Duration Specifications for Shared Processors*†

Zhou Chaochen¹,²,³
Michael R. Hansen¹
Anders P. Ravn¹
Hans Rischel¹

¹Department of Computer Science, Technical University of Denmark
²Programming Research Group, Oxford University, England
³on leave from
Institute of Software, Academia Sinica, Beijing, China

Abstract: We present a specification oriented real-time semantics for real-time programs consisting of communicating sequential processes running on a shared processor configuration. The semantics, which is given in Duration Calculus [7], separates properties of a (compiled) program from properties attributable to a scheduling strategy. This gives a clear division of concerns when a given program under a given scheduling strategy has to be proven correct wrt. hard real-time constraints.

Keywords: Duration Calculus, specifications, real-time systems, communicating systems, real-time programs, real-time semantics, scheduling.

1 Introduction

We are interested in specifying and verifying real-time properties of systems controlled by parallel programs. The systems we have in mind are occam-like [4] programs running on a transputer-like architecture, i.e. systems where several processes may share the same transputer (processor) and may share physical transputer links (channels). This requires a real-time semantics for programs \( P \in \text{Prog} \) and we begin with a general semantics

\[ \mathcal{L} \in \text{Prog} \rightarrow \mathcal{F} \]

which associates an implementation independent meaning with each program. Assume from now on the semantics domain \( \mathcal{F} \) to be logic formulas over real-time computations.

*This work is partially funded by ProCoS ESPRIT BRA 3104.
†This work is partially funded by the Danish Technical Research Counsil under project RapID.
If \( \text{Prog} \) includes an assignment \( z := e \), the only implementation independent real-time property of this assignment is that it takes some time \( t > 0 \) to execute. So the inferences about time from \( \llbracket x := e \rrbracket_G \) are weak; but they must be satisfied by any implementation of \( \text{Prog} \).

Knowing how \( \text{Prog} \) is compiled, more can be said about \( \text{Prog} \)'s real-time properties. For instance, from the compilation of \( z := e \) and from the speed of the target processor, we can calculate the time needed to execute the machine code for \( z := e \). Thus, we would have a lower bound of \( t \). Let \( \llbracket . \rrbracket_{\text{comp}} \) denote the compilation semantics of \( \text{Prog} \).

If \( \llbracket P \rrbracket_{\text{comp}} \) is consistent with \( \llbracket P \rrbracket_G \), then we say that \( \llbracket P \rrbracket_{\text{comp}} \) is a refinement of \( \llbracket P \rrbracket_G \) in the sense that

\[
\llbracket P \rrbracket_G \Leftarrow \llbracket P \rrbracket_{\text{comp}}
\]

In general one can have a sequence of such refinement steps, each consistently adding implementation details to the previous one.

A next possible step adds processor and channel scheduling information. From the compiler and target processor we know the time needed for e.g. \( z := e \), and when several processes share a processor, a scheduler specification \( SCH \) imposes constraints on how running time is granted to the individual processes. I.e. we can add a step to the above refinement:

\[
\llbracket P \rrbracket_G \Leftarrow \llbracket P \rrbracket_{\text{comp}} \Leftarrow (\llbracket P \rrbracket_{\text{comp}} \wedge SCH)
\]

Refinement is defined by implication and that requires that \( \llbracket . \rrbracket_G \), \( \llbracket . \rrbracket_{\text{comp}} \) and \( SCH \) have the same semantic domain, i.e. a single logical calculus over real-time computations. For this purpose we use the Duration Calculus [7].

A definition of \( \llbracket . \rrbracket_G \) essentially abstracts from concrete time bounds in the definition of \( \llbracket . \rrbracket_{\text{comp}} \). So we focus on \( \llbracket . \rrbracket_{\text{comp}} \), giving a specification oriented real-time semantics for a Timed CSP-like language, c.f. [6], at the compilation level where the speed of the target processor is known, and where we compile into a shared processor and channel architecture. From now on \( \llbracket . \rrbracket \) is used for \( \llbracket . \rrbracket_{\text{comp}} \).

The real-time semantics of a program \( P \) is a Duration Calculus formula \( \llbracket P \rrbracket \) describing a set of computations, which is independent of particular scheduling policies. Processes and computations are delineated in section 2, the Duration Calculus is introduced in section 3, and the semantics of processes is given in section 4. Schedulers are specified by separate Duration Calculus formulas. We illustrate this with a number of scheduler specifications in section 5 and 6.

In section 7 we illustrate that a program \( P \) with scheduler \( SCH \) satisfies the specification \( \text{SPEC} \) (see also [1]) if the formula

\[
\llbracket P \rrbracket \wedge SCH \Rightarrow \text{SPEC}
\]

is valid in Duration Calculus.