SDL AND HARD REAL-TIME SYSTEMS: NEW DESIGN AND ANALYSIS TECHNIQUES

J. M. Alvarez, M. Díaz, L. M. Llopis, E. Pimentel, J.M. Troya
{alvarezp,mdr,luisll,ernesto,troya}@lcc.uma.es
Dpto. Lenguajes y Ciencias de la Computación
Universidad de Málaga
Campus de Teatinos - 29071
Málaga - SPAIN

Abstract

The usage of formal description techniques (FDT) has arisen as a promising way of dealing the increasing complexity of embedded real-time systems. However, FDT do not take into account non-functional aspects, as the time requirements, that are especially important in the context of this kind of systems. In this paper, a method to predict the timing behaviour of real-time systems designed with SDL is proposed. In addition, if the system does not meet the time requirements we propose a set of heuristics to redesign the system in order to meet the time constraints. These techniques will also help us to consider the real-time requirements in the first stages in the design. To illustrate our proposal, an example of a computerized numerical control (CNC) machine is presented.

Keywords: SDL, real-time embedded systems, schedulability analysis, redesign

1. INTRODUCTION

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

However a great part of the complexity of these real-time systems comes not from the functional requirements but from the non-functional ones. These systems have to meet some non-functional requirements (safety, robustness and timeliness) that make them more difficult to develop than other systems. Most of these systems are controllers that have to respond in a certain period of time. A lost deadline can mean a wrong computation and a system failure.

The usage of formal description techniques (FDT) has arisen as a promising way to deal with the increasing complexity of real-time systems. These techniques were originally developed for the design of telecommunications systems, and for this reason, they were designed to cope with specific characteristics like concurrency, reactivity, etc that are common to real-time systems. One of the most widely extended FDT is the Specification and Description Language (SDL), which is an ITU standard [1] and it is currently well supported by commercial tools like SDT [2]. SDL is based on Extended Finite State Communicating Machines and it can be used through all the development cycle.

However, SDL presents difficulties, common to other message-based models, to express real-time constraints and to prevent real-time anomalies. In [3][4] these problems were addressed and some solutions were given, defining a predictable execution model for embedded real-time systems designed with SDL. A complete object-oriented methodology for embedded real-time systems is proposed in [5].

On the other hand, Rate-Monotonic Analysis (RMA) [6] provides a collection of quantitative methods that enable the analysis and prediction of the timing behaviour of real-time systems. This analysis can help us to organise processes and resources in our design to make possible to predict the timing behaviour of the final system.

In this paper, we propose to integrate RMA in the SDL design of real-time applications. An initial study for this integration was proposed in [7]. Based on these previous works and [13], we propose a more complete analysis. However, the analysis must help not only to predict the timing behaviour but, also, to think in redesign solutions in the system to get that it meets the timing requirements. In order to achieve it and based on our experiences, we present a set of techniques to redesign the system in order to meet the imposed deadlines.

This paper is organized as follows. In section 2 we survey the real-time execution model for SDL to integrate schedulability analysis. In section 3 the real-time analysis for systems specified in SDL is proposed. In section 4 a set of heuristics to redesign a SDL system in order to meet the time requirements of the events in the system are proposed. To illustrate our proposals, an example of a computerized numerical control (CNC) machine to produce workpieces designed by the user is presented in section 5. Last section presents some conclusions and future work.

1.1. Related Work

There exist several interesting proposals related to our work. For example, [9] designs real-time systems with UML methodology including timing constraints in the different phases of the development. It includes patterns and frameworks as help in the development of this kind of systems but it does not allow any kind of timing analysis.

In other interesting work, [10], it is shown how real-time scheduling theory may be applied to ROOM [11] models and they provide certain guidelines for the efficient execution. These works are very useful but the execution model is different and they do not study the systems to meet the response times. Also [12] shows how schedulability analysis can be integrated with object-oriented design developed using UML-RT[8] [9].

In [13] a schedulability analysis of tasks with precedence relations in distributed systems is presented. This analysis is very useful but it is necessary to adapt it to the SDL characteristics in order to get an accurate result in the calculation of the timing behaviour of the system.

Some important works have been developed in performance engineering with SDL. For example, in [15] a new approach for early performance prediction based on MSC specified systems in the context of SDL is presented. Also [16] presents a framework that demonstrates the relationship between formally specified SDL systems and appropriate performance analysis. In [17][18] extensions to describe timing constraints are proposed.

2. AN ANALYZABLE EXECUTION MODEL

In this section, the definition of the execution model for systems specified with SDL presented in [5] is summarized. This model has to be considered in order to get the integration of the schedulability analysis.

The execution model is based on fixed priority preemptive scheduling, however we do not assign fixed priorities directly to processes but to process transitions. The process priorities can vary from one state to another depending on the transitions that can be carried 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 it is executing another transition, this 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 priorities. In order to show how the process priorities are calculated we first define the following sets and functions:

- `Process`, `States` and `Signals`: the set of system processes, process states and signals.
- The function `sig` defined as

  $$\texttt{sig: Process x States} \rightarrow \wp(\texttt{Signals x N})$$

  that returns pairs (`signal,priority`) indicating the signals that can be received and the priority assigned to the transition associated to that signal.
- The function `received` defined as

  $$\texttt{received: Process} \rightarrow \texttt{Signals}$$

  `received(P)` returns the signals that are currently in process `P` queue.

Priorities are calculated as follows:

- Initially, every process has a default priority or the priority assigned to the its `START` transition.
- Every time signal `s` is received in state `e` by process `P`, the priority changes according with the following rule:

  $$pri(P) := \begin{cases} max\{pri(p), p\} & if\,(s,p) \in sig(P,e) \\ pri(P) & otherwise \end{cases}$$

  The new priority is the maximum between the current priority and the priority of the transition enabled by the signal reception, but only if that signal can be accepted in the current state. If the process was inactive, it can become active interrupting the current executing process, and if it was active it continues its execution with the new priority. In this case, this priority change can be considered a kind of priority inheritance between the transition currently under execution and the next transition to be achieved by that process.
- When process `P` changes to state `e`:

  $$pri(P) := max\{p : (s,p) \in sig(P,e) \ and\ s \in received(P)\}$$

  Every time a process finishes a transition, its priority is recalculated for being equal to the priority of the higher priority enabled transition.

