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@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 stimuli 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
3.1 Precedence constraint

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

\[ C_1 \rightarrow \tau_1 \rightarrow \tau_2 \rightarrow C_2 \]

Figure 1. Precedence constraint

It means that task \( \tau_2 \) will be activated by task \( \tau_1 \). In terms of operating system primitive, this corresponds to a signal sent from \( \tau_1 \) to \( \tau_2 \). 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 $\tau_3$ is activated when both the signal from $\tau_1$ and the one from $\tau_2$ are received. This constraint can be expressed by:

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

and in general:

$$t_i \geq \max\{t_j + C_j\},$$

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

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

![Figure 2](image)

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](image)

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 $\tau_i$ a signal. In mathematical formalism, it is expressed by:

$$t_i \geq \min\{t_j + C_j\},$$

where $S_i$ is the set of task indexes sending a signal to the task $\tau_i$. 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:

![Figure 3](image)

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 the figure can be expressed by the following sentence: “When task $\tau_1$ finishes executing, if the condition expressed by the boolean variable $x_i$ is true then $\tau_2$ is activated, otherwise $\tau_3$ 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:

$$
\begin{align*}
  t_2 &\geq t_1 + C_1 + M(1 - x_i) \\
  t_3 &\geq t_1 + C_1 + Mx_i
\end{align*}
$$

Figure 4. Conditional constraint
The non-concurrency constraint is defined as $x_i \in 0, 1$ 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 determine its value).

As we can see, if $x_1 = 1$, the last constraint becomes a precedence constraint for $\tau_2$. Instead, $\tau_3$ 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 $p$ tasks 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 non-concurrency between $\tau_1$ and $\tau_2$, we write the following relationship:

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

This so means that $\tau_2$ can start only after $\tau_1$ termination or $\tau_1$ can start after $\tau_2$. Representing this constraint in $t$-space leads to the following figure:

### 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


