Real-Time Systems Development with Duration Calculi: an Overview

Dang Van Hung

The United Nations University
International Institute for Software Technology
P.O.Box 3058, Macau

Abstract. In this paper we present the main achievements of UNU/IIST during the project "Design Techniques for Real-time Systems" carried out since 1993 by its staff and fellows. Duration Calculus was originally introduced in 1991 as a powerful logic for specifying the safety of real-time systems. During the project, it has evolved to a set of calculi that can capture many important aspects in real-time systems development including techniques for specification, design, discretisation and verification. These techniques are discussed informally in the paper.

1 Introduction

Development of safety-critical systems has received a great deal of attention for long time, and has shown challenges to the researchers and developers. There is no doubt that formal methods have played a key role in response to these challenges. To assist developing countries with advanced software development techniques, UNU/IIST has its focus in formal methods. In this context, the research group in UNU/IIST has carried out the project *Design Techniques for Real-Time Systems* since its establishment in 1993. The aim of the project is to help researchers from developing countries to do research at the international standard. The approach taken by the group is based on real-time logics: starting from Duration Calculus developed in the ProCoS projects ESPRIT BRA 3104 and 7071 in 1990 by Zhou, Hoare and Ravn [6], the group has developed new techniques for specifying all kinds of requirements of real-time systems including functionality requirements and dependability requirements. In order to support the design and verification of real-time systems the language of Duration Calculus has been developed further to be able to describe systems at more detailed and lower levels of abstraction that can be translated directly into a programming language. This language is supported with refinement methods, a powerful proof system and tool support for deriving, reasoning and verifying designs. The main achievements of this project are summarised in this paper.

To motivate our work, for the rest of this section we describe some concepts and aspects related to real-time systems that we have to handle during their development. They are time, durations, requirements, phases, synchrony hypothesis and discretisation.

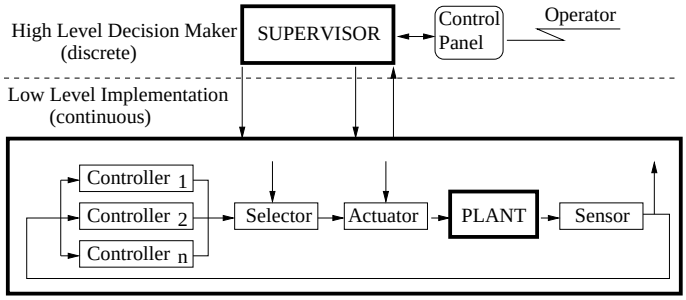

Fig. 1. Structure of Real-Time Embedded Systems

Time can be either continuous or discrete. Continuous time is represented by the set of non-negative reals, while discrete time is represented by the set of natural numbers. Although discrete time has been well accepted in computer science, we adopt continuous time. The reason for this is that for the purpose of specification, continuous time is easier to use and more familiar to the people who do not know computer science than discrete time is. Real-time systems have their behaviour dependent on time, and hence, can be presented as functions of time. For instance, a state P of a system can be considered as a Boolean value function $P : Time \to \{0, 1\}$. $P(t) = 1$ says that state P is present at time t, and $P(t) = 0$ says that state P is absent at time t. Hence, the accumulated time for the presence of state P over a time interval $[b, e]$ is exactly $\int_b^e P(t)\, dt$. We call $\int_b^e P(t)\, dt$ the duration of state P over interval $[b, e]$. When the states of a system are modelled as Boolean valued functions of time, their durations can be a powerful concept for specifying the requirements of the system. For example, one requirement of a simple gas burner is that "for any observation interval that is longer than 60 seconds, the ratio between the duration of the state *leak* and the length of the interval should not be more than 5%". This requirement means that there must not be too much leak relative to the length of the observation interval. A language which is able to formally specify all kinds of requirements for real-time systems based on state durations would be useful. This led to the development of Duration Calculus and its extensions.

A commonly accepted structure of a real-time embedded system ([56, 50]) is shown in Figure 1.

A 'control program' (or supervisor) is the goal of the development of a real-time embedded system. The design process with the step-wise refinement method can be described as follows. Starting from the requirement $\mathcal{R}$ of the plant and the assumption $\mathcal{E}$ about the behaviour of the environment:

1. Derive a finer specification D for the plant such that $\mathcal{R}$ is provable from D under $\mathcal{E}$. Repeat this step with D in the role of $\mathcal{R}$ until we feel that D is detailed enough for implementation,
2. Derive a specification of the control program $\mathcal{S}$ such that D is provable from $\mathcal{S}$ under the assumption $\mathcal{E}$ and the assumption $\mathcal{I}$ about the interface between the plant and the control program including the assumption about the behaviour of the sensors, actuators and the controllers.
3. Refine $\mathcal{S}$ in the discrete world.

The first step involves the refinement in the continuous world. DC (with its extension) and its powerful proof system are a good formalism for supporting and reasoning in this step. There are some issues in carrying out Step 2. Since the sensors and actuators are working in discrete time, there is a delay (bounded by the distance between two ticks of clock) between the change of a state and the time it is observed (by the sensor). If a state changes very frequently (relative to the sensor's clock), some changes may not be observable. If we take this phenomenon into account, we need a technique for discretisation of time. In many cases, where we can assume that the clocks of the sensor and actuator are very fast, and that the plant is relatively stable, all the changes of the plant states are observable within a negligible time-delay. In the literature, most of the techniques are based on this assumption, and skip the discretisation step for simplicity. However, this assumption cannot be satisfied when the plant is also a computing device, such as a communication system at the physical layer.

In carrying out the third step, most of the techniques use the following hypothesis called synchrony hypothesis: the communication and computation take no time, only the explicit delays or waiting (for an event) take time. While this assumption makes it easy to handle the time features in real-time programming languages, it causes some problems for reasoning. For instance, if the command $x := 1; x := 2$ takes place instantaneously at a time point t then x has two values at the time t which is a contradiction in the model. A formal reasoning technique for handling this situation is needed. For example, to express a discrete change of the value of a state at a time point, a neighbourhood of the point has to be considered. When there are a few changes happening sequentially at a time point, the time moment has to be treated as a virtual time period.

