D. (1996) *ADA95. The Lovelace Tutorial*. Springer, Alexandria.

[33] Alvarez, J. M., Diaz, M., Llopis, L., Pimentel, E. and Troya, J. M. (2000) Schedulability analysis in real-time embedded systems specified in SDL. In *Proc. 25th IFAC Workshop on Real-Time Programming*, Palma, Spain, May 17–19, pp. 117–123. Elsevier, Oxford.

[34] Palencia, J. C. and González, M. (1999) Exploiting precedence relations in the schedulability analysis of distributed real-time systems. In *Proc. 21st IEEE Real Time Systems Symp.*, Phoenix, AZ, December 1–3, pp. 328–339. IEEE Computer Society Press.

Downloaded from http://comjnl.oxfordjournals.org/ at Pennsylvania State University on September 15, 2016