Assigning priorities in this way, we always execute the transition with the highest priority, except when that transition belongs to the current executing process. In this case, we have to consider the executing transition time as blocking time for the new high priority task. This blocking source is called *run to completion blocking*. However, this effect can be minimized during design time trying to avoid having a single process attending to possibly conflicting concurrent events

3. REAL-TIME ANALYSIS

In this section we illustrate how the timing behavior analysis of a complete system can be achieved. An initial study for the integration of the schedulability analysis in SDL was proposed in [5] and [7]. Based on these previous works and [13] we propose a more complete analysis.

3.1. 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 1 we can see two possible examples of transitions to carry out the schedulability analysis. Every transition in the SDL system is labelled with a transition priority and, in order to do the timing analysis, all the transitions will have associated a worst case execution time (WCET).

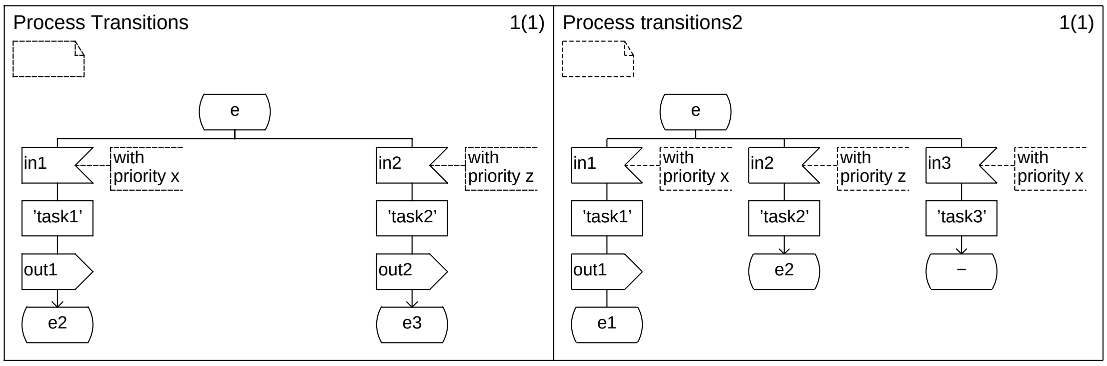

Figure 1. SDL Basic Set

We consider the response time of a 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 see the example of the figure 2, there are two external events which response sequence is composed by the transitions senv1, s_1, s_2 for the external event ev_1 and senv2, s_3, s_4 for the external event ev_2.

For instance, we obtain the response time of the event ev_1 by summing up the response times of the transitions activated by the signals senv1, s_1 and s_2.