In the following sections we will describe how these issues are treated by Duration Calculus and its extensions. In fact, they are motivations for the development of our design techniques for real-time systems.

2 Duration Calculus and Some of Its New Features

Research and development of Duration Calculus was initiated in [6]. In this section, we present some new main features of the development for specifying real-time systems.

As said in the previous section, a state of a real-time system is modelled by a Boolean valued function of the real-time. A duration term of a state P is denoted by $\int P$, and defined as a function of time intervals. The function

$\int P$ when applied to an interval $[b, e]$ is the duration of P over that interval, i.e. $\int_b^e P(t)\, dt$. A formula (or sentence) of the language is just a constraint over state durations, which is a kind of statements about intervals. For example, $20 * \int leak \leq \ell$ is a formula, where ℓ is a term to express the length of the interval to which the term ℓ is applied. This formula is satisfied by an interval $[b, e]$ iff $20 * \int_b^e leak(t)\ dt \leq e - b$ holds. A formula $\int P = \ell \wedge \ell > 0$ holds for an interval $[b, e]$ iff $[b, e]$ is not a point interval, and P holds almost every where in $[b, e]$. If this is the case, $[b, e]$ is called a phase of state P. The concept of phase here bears the same meaning as in the literature. For simplicity, a phase of P is denoted by $[\![P]\!]$, and the formula $\ell = 0$ is denoted by $[\![\ ]\!]$. A more complicated formula is built by using classical logical connectives like $\vee, \neg, \exists$, etc., and the infix binary modality $\frown$ called the *chop* modality. Formula $\psi \frown \phi$ is satisfied by an interval $[a, b]$ iff for some $m \in [a, b]$, ψ is satisfied by $[a, m]$ and ϕ is satisfied by $[m, b]$. Formula $\Diamond\phi$ holds for an interval $[a, b]$ iff there exists a sub interval of $[a, b]$ that satisfies ϕ. On the other hand, formula $\Box\phi$ holds for an interval $[a, b]$ iff any sub interval of $[a, b]$ satisfies ϕ. So, $\Box\phi$ is the dual modality of $\Diamond\phi$, i.e. $\Box\phi \Leftrightarrow \neg\Diamond\neg\phi$.

A proof system consisting of axioms and inference rules has been developed and proved to be complete relative to the completeness of the real numbers [20]. This makes Duration Calculus a powerful formalism for specifying and reasoning about safety requirements of real-time systems. For example, the safety requirement of the gas burner [6] mentioned in the introduction of this paper is written simply as

$$\Box(\ell \geq 60 \Rightarrow 20 * \int leak \leq \ell),$$

or the performance requirement for a double water-tank system [50] is written as

$$\Box([\![W]\!] \wedge \ell > L_o \Rightarrow \int Steady \geq \gamma * \ell)$$

which says that if the water-tank system is observed longer than L_0 time units during its working time, the accumulated time for the steady state of the water level should be at least γ times the observation time. The expressive power of DC lies on the fact that it can specify a property of a system in term of time intervals and state durations, the concepts that could not be captured by any real-time logic before the introduction of DC.

While this language is good enough for the specification of the safety requirement of real-time systems in general, and for the first step of development outlined in the previous section, it cannot be used to specify liveness and fairness since a formula can specify only a property of a system *inside* an interval. This observation has led to the development of a Duration Calculus with infinite intervals [7] and a neighbourhood logic [5].

2.1 Neighbourhood Logics

In [5] Zhou and Hansen introduced a Neighbourhood Logic which is able to specify liveness and fairness with a powerful proof system. This logic is similar to Venema's CDT logic ([51]), but more powerful. Namely it can be used to reason

about state durations. It extends the basic Duration Calculus with the neighbourhood modalities $\diamond_l$ and $\diamond_r$. For a formula ϕ, the formula $\diamond_l\phi$ is satisfied by an interval $[a, b]$ iff there exists an interval which is a left neighbourhood of a, i.e. $[m, a]$ for some m, that satisfies ϕ. On the other hand, the formula $\diamond_r\phi$ is a formula which is satisfied by an interval $[a, b]$ iff there exists an interval which is a right neighbourhood of b, i.e. $[b, m]$ for some m, that satisfies ϕ.

The neighbourhood logic is very expressive. Each of the thirteen possible relations between two intervals can be represented by a formula in this logic. We refer readers to [5] for the expressiveness and (relative) complete proof system of the neighbourhood logic. It turns out that not only fairness and liveness properties can be specified, but more complicated properties like the limit of durations can be defined in the logic as well. For example, the statement about interval "the interval can be extended to the right to an interval that satisfies ϕ" is specified by

$$\exists x.(\ell = x \wedge \diamond_l\diamond_r(\phi \wedge (\ell = x^\frown true))),$$

where for a formula D, $\diamond_l\diamond_r D$ says that D holds for an interval having the same starting point. Let us denote this formula by $\mathbf{E}\phi$. So, $\mathbf{E}\phi$ expresses the liveness of ϕ. Fairness of formulas ψ means that any extension to the right of the reference interval has an extension to the right of it for which ψ. This means that ψ is satisfied infinitely often. This fairness of ψ is specified by $\neg\mathbf{E}\neg\mathbf{E}\psi$.

The fact that a Boolean-valued state S has divergent duration when the right end point of the reference approaches ∞ can be specified simply by

$$\forall x.(\mathbf{E}(\textstyle\int S > x))$$

This is abbreviated as $\lim \int S = \infty$ as in mathematical analysis.

The fact that state S takes v as the limit of its duration when the right end point of the reference approaches ∞ can be specified by

$$\forall \epsilon > 0.(\mathbf{E}\neg\mathbf{E}(|v - \textstyle\int S| \geq \epsilon))$$

This is abbreviated as $\lim \int S = v$.

