Deriving Real-Time Programs from Duration Calculus Specifications

François Siewe[1] and Dang Van Hung[2]

[1] Department of Maths. and Computer Science, University of Dschang
P. O. Box 96 Dschang, Cameroon. Fax (237) 45 13 81
fsiewe@uycdc.uninet.cm

[2] The United Nations University International Institute for Software Technology
P. O. Box 3058 Macau. Fax (853) 71 29 40
dvh@iist.unu.edu

Abstract. In this paper we present a syntactical approach for deriving real-time programs from a formal specification of the requirements of real-time systems. The main idea of our approach is to model discretization at state level by introducing the discrete states approximating the continuous ones, and then derive a specification of the control program over discrete states. Then the control program is derived from its specification using an extension of Hoare triples to real-time.

Keywords: Continuous specification, discrete design, real-time program, concurrency, shared variables, Hoare triples

1 Introduction

Real-time control systems usually consist of some physical plant, in permanent interaction with its environment, for which a suitable controller has to be constructed such that the controlled plant exhibits the desired time dependent behaviour.

This paper presents a syntactical approach for deriving real-time programs from a formal specification of the requirements of real-time systems. The approach provides a logical framework that can handle both continuous time and discrete time models in a uniform manner. We consider Duration Calculus (DC for short) [ZHR91] as specification and top level design language, for its effectiveness in reasoning about the design of real-time systems, and the fact that DC is successfully used in many case studies. We denote by DC^* an extension of DC with iteration [HuG99]. We link DC and Hoare logic to reason about both the real-time behaviour and the functional behaviour of real-time programs in a uniform manner.

Our design technique is formulated as follows. A real-time control system is a distributed hybrid system as it comprises continuous components (e.g the plant) and discrete components (e.g the controller). At the first step of the design process a state variables model of the system should be defined. The state model comprises continuous state variables (modelling the behaviour of the continuous

T. Margaria and T. Melham (Eds.): CHARME 2001, LNCS 2144, pp. 92–97, 2001.
© Springer-Verlag Berlin Heidelberg 2001

components) and discrete state variables (modelling the behaviour of the discrete components). Then the requirement of the system is formalised as a *DC* formula *Req* over continuous state variables. A design decision must be taken as how the requirement of the system will be met and refined into a detailed design *Des* over continuous state variables such that $Des \Rightarrow Req$. Then the discretization step follows. We approximate continuous state variables by discrete ones and formalise the relationship between them based on the general behaviour of the sensors and actuators. Then the control requirement is derived from the detailed design and refined into a *simple* DC^* formula *Cont* over discrete state variables such that $\mathcal{A} \vdash Cont \Rightarrow Des$, for some assumptions $\mathcal{A}$ about the behaviour of the environment and the relationship between continuous state variables and discrete state variables. The discrete formula *Cont* is the formal specification of the controller. The last step of the design process consists to write a real-time program for the controller and verify its correctness w.r.t. the specification *Cont* using our extended Hoare triples.

In the literature, some works have addressed the problem. Fränzle has developed in [Fra96] a technique for synthesizing controllers from Duration Calculus specifications. His approach is semantical, and so more difficult to use by engineers. Our aim is to provide the designers with syntax-based compositional interface for the design and verification of real-time programs, hiding semantic details. Another advantage of a syntactical approach is that the design process can be assisted by proof tools. In [Hoo94,XuM98,PWX98], some extensions of Hoare triples are proposed for real-time programs verification. However the derivation of a discrete design is not considered.

The remainder of the paper is organized as follows. We give a summary of *DC* in Section 2. Section 3 details our design technique. The program construction technique is presented in Section 4. Section 5 concludes the paper.

2 Duration Calculus with Iteration

In this section we give a brief summary of DC^*. The readers are referred to [HuG99] for more details on the calculus.

A language for DC* is built starting from the following sets of *symbols*: a set of *constant symbols* $\{a, b, c, \ldots\}$, a set of *individual variables* $\{x, y, z, \ldots\}$, a set of *state variables* $\{P, Q, \ldots\}$, a set of *temporal variables* $\{u, v, \ldots\}$, a set of *function symbols* $\{f, g, \ldots\}$, a set of *relation symbols* $\{R, U, \ldots\}$, and a set of *temporal propositional letters* $\{A, B, \ldots\}$.

