Modeling event-driven real-time applications using DAGs *

Enrico Bini
Scuola Superiore S. Anna
Pisa, Italy
e.bini@sssup.it

Giuseppe Lipari
Scuola Superiore S. Anna
Pisa, Italy
lipari@sssup.it

Carlo Vitucci
Ericsson Lab Italy
Roma, Italy
carlo.vitucci@eri.ericsson.se

Abstract

In this paper, we present a novel formalism for modeling event-driven real-time applications based on Directed Acyclic Graphs (DAG). We explain how it is possible to express several kinds of software constructs with our model, and how these constructs can be mapped to mathematical equations. The goal is to express a real-time schedulability problem as a operation research problem. In the future we plan to develop novel optimization algorithms for solving this problem.

1 Introduction

Most of the research on real-time systems has been focused on schedulability analysis of periodic and sporadic task sets [4] [3] [2]. The periodic task abstraction is very useful when modeling real-time control applications, where sensors have to be read at a given sampling rate and actuators have to be driven periodically.

However, there is an entire class of applications for which the periodic task model is not completely adequate. We generically refer to these applications as *event-driven*, as opposed to *time-driven* applications. In these applications, tasks usually communicate by means of signals and they are activated by external interrupts or by user commands. Given such application, it is quite natural to come up with a directed acyclic graph (DAG) model: tasks are represented by nodes in the graph, and arrows are message between tasks. In this model, some interrupt can also be periodically activated: however, the temporal constraints are not on the single task, but on a given *flow* of data. This kind of constraints are also called "end-to-end" constraints. Other kind of constraints can arise from the underlying operating system support. For example, if the message buffer between two tasks is limited, we have also to impose that no signal will be lost because of a buffer overflow. Some model has already been presented in the literature [10] [8] [9] [7].

Many applications that can be found in the telecommunication world are modeled as event-driven applications. The typical design cycle in such context is done using formal languages and design tools which automatically generate the application code. However, such tools do not take into account real-time constraints explicitly, and it is not clear how the application functionality is mapped onto real-time process. For example, a completely ignored aspect is how to assign task priorities [5].

To better understand the real-time nature of event-driven application, we think that it is essential to be able to answer the following questions:

1. how much is the maximum frequency of the external stimula without losing any signal?
2. can we improve it by changing the tasks priorities?
3. how much does this response time increase if the processor is loaded additionally?

The main goal of our research is so to analyze the behavior of the system, and then to assign the tasks parameters (such as priorities and partitioning on multiprocessor) which optimize it. As we will show in more detail further, this optimization process will require:

- a formal modelization of the system;
- the definition of a cost function;
- an optimization algorithm to find the best solution.

In this paper, we present a novel graph model that is able to describe the interesting real-time properties of an event-driven application. The next step will be researching a good algorithm to find this optimum point.

2 Model overview

The first assumption we do in our model is that the application can be divided into a set of **non-preemptable** atomic tasks. This assumption can

seem too restrictive. Nevertheless, in the embedded software application for telecomunication handling, it is common and often a real design rule, to avoid resource allocation more than necessary, particularly for hardware interrupt process handling. Moreover, this is just a first approach to this problem; in the future we plan to extend our model to preemptive scheduling.

The model which best fits an event-driven application is undoubtedly a graph, because it can well represent the flow of signal from task to task in the application. In this graph, each node is associated to a task and each arrow is associated to a signal. Every task is usually labeled with τ_i, its worst-case execution time (WCET) is C_i and the first instant of execution is t_i. Because of the non-preemptive hypothesis, task τ_i ends its computation at $t_i + C_i$.

Signals between tasks are expressed by mathematical constraints. If, for example, the task τ_2 is activated by the task τ_1, we will write $\tau_1 \prec \tau_2$ and the implied constraint will be $t_2 \geq t_1 + C_1$. In Section 3, the possible relationships between tasks are discussed in great detail.

Our optimization process is performed on the space of all the possible schedules. A schedule is represented by the set of the starting times for all tasks. Formally we define a schedule as $\{t \in \mathbf{R}^+ \mid \exists\ \tau_i,\ t_i = t\}$. In all the rest of the paper, we will refer to this set as t-space.

So all the relationships between tasks can be expressed as constraints in t-space. The set of all the points in t-space that satisfy all the constraints is called **admissible region**. If, for example, we are working on a uniprocessor, it is not possible that two or more tasks share the same starting time. Consequently the admissible region must keep track of these constraints. Hence, our problem is *to find a set of t_i values in the admissible region which are optimal according to a given cost function.*

The goal of such optimization process can be:

- to minimize the *end-to-end response time*, which is the time required to complete a complex operation, independently by the specific sequence of tasks activation;
- to find the partitioning on multiprocessor which makes a tasks set schedulable;
- to minimize the *latency*, which is the time interval between the first instant when a task is ready and the instant when the task ends its computation.