Similarly, we can specify the absolute fairness of two states S_1 and S_2 by

$$\forall \epsilon > 0.(\mathbf{E}\neg\mathbf{E}(|\textstyle\int S_1 - \int S_2| \geq \epsilon))$$

This is denoted by $\lim(\int S_1 - \int S_2) = 0$ as usual. With this logic, the semantics of a real-time program P can be defined by a pair of formulas (S_f, S_i). The formula S_f specifies the termination behaviour of P, which is the specification of P for the interval from the time P starts to the time P terminates. The formula S_i specifies the nontermination behaviour of P, which is a specification of P for any prefix of the infinite interval starting from the time P starts. Because in general, a specification of the nontermination behaviour of P includes the liveness and fairness of P, S_i should preferably be a formula in the neighbourhood logic. The details of the technique are described in [23, 17, 48].

The synchrony hypothesis and handling the behaviour of a state at a time point have motivated several extensions of Duration Calculus: Duration Calculus with super-dense chop [19], Duration Calculus with projections [23, 17], Mean-Value Duration Calculus [10], Duration Calculus with Weakly Monotonic Time [40] and Two-Dimension Duration Calculus [2]. The first two calculi use the following idea to solve the problem arising with the synchrony hypothesis mentioned in the previous section: in fact, the communications and computations do take time, and therefore the temporal order between the operations are reflected by the real-time. Because the time spent in performing these operations is negligible, it is hidden to the observers. The approach of implicitly hiding the computation time is taken by the super-dense chop operator, and the approach of explicitly hiding the computation time is taken by the projections over states. The first approach is presented in the the following subsection.

2.2 Super-dense Chop

In 1996, Zhou and Hansen [19] extended the basic DC with the modality $\bullet$ called super-dense chop. The idea can be seen in the following example. Consider the command $x := x + 1$. The meaning of this command can be defined by a DC formula $\ell = 0 \wedge \overleftarrow{x} + 1 = \overrightarrow{x}$, where $\overleftarrow{x}$ and $\overrightarrow{x}$ are functions of intervals which, when applied to an interval, give the value of the variable x in the left and right neighbourhood of the interval respectively (note that x is a piecewise constant function of time having the finite variability, and hence $\overleftarrow{x}$ and $\overrightarrow{x}$ are defined). Now consider the program $x := x + 1; x := x + 1$. This program is executed instantaneously according to the synchrony hypothesis. The right meaning for it is $\ell = 0 \wedge \overleftarrow{x} + 2 = \overrightarrow{x}$. How is this meaning derived from the meanings of $x := x + 1$ and sequential composition? We may assume that the composition ';' takes a virtual time for passing the value of x from the first command to the second, i.e. there is a virtual time interval between the two commands, the value of x in this interval is exactly the output of the first command, and at the same time it is the input for the other command. This interval is virtual, i.e. it is considered as a point, and is treated as an interval only when necessary. This point is expressed by $\bullet$. So, a chopping point can be implicitly expanded (mapped) to an interval to express the computation with negligible time for which the state variables are kept unchanged (to express the on-going computation process). Let Φ and Ψ be DC formulas. Then $\Phi \bullet \Psi$ is also a formula. The semantics for this formula can be defined via higher order Duration Calculus. Higher order Duration Calculus extends basic with quantifications over state variables and with terms of the forms $\overleftarrow{s}$ and $\overrightarrow{s}$, where s is a piecewise constant function of time having finite variability [4, 32] (for higher order DC, Zhan Naijun in [38] has given a (relatively) complete proof system just like first order DC). The operator $\bullet$ is defined as follows. For any higher order Duration Calculus formulas $\Phi(V_1, \ldots, V_n)$ and $\Psi(V_1, \ldots, V_n)$, where $V_1, \ldots, V_n$ are all their free

state variables and real state variables,

$$\begin{aligned}\Phi \bullet \Psi \mathrel{\widehat{=}} \exists x_1, \ldots, x_n, V_{l1}, \ldots, V_{ln}, V_{r1}, \ldots, V_{rn}.\\ (\Phi(V_{l1}, \ldots, V_{ln}) \wedge_{i=1}^{n} ((\lceil V_i = V_{li} \rceil \vee \lceil\ \rceil) \wedge (\overleftarrow{V_i} = \overleftarrow{V_{li}}) \wedge (\overrightarrow{V_{li}} = x_i)))^\frown\\ \Psi(V_{r1}, \ldots, V_{rn}) \wedge_{i=1}^{n} ((\lceil V_i = V_{ri} \rceil \vee \lceil\ \rceil) \wedge (\overrightarrow{V_i} = \overrightarrow{V_{ri}}) \wedge (\overleftarrow{V_{ri}} = x_i))))\end{aligned}$$

Global variables x_k in this formula play the role of passing the values of the state variables V_k at the chop point. As said above, the operator $\bullet$ is introduced for giving precise semantics for sequential composition of programs under the synchrony hypothesis. Let $Sem[\mathcal{P}]$ be semantics of program P in higher order DC with $\bullet$. Then, we have

$$Sem[\mathcal{P}_1; \mathcal{P}_2] \mathrel{\widehat{=}} Sem[\mathcal{P}_1] \bullet Sem[\mathcal{P}_2]$$

Hence the logic can reason about both parallel composition and sequential composition in a very comfortable way: parallel composition corresponds to conjunction, and sequential composition corresponds to super-dense chop.

As an example, in this logic, we can prove that

$$Sem[x := x + 1; x := x + 2] \Leftrightarrow Sem[x := x + 3]$$

In fact, the semantics of the two sides is the formula $\overrightarrow{x} = \overleftarrow{x} + 3 \wedge \ell = 0$. So, the behaviour of the program at an instant is invisible, and is abstracted away by this logic. Logic with super-dense chop is convenient for reasoning about the abstract behaviour of programs. In reasoning about the behaviour of programs at the operational level, in some cases we may not want to identify the program $x := x + 1; x := x + 2$ with $x := x + 3$. We want to identify them only when we observe them at the higher level of abstraction. This is the idea for introducing the projection operator into the logic [23, 17].