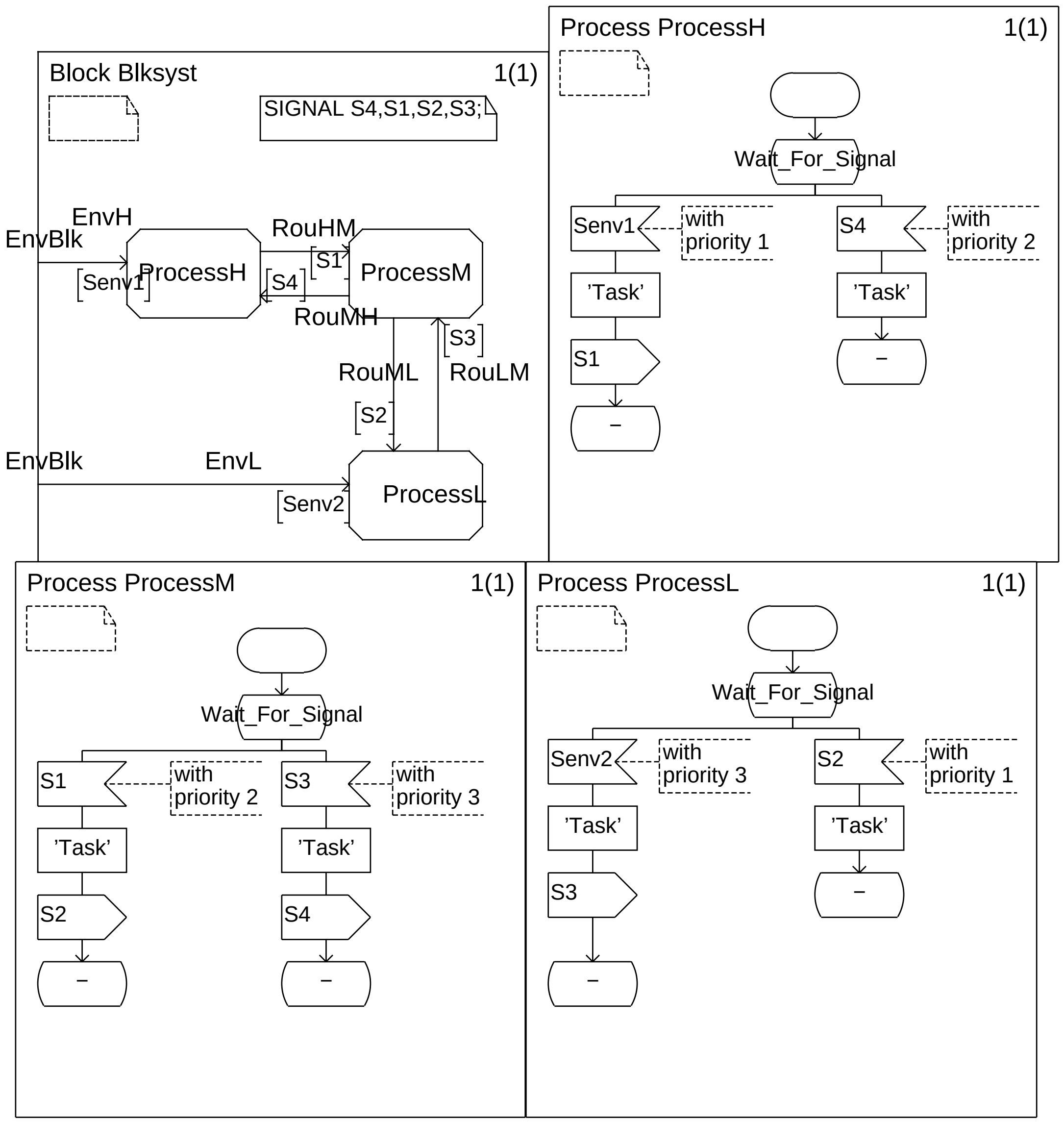

Figure 2. Response to external events

3.2. 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.

We can add the following aspects with respect to [13] to get more accurate results:

- There exist precedence relations between the transitions that respond to an external event. It can reduce the number of transitions that participate in the interference calculation.
- Also, the SDL semantics influences in the response time.

In the examples used to calculate the response times we have got improvements between 25% - 32% in the transitions that are affected by the above aspects.

In the following subsections we consider a set of external events $S_1,...,S_n$ and the sequence of transitions $t_{i1},...,t_{im}$ that respond to every external event S_i. All these transitions belong to the SDL processes in the system.

3.2.1. Precedence Relations.

Let us suppose an external event S_i composed by the sequence of transitions $t_{i1},...,t_{im}$ 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 will take into account all the transitions with higher or equal priority than transition t_{ab} that belongs to event S_i. However, the execution of the transitions in the event S_i, has an order in the execution due to the precedence relations between them. When the execution of transition t_{i1} finishes, then, transition t_{i2} begins and so 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 be interrupted only by sequence $t_{i1},...,t_{ij}$ or sequence $t_{ij+1},...,t_{in}$ but never by t_{ij}. We will have to analyse the sequence of transitions that give the worst case interference time.

Let us suppose now the situation shown in figure 3, where we want to calculate the interference of the event S_i with respect to transition t_{ab} that belongs to event S_a. The height of the boxes indicates the priorities of the transitions and the horizontal line indicates the priority of the transition to analyse. There exist two sequences of transitions that potentially can interrupt task t_{ab} but at most 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}.

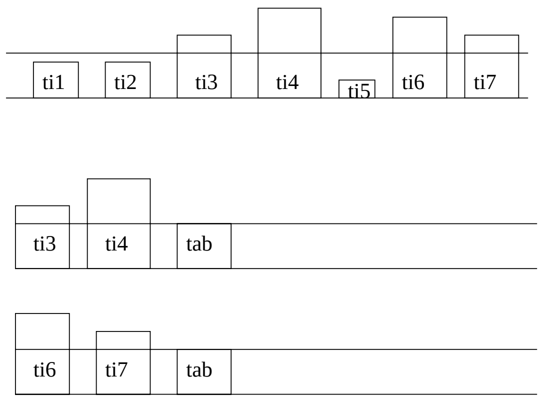

Figure 3. Interference for task t_{ab}

3.2.2. Including SDL Semantics

The SDL semantics does not allow that two transitions that belong to the same SDL process execute concurrently. Let us suppose that there exist 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 t_{ij} priority is greater than 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 transitions that can interrupt to others transitions but, as a consequence of this, we have to consider an additional blocking time called *run to completion blocking.*

In figure 4, we have an external event S_i and the sequence of transitions $t_{i1,}t_{i2},...,t_{i5}$ 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 5 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}.