A DC* language definition is essentially that of the sets of *state expressions* S, *terms* t and *formulae* φ of the language. These sets can be defined by the following BNFs:

$$\begin{array}{rcl} S & \widehat{=} & \mathbf{0} \mid P \mid \neg S \mid S \vee S \\ t & \widehat{=} & c \mid x \mid u \mid \int S \mid f(t, \ldots, t) \\ \varphi & \widehat{=} & A \mid R(t, \ldots, t) \mid \neg\varphi \mid (\varphi \vee \varphi) \mid (\varphi^\frown\varphi) \mid (\varphi^*) \mid \exists x \varphi \end{array}$$

A state variable P is interpreted as a function $I(P) : I\!R^+ \to \{0, 1\}$ (a state). $I(P)(t) = 1$ means that state P is present at time t, and $I(P)(t) = 0$ means

that P is not present at time t. We assume that a state has finite variability in any finite time interval. A state expression is interpreted as a function which is defined by the interpretations for the state variables and Boolean operators.

For an arbitrary state expression S, its duration is denoted by $\int S$. Given an interpretation I of the state variables and an interval, duration $\int S$ is interpreted as the accumulated length of time within the interval at which S is present. So for any interval $[t, t']$, the interpretation $I(\int S)([t, t'])$ is defined as $\int_t^{t'} I(S)(t)dt$.

A formula φ is satisfied by an interpretation in an interval $[t, t']$ when it evaluates to true for that interpretation over that time interval. This is written as $I, [t, t'] \models \varphi$.

Given an interpretation I, a formula $\varphi \frown \phi$ is true for $[t, t'']$ if there exists a t' such that $t \leq t' \leq t''$ and φ and ϕ are true for $[t, t']$ and $[t', t'']$ respectively.

We consider the following abbreviations: $\ell \mathrel{\widehat{=}} \int 1$, $\llbracket S \rrbracket \mathrel{\widehat{=}} (\int P = \ell) \land (\ell > 0)$, $\Diamond\varphi \mathrel{\widehat{=}} \mathit{true} \frown \varphi \frown \mathit{true}$, and $\Box\varphi \mathrel{\widehat{=}} \neg\Diamond\neg\varphi$.

3 From Continuous Specifications to Discrete Design

A model of real-time control systems is depicted in figure 1. The *plant* denotes the continuous components of the system, in permanent interaction with the physical environment. The *controller* is a discrete component denoting a program executed by a computer. The *sensors* sample the states of the plant for them to be observable by the controller. The *actuators* receive commands from the controller and control the plant accordingly. The sensors and the actuators constitute the continuous-to-discrete and discrete-to-continuous interfaces respectively.

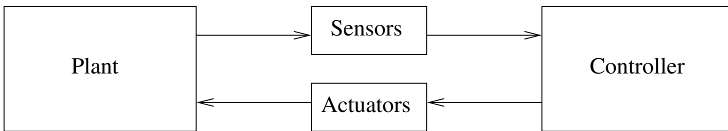

Fig. 1. A model of controlled system

We define three concepts for formalising the relationship between continuous state variables and discrete ones. The first concept is the *stability* of continuous state variables. This property requires that continuous state variables should not change very fast in order to be observable at discrete time.

Definition 1 (Stability). *Given a state variable s and a positive real number δ, we say s is δ-*stable *iff the following formula is satisfied by any interval*

$$\delta\text{-}\mathit{stable}(s) \mathrel{\widehat{=}} \Box(\llbracket \neg s \rrbracket \frown \llbracket s \rrbracket \frown \llbracket \neg s \rrbracket \Rightarrow \llbracket \neg s \rrbracket \frown (\llbracket s \rrbracket \land \ell > \delta) \frown \llbracket \neg s \rrbracket)$$

Definition 2 formalises the relationship between a state variable r and a state variable s that always follows r after some time.

Definition 2 (Control state). *Given two state variables r and s, and a non-negative real number δ, we say r δ-controls s iff the following formula is satisfied by any interval*

$$r \rhd_\delta s \mathrel{\widehat{=}} \Box(\llbracket r \rrbracket \wedge \ell > \delta \Rightarrow (\ell \leq \delta)^\frown \llbracket s \rrbracket)$$

The concept of *control state* can be used for formalising the behaviour of actuators. Let r be a state variable modelling a program command, and s a state of the plant. Then the relation $r \rhd_\delta s$ means that whenever the controller issues the command r, the plant gets into state s within at most δ time units. So the maximum response time is δ time units.