2.3 Recursive Duration Calculus and Iteration

To capture the behaviour of recursive procedures and loops in programs, Recursive Duration Calculus (μHDC) has been introduced by Paritosh Pandya [41] and was developed further by Xu Qiwen, Wang Hanpin and Paritosh Pandya [42]. Later on, He Jifeng and Xu Qiwen studied the operator using an algebraic approach [32] and developed some algebraic laws for a sequential hybrid language developed by them. A thorough study was given later by Guelev [15] with a completeness result for a class of μHDC. A subclass of μHDC to handle iterations was introduced earlier in [27] and was studied later in [25]. Presentation of the recursion operator μ needs a formal definition of Duration Calculus and is rather technical which is out of scope for this survey paper. Therefore, we present here just the iteration operator, which is a simple form of the recursion operator.

The iteration of a duration calculus formula ϕ is a formula denoted by ϕ^*. The iteration formula ϕ^* holds for an interval $[a, b]$ iff either $a = b$ or $[a, b]$ can

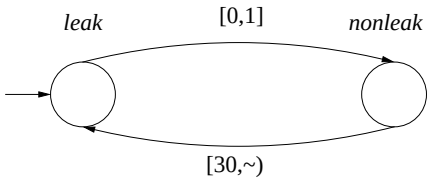

Fig. 2. Simple Design of Gas Burner

split into a finite number of intervals, each of which satisfies ϕ. Formula ϕ^* can be written using μ operator as:

$$\mu X.(\ell = 0 \vee X^\frown\phi)$$

The behaviour of the simple Gas-Burner depicted as a simple timed automaton in Fig. 2 can be written simply and intuitively by a formula with iteration $((\lceil leak \rceil \wedge \ell \leq 1)^\frown(\lceil nonleak \rceil \wedge \ell \geq 30))^*{}^\frown(\lceil\ \rceil \vee (\lceil leak \rceil \wedge \ell \leq 1) \vee ((\lceil leak \rceil \wedge \ell \leq 1)^\frown\lceil nonleak \rceil))$ similar to the derivation of a regular expression from a finite state machine. The correctness of the simple gas burner is expressed by

$$((\lceil leak \rceil \wedge \ell \leq 1)^\frown(\lceil nonleak \rceil \wedge \ell \geq 30))^* \Rightarrow \\ \Box(\ell \geq 60 \Rightarrow \textstyle\int leak \leq (1/20)\ell)\,.$$

With recursion and iteration operators, it has been shown that DC can be used to define semantics for real-time programming languages like Verilog, OC-CAM and WHILE like languages ([32, 48, 23]).

Apart from the fact that it makes the calculus more powerful and more comfortable to specify loops and recursive procedures in programs, proof systems have been developed for the calculus which are complete for a class of formulas that are good enough in most of applications encountered in practice. Readers are referred to [25, 15] for these proof systems. The proof system developed in [25] for Duration Calculus with iterations is complete for a class of formulas that is as expressive as the class of timed automata. Proof systems are a basis for the formal verification of designs. A formal proof of the validity of the above formula for the correctness of the simple gas burner is given in [25].

2.4 Probabilistic Duration Calculus

Reasoning about the probability of the satisfaction of a requirement written in DC by a system (a kind of timed automata) plays an important role in verifying the dependability of real-time systems. In the project, we have proposed a probabilistic model for analysing system dependability as finite probabilistic automata with stochastic delays of state transitions.

A continuous time probabilistic automaton is a tuple $M = (S, A, s_0)$, where S is a finite set of states, $s_0 \in S$ is the initial state of M, and A is a finite set

of transitions, $A \subseteq (S \times S) \setminus \{(s,s) \mid s \in S\}$ (here we reject idle transitions, and therefore assume $(s,s) \notin A$ for all $s \in S$). Each transition $a \in A$ is associated with a probability density function $p_a(t)$ and a probability q_a. Let $a = (s,s') \in A$ and $A_s = \{(s,s') \in A | s' \in S\}$. Our intention of introducing $p_a(t)$ is to specify that if M enters the state s at an instant τ of time, then the probability density that the transition a occurs at $\tau + t$ (delay-time of a is t) and causes M to change to the state s' is $p_a(t)$ independent of τ, given that this transition is chosen to occur in the case that there is more than one transition enabled in s. The probability that a is chosen to occur when M is in s is given by q_a. Thus, we require that $\sum_{a \in A_s} q_a = 1$ for $s \in S$ which satisfies $A_s \neq \emptyset$.

Each behaviour of M corresponds to an interpretation of the DC language over the set of state variables S in the obvious way. Given a DC formula D over the set of state variables S, we can partition the set of timed behaviours of the automaton into a finite number of subsets, for which the probability that D is satisfied in an interval $[0,t]$ can be defined. Hence, the probability that D is satisfied in an interval $[0,t]$ by all the behaviours of M is just the sum of the probabilities that D is satisfied in the interval $[0,t]$ by the behaviours in the partitions i. In this way, we can compute the probability $\mu_t(D)$ that the automaton M satisfies a DC formulas D in an interval of time $[0,t]$ according to the classical probability calculus. The computation is very complicated and is not necessary in many applications. Therefore, we need a technique for reasoning about this probability in a comfortable way. We have developed a set of rules towards this aim. The rules capture the properties of the $\mu_t(D)$s including the forward equation in the classical theory of stochastic processes. For example, consider a simple gas burner in the previous section. Assume that the rate of becoming *leak* in one second is $\lambda = (10 \times 24 \times 3600)^{-1}$ (meaning that on average, once in every 10 days), and that stopping *leak* has a normal probability distribution with mean value 0.5 and deviation δ. By using our rules, we can prove that the requirement is satisfied by the gas burner in one day with the probability at least 0.99 if the deviation δ is less than 0.0142. This value characterises the necessary precision of the components of the system.