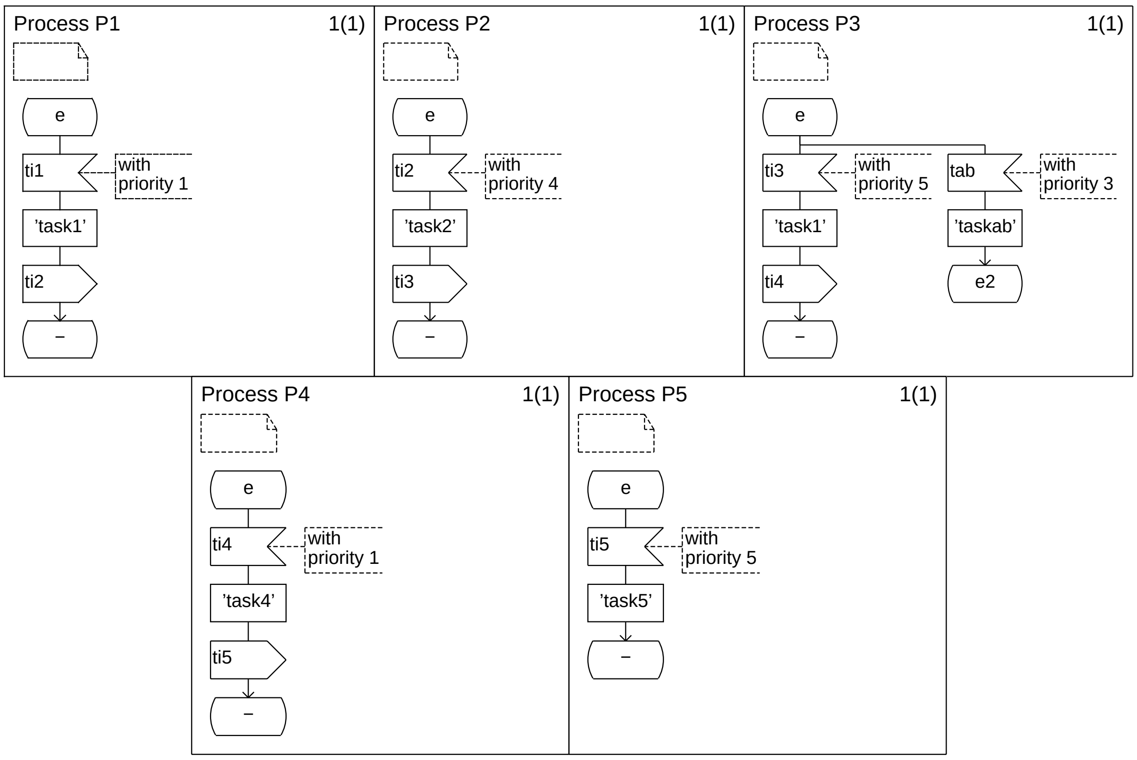

Figure 4. Calculating t_{ab} interference in SDL

Transitions t_{i3} and t_{ab} belong to the same SDL process so t_{i3} cannot preempt t_{ab}. If we take into account the precedence relations, 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} due to they share the same SDL process.

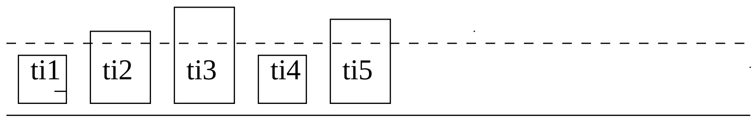

Figure 5. Relationship between t_{ab} priority and the task priorities of S_i

Although the previous situation can reduce the number of transitions that can interrupt to other transitions the blocking time can be increased 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 6, t_{i1} and t_{ab} belong to the same SDL process and t_{i1} priority is less than t_{ab} priority. 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 *run to completion blocking* time of transition t_{ab}.

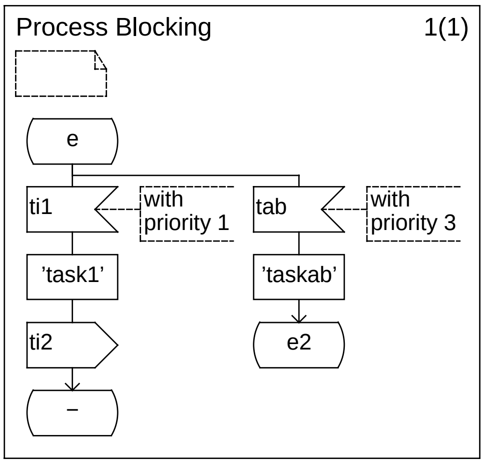

Figure 6. Calculating t_{ab} blocking time.

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

3.2.3. Worst case interference

Let us consider `Transitions` and `Process` as the set of transitions in the system and the set of SDL processes. The function p_{sdl} is defined as

$$P_{sdl}:\text{Transitions} \rightarrow \text{Process}$$

and it returns the SDL process that this transition belongs to.

The set `hp(t,S)` is defined as:

```
hp(t,S)={t′ :pri(t′)>=pri(t) and t′ takes part in the response to S}
```

In order to calculate the worst case interference time of transition t_{ab} we do the following steps:

- Select the consecutive transition sequences that belong to `hp` and do not share the same SDL process than t_{ab} for each event in the system. Several sequences can appear in every event.
- If there are several sequences per event we select the worst execution time sequence.
- Calculate the interference using the expression proposed in [13]. We call to this expression I_{ab}

3.2.4. Blocking time

In order to include the blocking time in the schedulability analysis we have to consider two possible blocking sources:

- The first blocking source is devoted to the access to shared resources. We encapsulate these shared resources in passive processes accessed by means of remote procedure calls (RPC). Using the highest locker protocol this blocking time is bounded. It is explained in depth in [5]. We call B_{sh} to this blocking source.

- The second possible source is due to the *run to completion blocking*, B_{rtc}.

The expression of B_{rtc} for transition t_{ab} is the following:

$$B_{rtc}=\max\{c_t : p_{sdl}(t) = p_{sdl}(t_{ab}) \text{ and } pri(t) < pri(t_{ab}) \text{ and } t <> t_{ab} \}$$

The blocking time is the maximum worst execution time (c_t) among the transitions that belong to the same SDL process and have a lower priority than the priority 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 like a possible blocking. In this case the blocking expression is:

$$B_{rtc}=\max\{c_t : p_{sdl}(t) = p_{sdl}((t_{ab}) \text{ and } pri(t) < pri(t_{ab})\}$$

The expression of the response time of transition t_{ab} , R_{ab}, that participates in the response of a external event S_a is the sum of worst case execution time of transition t, the interference of higher priority transitions and the maximum between the blocking sources:

$$R_{ab}=c_{ab} + I_{ab} + max\{B_{sh}, B_{rtc}\}$$

We do not include more details in the previous equation because it in out of the scope of this paper.

4. REDESIGNING THE SYSTEM

With this execution model we can analyse if a system meets their real-time constraints. However, this analysis cannot be achieved until late in the design process, since we need to know time characteristics as the worst case execution time and the blocking time of all the transitions in the system. In addition if the system does not meet its time requirements it has to be redesigned. We propose a set of heuristics that allow us:

- To redesign the system if it does not meet the deadlines. The changes in the design will be affect to the parameters in the response time equation (see section 3) like blocking and interference.
- To take into account the real-time requirements from the first stages of the design.

We propose four heuristics:

- Task transference. It reduces the interference with others transitions.
- Process creation. It reduces the blocking time.
- Transition replication. It also reduces the blocking time.
- Intermediate transition elimination. It eliminates transitions in the SDL design.