3 System constraints

For the sake of clarity, we find useful to introduce the following definition:

> A system without constraints is an application composed by n **independent** tasks, running on p processors, with $p \geq n$.

It's easy to understand that in this case, no constraint is needed because every tasks can always run as soon as it is activated.

A real world application posseses two kinds of constraints:

- the application constraints which follow from the application structure (typically precedence contraints);
- the platform constraints, which follow from the hardware (e.g. the number of processors, the use of a particular resource).

In our model, both the kinds of constraints are expressed by mathematical relationships between the involved variables. In the following subsections all the possible constraints are explained in great detail.

3.1 Precedence constraint

The precedence constraint, expressed in Figure 1, is one of the most common constraint present in an application.

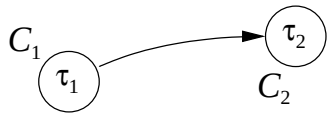
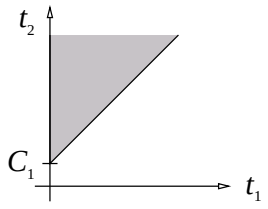


Figure 1. Precedence constraint

It means that task τ_2 will be activated by task τ_1. In terms of operating system primitive, this corresponds to a signal sent from τ_1 to τ_2. In compact form we write $\tau_1 \prec \tau_2$. This constraint, expressed in a mathematical formula, is:

$$t_2 \geq t_1 + C_1$$

and, consequently, the admissible region in the t-space, is



3.2 AND-synchronization constraint

This application constraint means that a task is activated when **all** the incoming signals have been received.

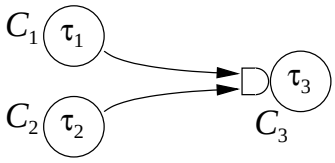


Figure 2. AND-synchronization constraint

In figure 2 the task τ_3 is activated when both the signal from τ_1 and the one from τ_2 are received. This constraint can be expressed by:

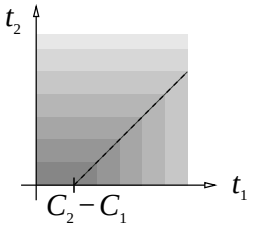
$$t_3 \geq \max\{t_1 + C_1, t_2 + C_2\},$$

and in general:

$$t_i \geq \max_{j \in S_i}\{t_j + C_j\},$$

where S_i is the set of task indexes sending a signal to the task τ_i. The meaning is clear: t_i, the starting time of τ_i, must be greater than the ending time of all the tasks sending τ_i a signal.

Following this definition, we can easily draw the admissible region in t-space, which is:



where the level curves of the t_3 surface are plotted in function of t_1 and t_2. A lighter color corresponds to a higher height.

3.3 OR-synchronization constraint

In a similar way, we introduce the OR-synchronization constraint. It means that a task is activated when **at least** one incoming signal has arrived. Its graphical representation is shown in figure 3.

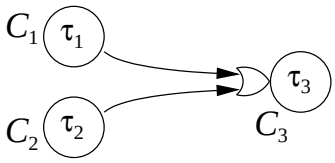


Figure 3. OR-synchronization constraint

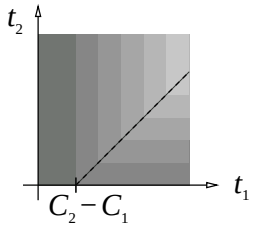
The only difference between the OR and the AND graphical representations is the input logic port of the task to be activated. Even the mathematical relationship differs slightly from the previous one. The concept is the following: t_i must be greater than the first ending time of all the tasks sending τ_i a signal. In mathematical formalism, it is expressed by:

$$t_i \geq \min_{j \in S_i}\{t_j + C_j\},$$

where S_i is the set of task indexes sending a signal to the task τ_i. In particular, for figure 3, it becomes:

$$t_3 \geq \min\{t_1 + C_1, t_2 + C_2\}.$$

The corresponding admissible region in the t-space (in the case when $n = 3$ and the tasks are related as shown in figure 3) is:



where the level curves of the t_3 surface are plotted in function of t_1 and t_2. A lighter color corresponds to a higher height.