The readers are referred to [24, 55] for the detail of the technique. This technique and idea has been generalised into a probabilistic neighbourhood logic recently by Guelev in [16].

3 Model Checking Techniques for Duration Calculus

In general, DC is highly undecidable. Only the class of DC formulas built from $[\![P]\!]$ using Boolean connectives and the modality $^\frown$ is decidable [9]. Note that the behaviour of a timed automaton can be represented by a DC formula, and hence, checking a timed automaton for a DC requirement can be converted to checking the validity of a DC formula by using the deduction theorem of DC. As a result, we cannot expect a universal model checking technique for DC formulas. In order to achieve relatively efficient model checking techniques, we

have to restrict ourselves to a restricted class of system models and restricted class of DC requirements.

3.1 Checking Timed Regular Expression for Duration Invariants

In [12], a class of DC formulas is introduced to express the behaviour of real-time systems. The formulas in this class are said to be simple and are generated by the following grammar:

$$\varphi \mathrel{\widehat{=}} \lceil S \rceil \mid a \leq \ell \mid \ell \leq a \mid (\varphi \vee \varphi) \mid (\varphi \wedge \varphi) \mid (\varphi^\frown\varphi) \mid \varphi^*$$

The primitive formulas in this class are either a phase of a state, or a simple constraint on time.

From the work of Asarin, Caspi and Maler [1], the expressive power of this class is equivalent to the class of timed automata with possible renaming of states. So, given a timed automaton A we can easily construct a formula $D(A)$ in this class to express the behaviour of A in the sense that any observation of A in an interval corresponds intuitively to a model of $D(A)$ and vice-versa. If the Büchi acceptance condition is used for A then $D(A)$ will be a conjunction of a simple DC formula and a formula in Neighbourhood Calculus to express the fairness of acceptance states. For simplicity, we do not apply the Büchi acceptance conditions here. The class of real-time properties to be checked is of the form of *linear duration invariants* [8]:

$$\Box(a \leq \ell \leq b \Rightarrow \textstyle\sum_{k=1}^{n} c_k \int s_k \leq M)$$

The requirement of the simple gas burner and the performance requirement of the double tank systems mentioned earlier in this paper are of this form. The model checking techniques developed by us are to decide automatically whether $D(A) \Rightarrow R$, where R is a linear duration invariant expressing the safety requirement of a real-time system. The fact $D(A) \Rightarrow R$ says that any observation of A in an interval of time corresponds to a model of R. So, R is satisfied by A.

The results presented in [8, 12, 13] can be summarised as follows: If there is no conjunction occurring in $D(A)$ then we can construct a finite set $\mathcal{P}$ of linear programming problems such that $A \models R$ if and only if the maximal value of any problem in $\mathcal{P}$ does not exceed M.

When conjunction is allowed in $D(A)$, checking $A \models R$ involves some mixed integer linear programming problems. It has been shown in [40] that in most cases, we can avoid mixed integer linear programs.

Even if the linear programming techniques are acceptable in practice, the above method could result in a huge number of programming problems in which some problems have very big size (with a large number of variables). To reduce further the complexity, we have to restrict ourselves to a more specific class of real-time properties. In [28, 53], we propose two other classes for the requirement R:

$$\Box\lceil S \rceil^\frown\lceil \neg S \rceil^\frown\lceil S \rceil \Rightarrow \textstyle\sum_{k=1}^{n} c_k \int s_k \leq M, \text{ and}$$
$$\Box\lceil s_1 \rceil^\frown\lceil s_2 \rceil^\frown \ldots ^\frown\lceil s_m \rceil \Rightarrow \textstyle\sum_{k=1}^{m} c_k \int s_k \leq M$$

An interesting property of this class is that $D(A) \Rightarrow R$ is satisfied by all the timed behaviours of A if and only if it is satisfied by all integer behaviours of A (corresponding to the DC models in which any state can only change its value at an integer). Hence, we can use exhaustive depth-first search in the region graph of the timed automaton A to decide on the problem. The complexity of the algorithm is in the same class as for the reachability problem of timed automata. For the second case of the requirement R, we can convert the problem of checking the validity of $D(A) \Rightarrow R$ to a finite set of integer problems, and because of the above results, they can be solved with linear programming techniques. One should notice here that the size of the linear programming problems to be solved in this case is bounded by m, and the number of the problems is the same as the number of the paths in the region graph that matches $[\![s_1]\!]^\frown[\![s_2]\!]^\frown \ldots ^\frown[\![s_m]\!]$.

3.2 Model-Checking Discrete DC

As it was mentioned earlier, the model checking problem we are concerned with can be converted into the validity checking problem. Although DC is highly undecidable for continuous time, it is mostly decidable for discrete time. In [18], a set of algorithms has been developed for checking the validity of a wide class of DC formulas for discrete time. Paritosh Pandya, in a long term cooperation with UNU/IIST staff and fellows, has developed a DC validity checker called DC-Valid to check the validity of discrete time DC formulas. This tool has been used to formally verify the correctness of a multimedia protocol [29]. We also developed some algorithms (different from and more comprehensive than that of Martin Frankzle's works [36]) to check if a DC formula is satisfied by all integer models (called synchronous models) [46].

4 Designing Real-time Systems with Duration Calculus

In parallel with the development of the formal calculi to satisfy the need in specification and reasoning, we have developed some techniques for designing real-time embedded systems, and apply them to some practical case studies. We have also developed some techniques for integrating Duration Calculus with other formalisms like Timed CSP [34], Phase Transition Systems [44], hardware description languages like Verilog [33, 31] and RAISE [14, 35]. In the work [26, 48, 47], we have developed a technique for discretisation of time, and for deriving a real-time program from a specification in Duration Calculus. The work [30] presents a technique to refine a formula for continuous time into a formula in discrete time. Due to the limitation of space, we show here only a case study on our discretisation technique.