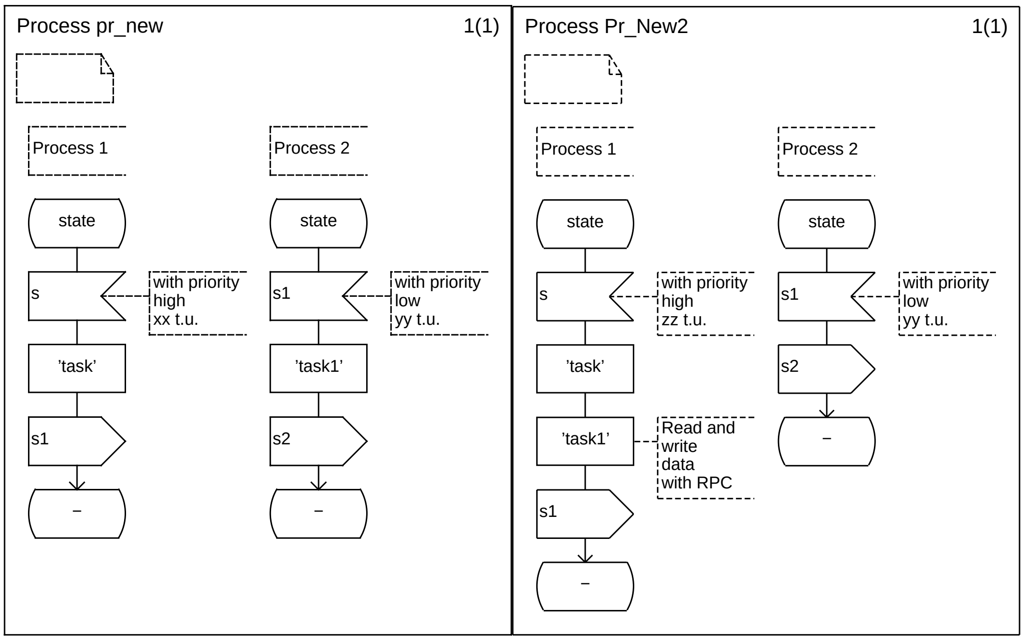

Figure 7. Task transference in SDL processes

4.1. Task Transference

This heuristic modifies the design in two phases:

- It looks for consecutive transitions that take part in the response to an event that does not meet its deadlines where the priority of the first one is higher than the priority of the second one or vice versa.
- It transfers the task from the lower priority transition to 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 others events it can do possible that 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 priority of t is higher than t', then we transfer the t' computation to t. Due to the SDL semantics, a SDL process cannot access to the variables of other processes, so, t' computation cannot access to its variables. We consider these variables like shared resources and they have to be accessed by means of RPC [5]. It can increase the response time since the blocking time (B_{sh}) of both processes can be affected. However, due to we use the priority ceiling protocol, the blocking time is bounded and we have to select the maximum possible blocking time between B_{rtc} and B_{sh} for all the transitions. In the most of the cases B_{rtc} is higher than B_{sh} and then the last blocking source does not affect to the response time.

As we can see in the figure 7, process 2 maintains the signal reception and sending but these operations are atomic. This transition will disappear by applying *Intermediate Transition Elimination.*

4.2. Creating SDL Processes

Many processes in a design of a system with SDL can be designed as in figure 8 where there exists a state in a process with more than one signal reception and we consider that both transitions would be able to execute concurrently. If we see the figure, the process activation can occurs due to reception of *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 that is very important in this kind of systems. In the figure it is not possible to execute transition *send* until the completion of transition *receive* and vice versa. As commented in section 3 it produces a *run-to-completion blocking*. The solution is to create two different processes to send and receive data concurrently.

