

Specification of Real-Time System
Using Duration Calculus in Z

by

Man-Wai CHANG

(Third Revision)

Abstract

This paper aims to augment model-based formal specification languages Z and VDM with real-time semantics through the use of duration calculus. It will begin with a discussion of traditional specification methods for real-time systems and their shortcomings. A brief introduction to duration calculus will be given. Examples for both hard and soft real-time systems will be used to illustrate the use of the augmented Z and VDM.

1. Real-time Systems

1.1. A Historical Perspective

The concept of real-time system began to attract attention in the data processing field back in 1950's [12]. At that time the general perception of real-time system was a data processing system with a processing cycle determined by the user's deadlines for receiving the system outputs. Nevertheless, at that time, the length of period before the deadline was measured in hours, rather than seconds as commonly found in contemporary real-time systems.

There are many ways of classifying real-time system. At the very high level, real-time system can be grouped into hard and soft real-time system [9]. Soft real-time system refers to online business computer systems. The functions of hard real-time system can be classified into two categories: process control and operations control [12]. Process control systems make use of automated environment sensors and control actuators to monitor and affect the environment according to well-know mathematical laws. Examples include power plant monitoring system and railway switching system. Operations control are employed to improve the performance of government and business operations. Examples include air-traffic control, flight reservation system and others.

1.2. Issues in Engineering Real-time System

In general, a real-time system has the following characteristics:

- Requirement that the aggregate of all sub-processes satisfy a critically defined deadline
- Assumption that the critically timed process within the real-time system produces control outputs
- Requirements for continuous system availability, implying some sort of backup equipment

In particular, Jonathan [22] has addressed certain important problems relevant to timing constraints:

- The system have to be adjusted at a high rate, the upper bound, to prevent catastrophic destruction, while lower bounds are needed in operating systems to keep an intruder waiting for some minimum time after a false password. A good example is the job scheduler in distributed systems [20]
- The non-faulty processors must arrive at a consensus to perform some action in the presence of other processors that can exhibit faulty behaviour. Good examples are leader election in distributed systems [23].
- In a communication protocol, the receiver guarantees the sender that it will be able to process, packets at a certain rate. A good example is the synchronization protocol found in multimedia conference systems [24].

1. Functional Requirements

Specification of real-time system must begin with the description of the function of system. An obvious approach is to use state-based algebra to record the temporal properties of the real-time.

2. Timing Requirements

The timing problem with real-time system can be studied under three headings [26]:

System response:	The real-time system is to keep performance within promptness requirements. This can be a matter of algorithms, as well as a matter of physical limitations of the underlying hardware.
Event processing:	The real-time system is to detect frequency variable in the external environment. This response mechanism of the real-time system will depend on the frequency of occurrence of the events, the maximal number of events to be buffered and the worst case average rate.
State processing:	The real-time system is to maintain an adequate sampling rate to determine the amplitude variation in the environment. The system can operate in cycles of self-timed or periodic manner.

In this paper, we will group the timing constraints found commonly in real-time system into two categories:

- i) Periodic: some action must be executed repeated at fixed interval
- ii) Sporadic: an action must be executed following some triggering event, within a specified deadline.

1.3. Traditional Techniques for Real-time System

Ever since the appearance of real-time system, many scholars have introduced various specification techniques to specify real-time system. Here is a brief of the techniques involved:

- Petri-Nets
- State-transition diagrams
- Temporal Logic

Temporal logic uses a set of symbols for the individual variables and propositions. It includes the data variables $y = y_1, \dots, y_n$, and the control variables $r = r_1, \dots, r_m$, as well as other logical variables and propositions. Functions and predicates are also used in our specifications.

A state formula is any formula of first order logic constructed

1.4 Formal Methods

A method is formal if it has a sound mathematical basis, typically given by a formal specification language [16]. This allows the developer to provide the means of precisely defining notions like consistency, completeness and correctness. On the other hand, as a formal notations can be analyzed and manipulated using mathematical operators, mathematical proof procedures can be used to test the internal consistency and syntactic correctness of the specifications. Finally, a formal method can avoid ambiguities and misconceptions in implementation of systems. The advantages of using formal methods were discussed in detail in [1].