4.1 Specification and Verification of Biphase Mark Protocols in DC

We present in this subsection a case study on the specification and verification of the Biphase Mark Protocol (BMP) described in [21]. There have been several

papers presenting methods for formal specification and verification of BMP, e.g. [37]. However, the model in that paper is too abstract, and does not give the concrete relations on parameters. We need a natural way to specify the Biphase Mark Protocol with more detailed physical assumptions and higher accuracy.

The BMP protocol encodes a bit as a cell consisting of a mark subcell and a code subcell. If the signal in the mark subcell is the same as the one in the code subcell then the information carried by that cell is 0. Otherwise, the information carried is 1. There is a phase reverse between two consecutive cells, i.e. the signal at the beginning of the following cell is held as the negation of the signal at the end of the previous cell. The receiver detects the beginning of a cell by recognising a change of the signals received, which is called an *edge*. The receiver, after having detected the beginning of a cell, skips some cycles called sampling distance and samples the signal. If the sampled signal is the same as the signal at the beginning of the cell, it decodes the cell as 0; otherwise it decodes the cell as 1. Let b be the length in time of a mark subcell, a the length in time of a code subcell and d the sample distance. We model the sender's signals by the state variable X and receiver's signals by the state variable Y. Since we use the sender's clock as the reference for time, the state X is *discrete*. Suppose that the signals sent are unreliable for r cycles ($r < a$ and $r < b$) after a change. Without loss of generality, we assume that the delay between the sender and the receiver is 0.

Suppose the time length between two consecutive ticks of the receiver's clock is Δ. The parameters b, a, r and Δ must satisfy certain constraints to guarantee conditions: (1) the receiver recognises the edges, and (2) the receiver samples the belief codes. We suggest the following constraints: (C1) $b - r \geq 2\Delta$, $a - r \geq 2\Delta$, and (C2) $d \geq \lceil (b + r)\theta \rceil + 1$, $d \leq \lceil (b + a - r)\theta \rceil - 3$, where $\theta = \Delta^{-1}$ and $\lceil x \rceil$ is the least integer greater than or equal to x.

Our discretisation technique is essentially a formalisation of the relationship between the discrete state X and the continuous state Y in DC:

$$(\mathcal{A}1)\ \Box(\lceil X \rceil \wedge \ell > r + \delta \Rightarrow (\ell \leq r + \Delta)^\frown \lceil Y \rceil)$$
$$(\mathcal{A}2)\ \Box(\lceil \neg X \rceil \wedge \ell > r + \delta \Rightarrow (\ell \leq r + \Delta)^\frown \lceil \neg Y \rceil)$$

Let $\mathcal{L}_{DC}$ be the language of DC. For each $\omega \in \{0,1\}^+$, let the DC formula $f(\omega)$ represent the coding of ω by BMP as a DC formula to characterise the behaviour of the state X in the time interval for sending ω. Let $g(\omega)$ be a DC formula representing the behaviour of the state Y for the received signals that results in ω after decoding by BMP. It is not difficult to give a definition of the encoding functions f and g to formalise the BPM protocol. Readers are referred to [26, 39] for the details of the definition.

The protocol will be correct if for all $\omega \in \{0,1\}^+$, $f(\omega)$ implies $g(\omega)$ in DC, and for all $\omega, \omega' \in \{0,1\}^+$ such that $\omega \neq \omega'$, we have $\neg(g(\omega) \wedge g(\omega'))$ holds.

A formal proof of the correctness of BMP with the PVS based Duration Calculus proof checker has been carried out in [22] on the assumption ($\mathcal{C}1$) and ($\mathcal{C}2$). The constraints ($\mathcal{C}1$) and ($\mathcal{C}2$) are a basis for reasoning about the values of parameters in the design of BMP. For example, let $b = 5$, $a = 13$ and $r = 1$. If

we choose $d = 10$, the allowed ratio of clock rates is within 33%. If the ratio of clock rates is given to be within $\alpha = 10\%$, the allowed value of d is in between 8 and 13.

4.2 Some Other Case Studies

Besides the modelling and mechanical verification of the Biphase Mark Protocol mentioned above, Zheng Yuhua and Zhou Chaochen have given the first formal verification of the Earliest Deadline Driven Scheduler [54]. Later on, the techniques have been generalised for more general kinds of schedulers by Philip Chan and Dang Van Hung [3], and by Xu Qiwen and Zhan Naijun [45]. Some other case studies about the development of hybrid systems have been also carried out in the project. In [52], Zhou Chaochen and He Weidong have shown that DC can be used to specify and prove the correctness for the optimal controller TMC (time minimal controller). In [50] Hong and Dang have shown how DC can be used together with control theory for the formal development of hybrid systems. We have tried to use DC for some other kinds of real-time systems like real-time databases [43]. This research is still going on.

5 Research on DC in Other Institutions and Conclusion

To our knowledge, DC has been used and developed further in some other universities such as the Technical University of Denmark, the University of Oldenburg, Germany and the Tata Institute for Fundamental Research. Some UNU/IIST ex-fellows and visitors also carry out projects in their home institutions to continue their works at UNU/IIST such as Li Xuandong, Wang Hanpin, Victor A. Braberman. Some works on DC carried out in the University of Oldenburg are summarised as follows.

In [11] Henning Dierks et al. have considered Programmable Logic Controller (PLC) Automata as an implementable model for design. This model is widely used for hardware design. A PLC automaton is an I/O automaton working in cycles. On entering a state, the automaton starts the polling phase. In this phase, it waits for inputs and then decides a polled input. Depending on the polled input, it moves to the next state after broadcasting its output. There is a time bound for the polling phase. He has given a DC semantics and a timed automata semantics for the PLC automata as their denotational and operational semantics. A tool has been developed [49] to design a PLC automaton from a DC specification and to verify it against some DC properties.

We have given a brief description of the main achievements of UNU/IIST in the project "Design Techniques for Real-Time Systems" using Duration Calculus. Our techniques seem to be simple and cover all important aspects of the development of the real-time systems. In dealing with some special aspects of systems such as discretisation, synchrony hypothesis, and recursion, some small extensions to the calculus have been developed and these seem to be inevitable.