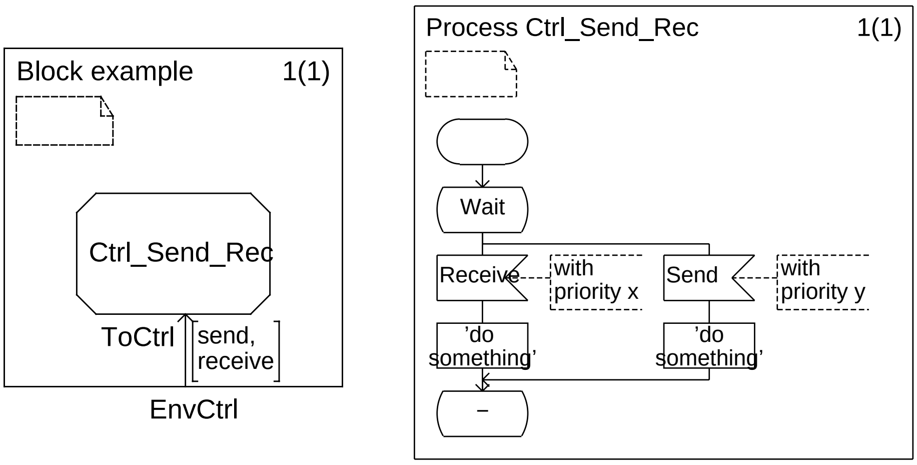

Figure 8. Initial design

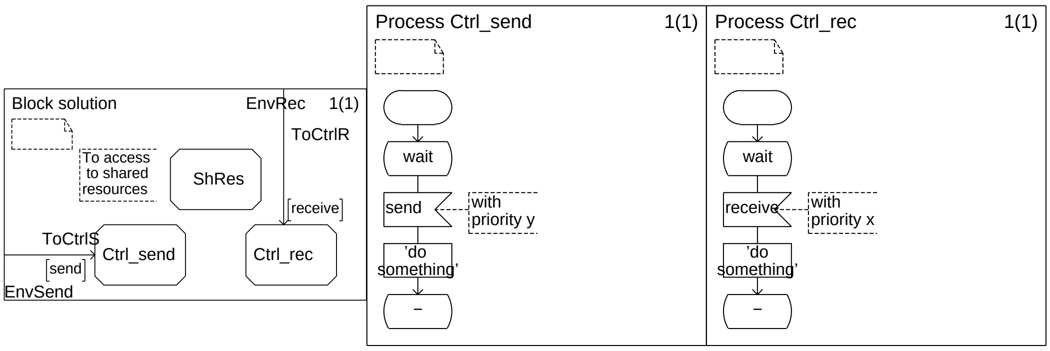

Figure 9. Final design

4.3. Transition Replication

This heuristic looks for more situations in the SDL system to reduce the blocking time of the transitions. Transitions of different events can execute common tasks so the designers will join these transitions to get a more efficient code. However, it can increase the blocking time. We will consider figure 10, where if the option is true the transition executes *task1* and *taks2* belonging to event *e* and the transition executes *task1* and *task3* belonging to event *e'* if the option is false. The WCET is 25 time units (t.u.) and B_{rtc} is 25 t.u (see section 3). The technique proposes to create two transitions: the first one is composed by *task1* and *task2* and the WCET is 25 t.u. but the B_{rtc} is 18 t.u. The second one is composed by *task1* and *task3* with WCET of 18 t.u. and B_{rtc} of 25 t.u.

4.4. Intermediate Transition Elimination

This design technique allows reducing the concurrency in the systems because it can eliminate processes in the design without affecting system responsiveness. It can be seen in figure 11. It basically consists of eliminating intermediate transitions that has not real computation to achieve. It is necessary to take the signal sendings of the eliminated transition to the predecessor one. This way, there are less transition 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.

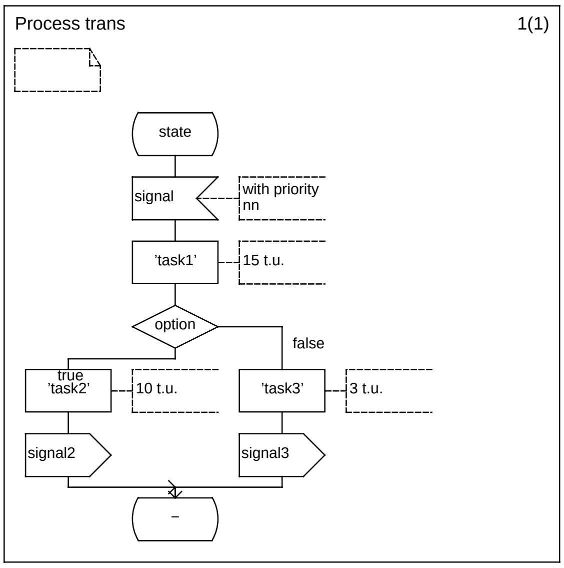

Figure 10. Shared Transitions

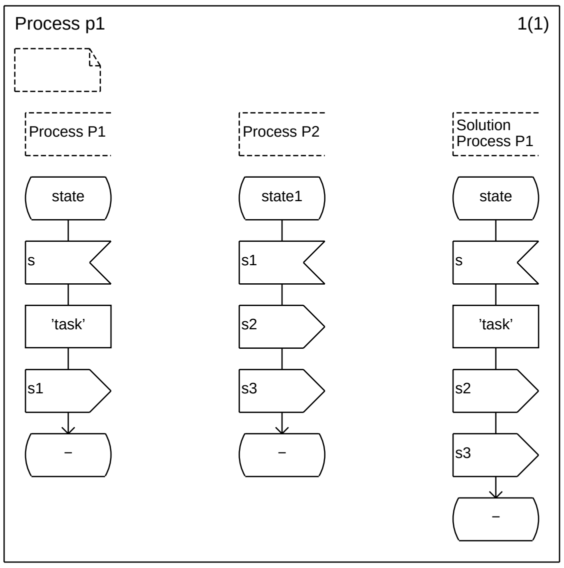

Figure 11. Intermediate Transition elimination

5. EXAMPLE

In this section we discuss an application example in order to clarify the utility of our proposals. The example is devoted to the design of computerized numerical control (CNC) machines[14]. A CNC machine is an automatic machining tool that is used to produce workpieces designed by user. It is equipped with a computer-based digital control system. We have designed this controller that should be able to position the cutter precisely and automatically along the reference trajectory of the machined workpiece. As we can see in figure 12, there are three periodic events in the controller, two of them to calculate the new position of the cutter and the third to estimate the disturbance and to monitor the plant. All of these events are periodic. A previous step to the SDL design is to describe the events in the system be means of MSC. Figure 13 is the block diagram of the system and figure 14 gives a high level description of the system processes. We have designed a passive process to encapsulate the position as a shared resource. The access to the shared variables will be by means of RPC as we proposed in [5]. The time requirements of each event in the system can be seen in table 1.

Event	Period	Deadline
Calculate cutter X position	2400	4000
Calculate cutter Y position	2400	4000
Calculate disturbance	2400	6000

Table 1. Time requirements in the system.

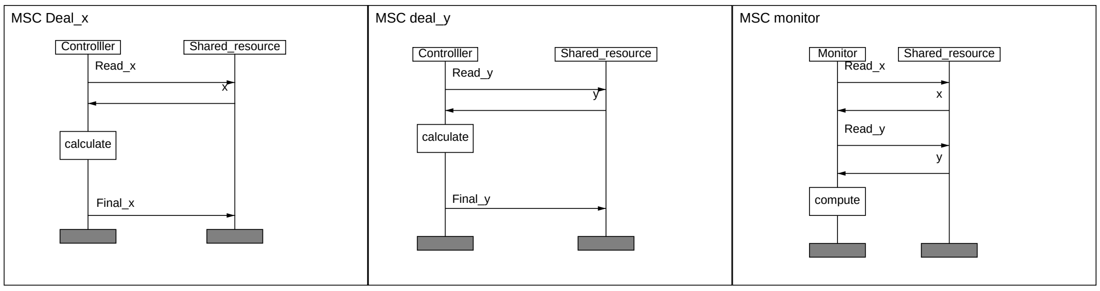

Figure 12. MSC event description

When timer *Tx* expires, process *Deal_Ref* reads the *x* position and does a computation, then a transition *x_ref* is activated in process *Control*. It updates the new *x* position and reports the new position in the monitor system with the signal sending *x_ctrl_mon*. The same action occurs with the timer *Ty* but for *y* position. The third periodic event activates the transitions of the processes *Deal_pos, Calc_vel, Monitor, Mon_state_Plant* when timer *Tpos* expires.