Formal method can be defined concisely as follows:

Definition: A formal specification language is a triple $\langle Syn, Sem, Sat \rangle$, where Syn and Sem are sets and $Sat \subseteq Syn \times Sem$ is a relation between them. Syn is called the language's syntactic domain, Sem , its semantic domain, and Sat , its satisfies relation.

Definition: Given a specification language, $\langle Syn, Sem, Sat \rangle$, if $Sat(syn, sem)$ then syn is a specification of sem , and sem is a specificand of syn .

Definition: Given a specification language, $\langle Syn, Sem, Sat \rangle$, the specificand set of a specification syn in Syn is the set of all specificands sem in Sem such that $Sat(syn, sem)$.

In other words, a formal specification language provides a notation for its syntactic domain Syn , a universe of objects Sem , and a precise rule defining which objects satisfy each specification Sem . On the other hand, a specification is a sentence made up of elements of the syntactic domain, which denotes a specificand set, a subset of the semantic domain. A specificand is an object satisfying a specification. Sat , the satisfies relation provides the interpretation for the syntactic elements.

Most previous work on formal specification of real-time system uses absolute time values represented by natural numbers, real number or time intervals. Future work on the specification of distributed systems which lack a central time-base will see a role for partially ordered representation of time.

Traditionally, temporal logic, and in particular, its application to the specification and analysis of reactive and concurrent systems, have been considered appropriate for the qualitative treatment of time. This can be seen by some of the abstractions that are inherent in temporal methodology.

To mention some of them, the concept of eventuality, which guarantees an eventual occurrence of an event but provides no bound on how soon; the notion of fairness which requires that frequent attention be directed to certain components of the system, but places no bound on the frequency. All these represent conscious efforts to deal with essentially quantitative phenomena in a qualitative way.

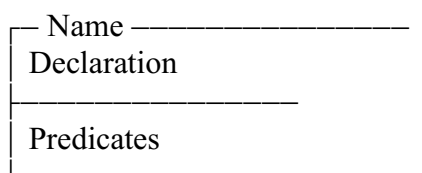
However, there are many systems and applications, specifically in the area of reactive system, for which purely qualitative specification and analysis are inadequate. An example is a requirement like "every p should be followed by a q, within no more than 2 seconds", which mentions real time explicitly. Another example would be "no two processes may attempt to write on the channel at the same time". In both cases, we require a logic that has the capabilities of specifying and verifying reactive system, similarly to temporal logic, as well as the ability to express and reason about real time. That explains the arrival of interval temporal logic and duration calculus.

2. Background

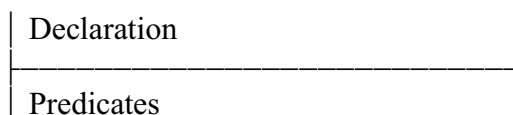
2.1. Z Notations

Z is another specification language based on set theories suitable for defining, and reasoning about a wide range of systems. It was originally developed at Oxford University, and has been used in non-trivial "real-world" projects. A very good example was IBM's effort of using Z to re-design CICS [*** IBM Experience with Z]. It is particularly suited to specify non-constructive requirements specification. Semantics-preserving characteristic of Z allow formal translation of Z specifications to executable code, and formal proof techniques are also well established.

The Z notation consists of two components: the schema language, a specification structuring technique, and a mathematical language. A Z schema is usually represented as named box partitioned into two parts, variable declarations and optional predicates relating the variables.



Unnamed axiomatic definitions use a similar notation:



A number of operators are available for composing schemata. The declarations and predicates in a schema may be "imported" into another by including the schema name in the declaration part. By convention such a name preceded by Δ denotes that the operation may change the values of variables in the imported schema.

When a Z specification is interpreted operationally, three uses of schema boxes are normally distinguished: declaration of global variables, definition of the effect of performing certain operations, and definition of the initial system states.

Each operation schema defined the effect of an operation by describing the pre-condition (on the global state) required for the operation to occur, and the post-condition after its occurrence. By convention a primed variable name represents the value of the variable after the operation has been performed and an unprimed name represents the value before.