Definition 3 formalises the relationship between a state variable s and a state variable r that observes any change (stable enough) in s.

Definition 3 (Observation state). *Given two state variables r and s, and a non-negative real number δ, we say r δ-observes s iff the following formula is satisfied by any interval*

$$r \overrightarrow{\approx}_\delta s \mathrel{\widehat{=}} (s \rhd_\delta r) \wedge (\neg s \rhd_\delta \neg r)$$

The concept of *observation state* can be used for formalising the behaviour of sensors. Let r be a state variable modelling a program variable, and s a state of the environment. Then the relation $r \overrightarrow{\approx}_\delta s$ means that any change (stable enough) in s is observed by the controller within δ time units. So the sampling step is δ time units. Note that the definition says nothing about unstable change of s. We will assume that continuous state variables are stable enough, otherwise there is no way for a digital controller to guarantee safety requirements in general.

The discrete interface is formalised as follows. For any continuous state variable s, we consider a discrete state variable s_c used by the controller to observe s via the sensors. The relationship between s and its sampling s_c is formalised by $s_c \overrightarrow{\approx}_\delta s$, for some non-negative real number δ (representing the sampling step). Similarly, for any state p of the plant we consider a command p_c, a discrete state, for requesting (via the actuators) the plant getting into state p. The relationship between p and p_c is formalised by $p_c \rhd_\tau p$, for some non-negative real number τ (representing the response time).

We provide rules useful for refining a continuous design towards a discrete design. Following are some examples. More rules and their proofs are given in [SiH00a].

Rule 1 (Transitivity of control)

(a) $\dfrac{(r \rhd_\delta s) \quad (s \rhd_\tau t)}{r \rhd_{(\delta+\tau)} t}$; (b) $\dfrac{(r \overrightarrow{\approx}_\delta s) \quad (s \overrightarrow{\approx}_\tau t)}{r \overrightarrow{\approx}_{(\delta+\tau)} t}$

Rule 2 (Monotonicity)

(a) $\dfrac{r \rhd_\tau s \quad \tau \leq \delta}{r \rhd_\delta t}$; (b) If $r \Rightarrow s$ then $r \rhd_0 s$; (c) $\dfrac{r \rhd_\delta s \quad t \rhd_\delta u}{(r \wedge t) \rhd_\delta (s \wedge u)}$

Rule 3 (Sequential (a) and Parallel (b) composition)

$$\text{(a)}\ \frac{\begin{array}{c}\psi \Rightarrow \Box\chi \quad \varphi \Rightarrow \Box\chi \\ \psi \Rightarrow \neg(\neg\beta^\frown true) \quad \varphi \Rightarrow \neg(true^\frown\neg\alpha) \quad \alpha^\frown\beta \Rightarrow \chi\end{array}}{\varphi^\frown\psi \Rightarrow \Box\chi}\ ;\quad \text{(b)}\ \frac{A \Rightarrow \Box\psi \quad B \Rightarrow \Box\varphi}{A \wedge B \Rightarrow \Box(\psi \wedge \varphi)}$$

4 Real-Time Programs Construction

In this section we present a real-time programming language and our extension of Hoare triples as an abstract semantics of it. We develop a set of verification and design rules for a program. The readers are referred to [SiH00b] for the formal semantics of the language and the proof of the soundness of the verification rules which are based on Weakly Monotonic Time Duration Calculus ([PaH98]).

In the following BNFs, S ranges over programs, P and Q range over processes, e over arithmetic expressions, g over guards (Boolean expressions), d over positive natural numbers ($d > 0$), and x over program variables.

$$\begin{array}{rl} P ::= & \textbf{skip} \mid \overline{x} := \overline{e} \mid \textbf{delay}\ d \mid \textbf{await}\ g \mid P; Q \mid \\ & \textbf{if}\ g\ \textbf{then}\ P\ \textbf{else}\ Q\ \textbf{fi} \mid \textbf{while}\ g\ \textbf{do}\ P\ \textbf{od} \end{array}$$

$$S ::= P_1 \parallel P_2 \parallel \ldots \parallel P_n, \quad (n \geq 1)$$

For simplicity, we do not include non-determinism in the language. We assume the *true synchrony hypothesis*, viz, computation and communication do not take time, only the waiting for external synchronisation or explicit delay statements take time. The statement **delay** d delays the process for d time unit(s). During this period, the process could not do anything. Another timing control statement is **await** g which makes the process wait until the guard g is fired by the occurrence of an external event. The statements **delay** and **await** are the only statements that take time. The other statements conserve their usual meaning. Processes can share variables. We denote by $\mathcal{WO}(P_i)$ the set of variables written only by process P_i. Of course, local variables of P_i belong to that set. We also denote by $\mathcal{VAR}(\alpha)$ the set of variables occurring in $\alpha \in \{P, e, g\}$.