All the transitions in the system have associated a priority and a worst execution time to be able to do the schedulability analysis proposed in section 3.

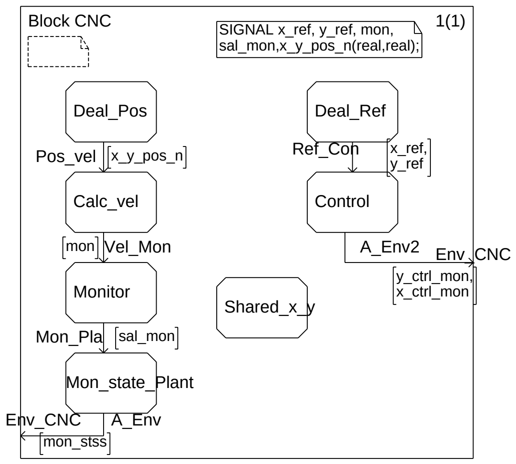

Figure 13. Block diagram

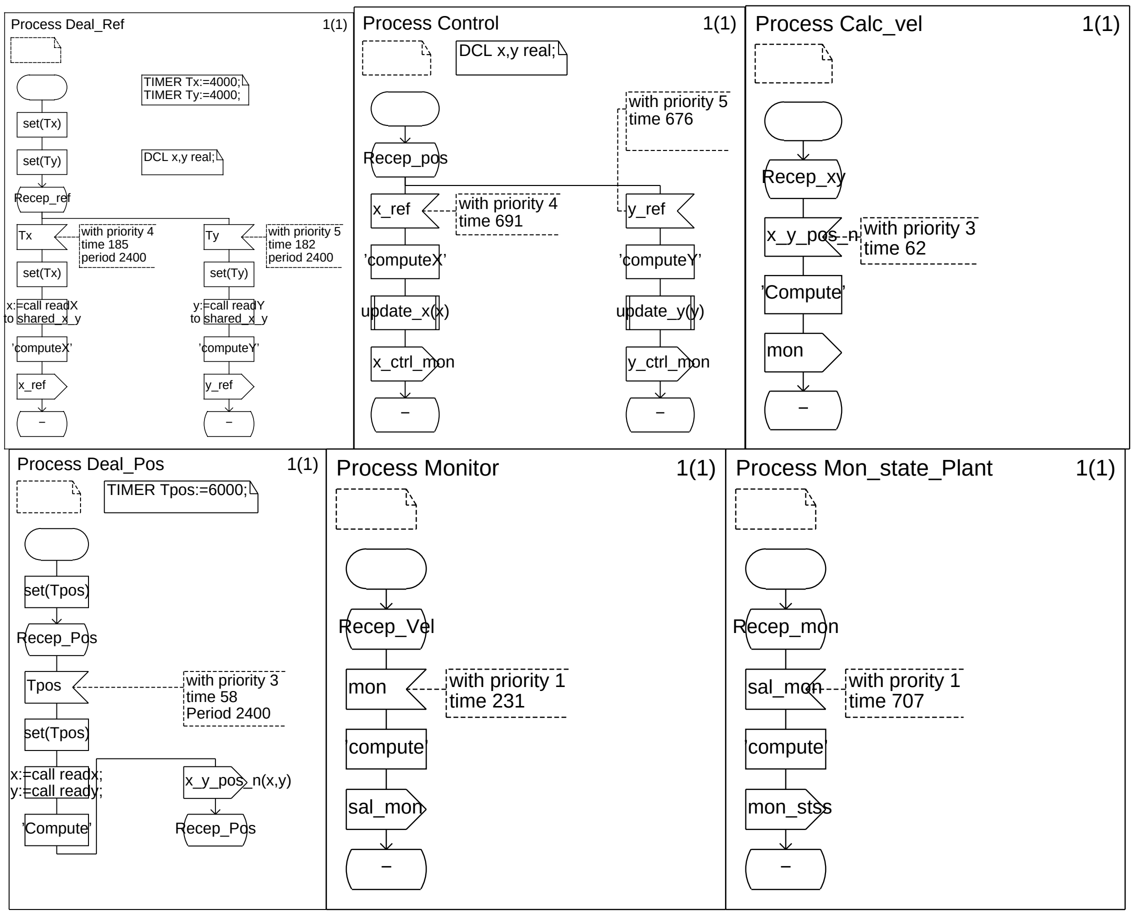

Figure 14. CNC controller design

As we can see in figure 14 transitions *x_ref* and *y_ref* belong to the same SDL process. In table 2 we have applied the proposals of the section 3 and we have not included transition *y_ref* in the interference of transition *x_ref* although the former has a higher priority. Also, we have included the run-to-completion blocking in the calculations of the response time.

	Response time	**Deadline**
Calculate cutter x position	1734 µs	4000 µs
Calculate cutter y position	1734 µs	4000 µs
Calculate disturbance	7994 µs	6000 µs

Table 2. Event Response Times

The event that calculates the disturbance in the system does not meet the deadlines. We can apply some of the redesign heuristics presented in the previous section to try that the system meets the deadlines. For example, using the *task transference* heuristic we can take the task of the transition of process *Monitor* to the transition of process *Calc_vel* to execute it with higher priority and reduce the interference. Also, we apply the *intermediate transition elimination* heuristic to eliminate the transition of process *Monitor* once it only has the signal reception and the signal sending. With these changes, we reduce the number of process in the system. The new response times can be seen in the table 3. In figure 15 we can see the final block diagram and the new SDL process *Calc_Vel*.