The set-theoretic mathematical notations used within schema boxes includes a vast selection of operations. Relations, functions, sequences and bags are represented by sets of ordered pairs which can be manipulated via the set operators.

2.2. Time and States

The most important addition when switching to real-time is the ability to measure time. [***Time: a philosophy]. Time measurement has to do with a quantitative notion of time. Not only the question whether a time element is earlier than another time element is to be answered, but also the question how much earlier. Several issues can be investigated in the context of time measurement such as:

- How should time elements be represented? Should it be cited explicitly in the syntax of the language or implicitly in the semantics of the language?
- How should measured be presented? Should it be additive by turning the time domain into a group or metric by adding a distance function like in topology?
- How is time measurement calibrated? Should be it absolute or relative?

An example of a formalism using relative time reference is provided by classical temporal logic because it uses an implicit notion of time based on the current moment of time (now).

There are three fundamental choice of time elements: points, periods, and events. When considering points and events to be the basic time elements, we only need a precedence relation $(T, <)$ and $(E, <)$. The precedence relation $<$ is also referred to by 'before'. For period structures, we need apart from the precedence relation also an inclusion relation $(I, \sqsubseteq, <)$. The inclusion relation \sqsubseteq is also referred to by 'during'. Since we will be focusing on duration calculus, our discussion will limit to the use of periods or durations as the basic time element.

2.3. Limitations of Z

Although Z can be used to model real-time requirements, there is little agreement on the best way to do this. In fact, both Z fails to offer any guidance on how to tackle a given specification problem, but rather, a way of describing the problem.

A number of real-time system has been specified in Z. Raymond et al. (1990) introduce a global auxiliary integer time variable and a special "tick" operation to effect the passage of time. King (1989) expressed via temporal logic assertions on the auxiliary variables. Coombes (1990) defines a number of interval time operators, and uses these to define functions from time intervals to values for each variable, representing the change of value of variables with time. No single approach has yet asserted itself as the best starting point for defining reasoning about real-time behaviour in Z. However, by augmenting Z and VDM with basic duration calculus notations, we intend to provide such a discipline for operational formal specifications.

1. Z operations are atomic and only one can occur at a time. No facility is provided for specification of concurrency and time constraints.
2. Z makes no formal distinction between internal and external operations -- if the pre-condition of an operation is satisfied it is free to occur, but particular entity is assumed to initiate it.

This paper aims to augment the state-based (or more correctly model based) formal methods Z with duration calculus. Duration calculus provides a highly comprehensive method of specifying the time constraints, and relative ordering of event occurrence in real-time system. To obtain a theory of timed state sequence, we need to identify a suitable time domain with appropriate primitives, and mix the theory of state transition with the theory of time through a unary "time" function which associates a time with every state.

2.4. Duration Calculus

Real-time systems are reactive systems which must satisfy timing properties. Temporal logic, both linear-time and branching-time versions, have been widely used to specify and reason about reactive systems. A linear-time temporal logic describes and verifies the properties of state sequences which are semantic denotations of program executions. A branching-time temporal logic, on the other hand, is used to reason about state trees which describe the non-deterministic behaviour of a program for a given input. Later, linear temporal logic have been used to define interval temporal logic in which intervals (in computations) and their properties can be specified and manipulated.

Little attention has been paid to the possibility of extending interval temporal logic for specifying quantitative temporal properties. As most real-time properties hold in specific intervals in a computation, a calculus of such properties can be embedded in an appropriate interval temporal. Recently, Zhou Chaochen et al [1] shown that many of the requirements of time-critical system can be expressed and reasoned about very effectively in a calculus of durations based on an interval temporal logic.

Duration Calculus is an extension to interval temporal logic where one can reason about integral of propositional states, i.e. about their durations within a given bounded and closed interval. The formulas of duration calculus are built from state variables. Composite states are formed by propositional combinations of state variables. A brief introduction to duration calculus is given below [21].