3.4 Conditional constraint

This is a very common application constraint, present in every programming language: the IF statement. As we did before, we begin with the graphical representation shown in figure 4. The condition in

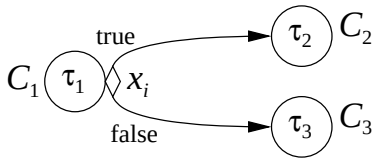


Figure 4. Conditional constraint

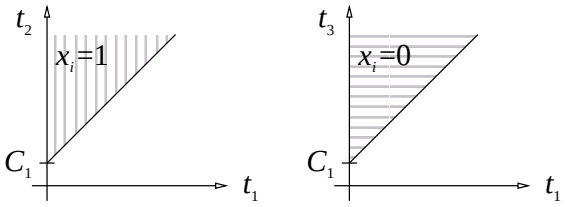
the figure can be expressed by the following sentence: "When task τ_1 finishes executing, if the condition expressed by the boolean variable x_i is true then τ_2 is activated, otherwise τ_3 is activated".

Putting this constraint in a mathematical expression is not immediate; therefore we will need an additional explanation for it. Here's the expressions related to the situation shown in the last figure:

$$\begin{cases} t_2 \geq t_1 + C_1 + M(1 - x_i) \\ t_3 \geq t_1 + C_1 + M x_i \end{cases}$$

where $x_i \in 0, 1$ is the boolean variable representing the condition, and M is a "big-enough" number (we mean that M must be enough to ignore everything you sum to it. There exists techniques to determinate its value).

As we can see, if x_i is 1, the last constraint becomes a precedence constraint for τ_2. Instead, τ_3 can never run because $t_3 \geq M$ means $t_3 \in \emptyset$. These relationships can then be displayed in the t-space in the following way.



3.5 Non-concurrency constraint

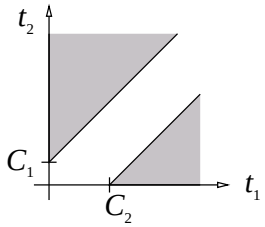
This kind of constraint can follow from both hardware and software specification:

- the number of processors. If p processors are present there cannot be more than p task executing concurrently;
- the use of shared resources. If a shared resource allows a maximum number r of concurrent access, this constraint must be specified in some way;
- the software organization. Usually all the jobs belonging from the same process cannot run concurrently.

In this case we don't introduce any graphical way to represent the constraint. We only express it in mathematical formulation. So, if we want impose the nonconcurreny between τ_1 and τ_2, we write the following relationship:

$$t_2 \geq t_1 + C_1 \ \vee \ t_1 \geq t_2 + C_2.$$

This so means that τ_2 can start only after τ_1 termination **or** τ_1 can start after τ_2. Representing this constraint in t-space leads to the following figure:



Following from both the last figure and the mathematical formula, this constraint can also be expressed as $(\tau_1 \prec \tau_2) \vee (\tau_2 \prec \tau_1)$. This means that every precedence constraint implies a non-concurrency constraint too (this last property will be very useful to simplify the set of all the constraints).

4 Future work

Clearly our research on this field isn't yet mature and this project is still undergoing. Our next steps will be to find a computationally reasonable algorithm for finding optimal solutions according to some cost function. Then, we plan also to extend the model to other optimization problems. Finally, all the explained techniques will be applied to a real world problem.

References

[1] PISATEL (**P**isa **I**nitiative on **S**oftware **A**rchitectures for **T**elecommunications), `http://galileo.iei.pi.cnr.it/ERI/`

[2] C.L. Liu and J.W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard real-Time Environment," *Journal of the ACM*, 1973.

[3] E. Bini, G. Buttazzo and G. Buttazzo, "A Hyperbolic Bound for the Rate Monotonic Algorithm", *IEEE Proc. of the* 13th *Euromicro Conf. on Real-Time Systems*, June 2001.

[4] G.C. Buttazzo, "Hard Real-Time Computing System", Kluwer Academic Publishers, 1997.

[5] C. Drodos, M. Zayadine and D. Metafas, "Real-Time Communication Protocol Development using SDL for an Embedded System on Chip Based on ARM Controller", *IEEE Proc. of the* 13^{th} *Euromicro Conf. on RT Systems*, June 2001.

[6] A.K. Mok and S. Sutanthavibul, "Modeling ans Scheduling of Dataflow Real-Time System", *Proc. of the IEEE Real-Time System Symposium*, 1985.

[7] "Processing Graph Method Specification, prepared by NRL for use by the Navy Standard Signal Processing Program Office (PMS-412)", 1987.

[8] S. Goddard and K. Jeffay, "Analyzing the Real-Time Properties of a Dataflow Execution Paradigm using a Synthetic Aperture Radar Application", *Proc. of the Real-Time Technology and Applications Symposium*, Montreal, Canada, 1997.

[9] E.A. Lee and D.G. Messerschmitt, "Static Scheduling of Data Flow Programs for Digital Signal Processing", *IEEE Transaction on Computers*, 1987.

[10] M. Di Natale, A. Sangiovanni-Vincentelli, F. Balarin, "Task Scheduling with RT Constrains", *Proc. of the Design Automation Conf.*, June 2000.