	Response time	**Deadline**
Calculate cutter x position	1734 µs	4000 µs
Calculate cutter y position	1734 µs	4000 µs
Calculate disturbance	5860 µs	6000 µs

Table 3. Event Response Times after redesigning

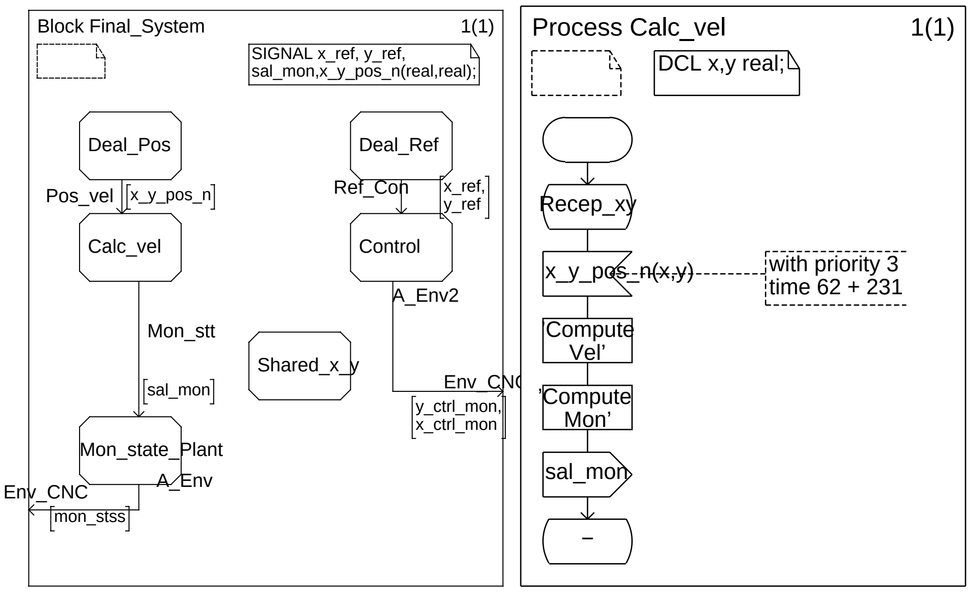

Figure 15. Final block diagram and process Calc_Vel

6. CONCLUSIONS AND FUTURE WORK

In this paper, we have presented a method to integrate the schedulability analysis in systems specified in SDL. In addition, we have introduced a set of techniques to redesign the SDL system in order to all the events in the system meet their deadlines. Also, these heuristics will help us to take into account the time requirements in the first stages of the design.

As future work, we are developing applications and the necessary interfaces to integrate our proposals in the commercial tools. We are implementing a microkernel similar to Cmicro[2] that executes our real-time model of SDL joined to monitoring tools to calculate worst case execution times and time failures detection in the execution of SDL systems.

7. REFERENCES

[1] ITU recommendation Z. 100. "Specification and Description Language (SDL)". 1994.
[2] Telelogic "SDT 3.5 Manuals". 1999.
[3] Alvarez J..M., Diaz M., Llopis L., Pimentel E., Troya J.M. "An Analyzable Execution Model for SDL for Embedded Real-Time Systems". Workshop on Real-Time Programming. Elsevier. 1999.

[4] Alvarez J..M., Diaz M., Llopis L., Pimentel E., Troya J.M. “Embedded Real-Time Systems Development Using SDL”. IEEE Real-Time System Symposium . 1999.

[5] Alvarez J..M., Diaz M., Llopis L., Pimentel E., Troya J.M. “Integrating Schedulability Analysis and SDL in an Object Oriented Methodology for Embedded Real-Time Systems”. SDL Forum. 1999.

[6] Klein, M.H. et al. “A Practitioner’s Handbook for Real-time Analysis”. Kluwer Academic Publishers. 1993.

[7] Alvarez J..M., Diaz M., Llopis L., Pimentel E., Troya J.M. “Integrating Schedulability Analysis in Real-Time Systems Specified in SDL”. To appear in Workshop on Real-Time Programming. 2000.

[8] Douglass Powel. "Real-Time UML. Developing Efficient Objects for Embedded Systems". Addison-Wesley. 1998

[9] Douglas Powel. "Doing Hard Time. Developing Real-Time Systems with UML, Objects, Frameworks and Patterns". Addison -Wesley. 1999

[10] Saksena M, Ptak A., Rodziewicz P. "Schedulability Analysis for Automated Implementations of Real-Time Object-Oriented Models". IEEE Real-Time System Symposium. 1998.

[11] Sellic B. et al. “Real-Time Object-Oriented Modeling”. John Wiley Publisher. 1994.

[12] Saksena M., Karvelas P. "Designing for Schedulability Integrating Schedulability Analysis with Object-Oriented Design". To appear in Euromicro on Real-Time Systems. 2000

[13] Palencia J.C., Gonzalez M, "Exploiting Precedence Relations in the Schedulability Analysis of Distributed Real-Time Systems". IEEE Real-Time Systems Symposium. 1999.

[14] Kim N., Ryu M. et al. "Experimental Assessment of the Period Calibration Method: A Case Study" Real-Time Systems Journal. Kluwer Academic Publishers. 1999

[15] Dulz W. , Grughl S., Kerber L., Söllner M. “Early Performance Prediciton of SDL/MSC Specified Systems by Automatic Synthetic Code Generation”. 9th SDL FORUM. Elsevier. 1999.

[16] Dulz W. “Performance Evaluation of SDL/MSC-Specified System”.ESM96 European Simulation Multiconference. 1996

[17] Spitz S. Et al. “SDL*- An annotated Specification Language for Engineering Multimedia Communications Systems”. 6th Open Workshop on High Speed Networks. 1997.

[18] Faltin N. et al. "An annotational Extension of MSC to support Performance Enginnering " 7th SDL FORUM 1997