However, these extensions are conservative and intuitive which make them easily acceptable by the developers. From many case studies, we strongly believe that our techniques for the formal development of real-time systems are not only powerful but also promising in practice. We understand that in order to put our techniques into practice, more effort has to be paid to develop some efficient tools to support design and verification, and to develop a language, methods and rules that are intuitive and easy enough to be accepted by engineers and programmers.

Because of the limitation of the space, we cannot describe here all of our achievements. We refer the readers to the web page of the UNU/IIST reports www.iist.unu.edu/newrh/III/1/page.html for the materials that detail our work.

References

1. E. Asarin, P. Caspi, and O. Maler. A Kleene Theorem for Timed Automata. In G. Winskel, editor, *International Symposium on Logics in Computer Science LICS'97*, pages 160–171. IEEE computer Society Press, 1997.
2. Rana Barua and Zhou Chaochen. Neighbourhood Logics : NL and NL^2. Presented at ComBaS Group, Technical University of Denmark, 4–7 September, 1997.
3. Philip Chan and Dang Van Hung. Duration Calculus Specification of Scheduling for Tasks with Shared Resources. LNCS 1023, Springer-Verlag 1995, pp. 365–380.
4. Zhou Chaochen, Dimitar P. Guelev, and Zhan Naijun. A Higher-Order Duration Calculus. Published in: the proceedings of the Symposium in Celebration of the Work of C.A.R. Hoare, Oxford, 13-15 September, 1999.
5. Zhou Chaochen and Michael R. Hansen. An Adequate First Order Interval Logic. *International Symposium, Compositionality – The Significant Difference*, Hans Langmaack et al. (eds), Springer–Verlag, 1998.
6. Zhou Chaochen, C.A.R. Hoare, and Anders P. Ravn. A calculus of durations. *Information Processing Letters*, 40(5):269–276, 1991.
7. Zhou Chaochen, Dang Van Hung, and Li Xiaoshan. A Duration Calculus with Infinite Intervals. *Fundamentals of Computation Theory*, Horst Reichel (ed.), pp. 16–41, LNCS 965, Springer-Verlag, 1995.
8. Zhou Chaochen, Zhang Jingzhong, Yang Lu, and Li Xiaoshan. Linear Duration Invariants. LNCS 863, Springer-Verlag, 1994.
9. Zhou Chaochen, Hansen Michael R., and Sestoft P. Decidability and Undecidability Results in Duration Calculus. LNCS 665. Springer-Verlag, 1993.
10. Zhou Chaochen and Li Xiaoshan. A Mean Value Duration Calculus. *A Classical Mind*, Festschrift for C.A.R. Hoare, Prentice-Hall International, 1994, pp. 432–451.
11. Henning Dierks, Ansgar Fehnker, Angelika Mader, and Frits Vaandrager. Operational and Logical Semantics for Polling Real-Time Systems. *LNCS* 1486, pages 29–40, Springer-Verlag, 1998.
12. Li Xuan Dong and Dang Van Hung. Checking Linear Duration Invariants by Linear Programming. LNCS 1179, Springer-Verlag, Dec 1996, pp. 321–332.
13. Li Xuan Dong, Dang Van Hung, and Zheng Tao. Checking Hybrid Automata for Linear Duration Invariants. LNCS 1345, Springer-Verlag, 1997, pp. 166–180.
14. Chris George and Xia Yong. An Operational Semantics for Timed RAISE. LNCS 1709, Springer-Verlag, 1999, pp. 1008–1027.
15. Dimitar P. Guelev. A Complete Fragment of Higher-Order Duration μ-Calculus. LNCS 1974, Springer-Verlag, 2000, pp. 264–276.

16. Dimitar P. Guelev. Probabilistic Neighbourhood Logic. LNCS 1926, Springer-Verlag, 2000, pp. 264–275.
17. Dimitar P. Guelev and Dang Van Hung. Prefix and Projection onto State in Duration Calculus. *Electronic Notes in Theoretical Computer Science*, Volume 65, Issue 6, Elsevier Science 2002
18. Michael R. Hansen. Model-checking Discrete Duration Culculus. *Formal Aspects of Computing*, 6(6A): 826–845, 1994.
19. Michael R. Hansen and Zhou Chaochen. Chopping a point. In *BCS-FACS 7th refinement workshop*, Electronics Workshops in Computing. Spriger-Verlag, 1996.
20. Michael R. Hansen and Zhou Chaochen. Duration calculus: Logical foundations. *Formal Aspects of Computing*, 9:283–330, 1997.
21. Zhu Huibiao and He Jifeng. A DC-based Semantics for Verilog. Published in the proceedings of the ICS2000, Yulin Feng, David Notkin and Marie-Claude Gaudel (eds), Beijing, August 21-24, 2000, pp. 421–432.
22. Dang Van Hung. Modelling and Verification of Biphase Mark Protocols in Duration Calculus Using PVS/DC$^-$. The proceedings of CSD'98, 23-26 March 1998, Aizu-wakamatsu, Fukushima, Japan, IEEE Computer Society Press, 1998, pp. 88 - 98.
23. Dang Van Hung. Projections: A Technique for Verifying Real-Time Programs in Duration Calculus. Technical Report 178, UNU/IIST, Macau, November 1999.
24. Dang Van Hung and Zhou Chaochen. Probabilistic Duration calculus for Continuous Time. *Formal Aspects of Computing* (1999) 11: 21–44.
25. Dang Van Hung and Dimitar P. Guelev. Completeness and Decidability of a Fragment of Duration Calculus with Iteration. LNCS 1742, Springer-Verlag, 1999, pp. 139–150.
26. Dang Van Hung and Ko Kwang Il. Verification via Digitized Model of Real-Time Systems. Published in the proceedings of *Asia-Pacific Software Engineering Conference 1996* (APSEC'96), IEEE Computer Society Press, 1996, pp. 4–15.
27. Dang Van Hung and Wang Ji. On The Design of Hybrid Control Systems Using Automata Model. LNCS 1180, Springer-Verlag, Dec 1996, pp. 156–167.
28. Zhao Jianhua and Dang Van Hung. Checking Timed Automata for Some Discretisable Duration Properties. *Journal of Computer Science and Technology*, Vol. 15, No. 5, September 2000, pp. 423–429.
29. Wang Jianzhong, Xu Qiwen, and Ma Huadong. Modelling and Verification of Network Player System with DCValid. The proceedings of the First Asia-Pacific Conference on Quality Software, Hong Kong, October 2000, IEEE Computer Society Press, pp. 44–49.
30. He Jifeng. A behavioural Model for Co-design. LNCS 1709, Springer-Verlag, 1999, pp. 1420–1439.
31. He Jifeng. An Integrated Approach to Hardware/Software Co-design. Published in the proceedings of the ICS2000, Yulin Feng, David Notkin and Marie-Claude Gaudel (eds), Beijing, August 21-24, 2000.
32. He Jifeng and Xu Qiwen. Advanced Features of DC and Their Applications. The proceedings of the Symposium in Celebration of the Work of C.A.R. Hoare, Oxford, 13-15 September, 1999.
33. He Jifeng and Xu Qiwen. An Operational Semantics of a Simulator Algorithm. The proceedings of the PDPTA'2000, Las Vegas, Nevada, USA, June 26-29, 2000.
34. He Jifeng and Viktor Verbovskiy. Integrating CSP and DC. Published in the proceedings of the 8th IEEE International Conference on Engineering of Complex Computer Systems, Maryland, USA, 2–4 December 2002.
35. Li Li and He Jifeng. Towards a Denotational Semantics of Timed RSL using Duration Calculus. Chinese Journal of Advanced Software Research, 2000.