We extend the Hoare logic to real-time concurrent programming with shared variables to reason about terminating as well as non-terminating real-time programs. Unlike computational programs, real-time control programs in general run forever. Our extension of Hoare triples has two forms. The first form is $\{p\}[P, \varphi]\{q\}$, meaning that if the precondition p holds in the initial state and P terminates, then the postcondition q holds in the final state and the execution period of the program satisfies the duration formula φ. The second form is $\{p\}[P, \varphi]\{\}$, meaning that if the precondition p holds in the initial state and P does not terminate, then any prefix of the execution period of the program satisfies the duration formula φ. This couple of triples is used to reason about the properties of real-time programs. Following are some examples of our extended Hoare triples.

Rule 4 (**Assignment**)
$\{q[\overline{x}\backslash\overline{e}]\}[\overline{x} := \overline{e}, \ell = 0]\{q\}$

Rule 5 (**Delay** in process P_i)
$\{q\}[\textbf{delay}\ d, \ell = d \wedge \llbracket q \rrbracket]\{q\}$, provided $\mathcal{VAR}(q) \subseteq \mathcal{WO}(P_i)$.

Rule 6 (**Consequence**)
(a) $\dfrac{\{p\}[P,\varphi]\{q\} \quad p' \Rightarrow p \quad q \Rightarrow q' \quad \varphi \Rightarrow \phi}{\{p'\}[P,\phi]\{q'\}}$; (b) $\dfrac{\{p\}[P,\varphi]\{\} \quad p' \Rightarrow p \quad \varphi \Rightarrow \phi}{\{p'\}[P,\phi]\{\}}$

5 Conclusion

In this paper we have presented a technique for the design of real-time hybrid systems using Duration Calculus. We provide a set of rules for deriving a discrete design from the specification of a real-time system. Then we extend Hoare triples to real-time and develop rules for designing a real-time program that satisfies the discrete design.

References

[Fra96] M. Fränzle. Synthesizing Controllers from Duration Calculus Specifications. Proceedings of *Formal Techniques in Real-Time and Fault-Tolerant Systems (FTRFT'96)*, LNCS 1135, Springer-Verlag, 1996.

[Hoo94] Jozef Hooman. Extending Hoare Logic to Real-Time. *Formal Aspects of Computing*, 6A:801-825, 1994.

[HuG99] Dang Van Hung and Dimitar P. Guelev. Completeness of a fragment of Duration Calculus with Iteration. Proceedings of Asian Computing Science Conference (ASIAN'99), LNCS 1742, Springer-Verlag, 1999, pp. 139–150.

[PaH98] Paritosh K. Pandya and Dang Van Hung. Duration Calculus with Weakly Monotonic Time. Proceedings of *Formal Techniques in Real-Time and Fault-Tolerant Systems*, LNCS 1486, pp. 55–64, Springer-Verlag, 1998.

[PWX98] Paritosh K. Pandya, Wang Hanpin, and Xu Qiwen. Towards a Theory of Sequential Hybrid Programs. Proceedings of the *International Conference on Programming Concepts and Methods* (PROCOMET'98), Chapman & Hall, 1998, pp. 366-384.

[SiH00a] François Siewe and Dang Van Hung. From continuous specification to discrete design. Proceedings of the *International Conference on Software: Theory and Practice* (ICS2000), Yulin Feng, David Notkin and Marie-Claude Gaudel (eds), Beijing, August 21-24, 2000, pp. 407-414.

[SiH00b] François Siewe and Dang Van Hung. Deriving Real-time Programs from Duration Calculus Specifications. Technical Report 222, UNU/IIST, P.O. Box 3058, Macau, December 2000.

[XuM98] Xu Qiwen and Mohalik Swarup. Compositional Reasoning using Assumption-Commitment Paradigm. Technical Report 136, UNU/IIST, P.O. Box 3058, Macau, February 1998.

[ZHR91] Zhou Chaochen, C.A.R. Hoare, and Anders P. Ravn. A calculus of duration. *Information Processing Letters*, 40(5):269–276, 1991.