36. Franzle Martin. Synthesizing Controllers of Duration Calculus. LNCS 1135, pages 168–187, Springer-Verlag, 1996.
37. J S. Moore. A Formal Model of Asynchronous Communication and Its Use in Mechanically Verifying a Biphase Mark Protocol. FAC, 6:60–91, 1994.
38. Zhan Naijun. Completeness of Higher-Order Duration Calculus. The proceedings of CSL2000, Fischbachau, Munich, Germany, 22–27 August, 2000.
39. Do Van Nhon and Dang Van Hung. A Systematic Design of Real-time Systems Using Duration Calculus. Published in the proceedings of the conference SCI 2001, Orlando, USA, July 22-25, 2001, IEEE Computer Society Press, pp. 241–246.
40. Paritosh K. Pandya and Dang Van Hung. Duration Calculus with Weakly Monotonic Time. LNCS 1486, pp. 55–64, Springer-Verlag, 1998.
41. Paritosh K. Pandya and Y Ramakrishna. A Recursive Duration Calculus. Technical Report CS-95/3, TIFR, Mumbai, 1995.
42. P.K. Pandya, Wang Hanpin, and Xu Qiwen. Towards a Theory of Sequential Hybrid Programs. The proceedings of PROCOMET'98, 8-12 June 1998, Shelter Island, New York, USA, David Gries and Willem-Paul de Roever (eds), Chapman & Hall, 1998, pp. 366–384.
43. Ekaterina Pavlova and Dang Van Hung. A Formal Specification of the Concurrency Control in Real-Time Databases. In the proceedings of APSEC'99, December 7-10, Takamatsu, Japan. IEEE Computer Society Press, 1999, pp. 94–101.
44. Xu Qiwen. Semantics and Verification of the Extended Phase Transition Systems in the Duration Calculus. LNCS 1201, Springer-Verlag, 1997, pp. 301–315.
45. Xu Qiwen and Zhan Naijun. Formalising Scheduling Theories in Duration Calculus. Technical Report 243, UNU/IIST, P.O. Box 3058, Macau, October 2001.
46. Manoranjan Satpathy, Dang Van Hung, and Paritosh K. Pandya. Some Results on The Decidability of Duration Calculus under Synchronous Interpretation. LNCS 1486, pp. 186–197, Springer-Verlag, 1998.
47. François Siewe and Dang Van Hung. From Continuous Specification to Discrete Design. Published in the proceedings of the ICS2000, Yulin Feng, David Notkin and Marie-Claude Gaudel (eds), Beijing, August 21-24, 2000, pp. 407–414.
48. François Siewe and Dang Van Hung. Deriving Real-Time Programs from Duration Calculus Specifications. LNSC 2144, Springer-Verlag, 2001, pp. 92–97.
49. Josef Tapken and Henning Dierks. MOBY/PLC - Graphical Development of PLC-Automata. LNCS 1486, pages 311–314, Springer-Verlag, 1998.
50. Hong Ki Thae and Dang Van Hung. Formal Design of Hybrid Control Systems: Duration Calculus Approach. The proceedings of COMPSAC 2001, October 8–12, Chicago, USA, IEEE Computer Society Press, 2001, pp. 423–428.
51. Yde Venema. A modal logic for chopping intervals. *Journal of Logic Computation*, 1(4):453–476, 1991.
52. He Weidong and Zhou Chaochen. A Case Study of Optimization. *The Computer Journal*, Vol. 38, No. 9, British Computer Society, pp. 734–746, 1995.
53. Li Yong and Dang Van Hung. Checking Temporal Duration Properties of Timed Automata. *Journal of Computer Science and Technology*, 17(6):689-698, 2002.
54. Zheng Yuhua and Zhou Chaochen. A Formal Proof of a Deadline Driven Scheduler. LNCS 863, 1994, pp. 756–775.
55. Liu Zhiming, Sørensen E. V., Ravn P. Anders, and Zhou Chaochen. Towards a calculus of systems dependability. *Journal of High Integrity Systems*, 1(1):49–65, 1994. Oxford Press.
56. Chen Zongji, Wang Ji, and Zhou Chaochen. An Abstraction of Hybrid Control Systems. Research Report 26, UNU/IIST, Macau, June 1994.