If S is a composite state then $\int S$, the duration of S , denotes a real-valued interval function. For a given interval $[b, e]$ of a given computation, the value of $\int S$ is

$$\sum_b^e S(t) dt$$

The point interval and duration formula are defined as follows:

$$\begin{aligned} [\cdot] &\triangleq (l = 0) \\ [S] &\triangleq (\int S = l) \wedge (l > 0) \end{aligned}$$

For a given interval $[b, e]$, the duration formula $[S]$ holds if $b < e$ and the value of $\forall t \in [b, e] \square S(t) = l$. Composite duration formula are formed using conventional logic operators. For duration formulas A and B , "A followed by B" (written $A \wedge B$) holds for given computation and intervals $[b, e]$ iff there is a m such that $b \leq m \leq e$, A holds for the initial interval $[b, m]$, and B holds for the final interval $[m, e]$.

The modal operator \diamond (there is a sub-interval) and \square (for any sub-interval) in temporal logic can be re-defined using the duration formula:

$$\begin{aligned} \diamond D &\triangleq \text{true} \wedge D \wedge \text{true} \\ \square D &\triangleq \neg (\diamond \neg D) \end{aligned}$$

Any interval $[0, t]$ is called a prefix interval of $[0, \text{infinity}]$.

2.5. Mixing Z with Basic Duration Calculus

As mentioned before, Z is a state-based specification language that lacks notations to specify timing requirements found in real-time systems.

3. Examples

In this section, four example real-time system are to be specified using the augmented Z and VDM. The first three are hard real-time systems which have tight timing constraints, while the last one is a soft real-time system which requires a looser timing requirements.

3.1. Gas Burner

In his paper on duration calculus [1], Zhou used a gas burner example to explain methods of using duration calculus in specification of a simple real-time system. Below is an alternative specification of the same problem using Z and VDM.

3.2. Synchronization in Real-Time System

3.3. A scheduler for operating system

Zhou has shown a way of specification synchronization between shared processors [21]. Inter-process communication is achieved through channels, which could be considered as the extension of C.A.R. Hoare's CSP. Zhou introduced a set of duration calculus notations to represent operating system primitives like WAIT, SEND_MESSAGE, and RECEIVE_MESSAGE. Channel and process scheduling is achieved by mixing the primitives with a state machine for the entities involved.

In Lister's book [17], he introduced a scheduling mechanism for operating systems. The informal descriptions follow:

3.4 Commercial Applications

Despite of the fact that time in general is not a critical requirements for most commercial applications, duration calculus can still be applied to commercial systems. Duration calculus does not specify the absolute value of the duration $/P$. In this section, I will use duration calculus and Z to specify the automatic renewal function in a general insurance system I have been working on.

The general insurance system needs to keep track of policy contracts whose states change with time. Since we are only interested in the states relevant to the time constraints, the whole picture of policy state changes will not be discussed here.

The informal requirement for policy renewal is:

Renewal notice is to be issued 30 days before the policy expired. If the policy holder does not response, a renewal reminder will be issued 15 days later. The policy will be expired automatically should the policy holder fails to confirm the renewal one day after the policy expiration date. The policy will be in-force again after the policy holder confirmed the renewal by signing either the renewal notice or the renewal reminder.

The state-transition diagram is shown below:

To specify this requirements formally, a boolean data type called PolicyState is introduced.

$$\text{State} ::= \{ \text{InForce}, \text{Expired}, \text{RenewalNotice}, \text{RenewalReminder} \}$$

The augmented Z specification for this problem follows:

5. Conclusion

We have given a formal definition of duration calculus concepts as they apply to specification in Z. This exercise has produced a concise notation for expressing temporal requirements in Z specifications. With these formal definitions in place an approach to verification of real-time properties was explored. In this paper, no coverage has been made on the semantics of the augmented Z notations in specification of real-time systems. Nevertheless, there is work going relating to this topic [25].

Bibliography

- [1] Zhou Chaochen, C.A.R.Hoare, and A.P.Ravn: A Calculus of Durations, In *Information Processing Letters* 40(5), 1992, pp. 269-272.
- [2] K.M. Hansen, A.P. Ravn, and H. Rischel: Specifying and Verifying Requirements of Real-time Systems. In proc. of the ACM SIGSOFT'91 Conference on Software for Critical Systems, New Orleans, December 4-6, 1991, *ACM Software Engineering Notes*, vol. 15, 1991, pp. 44-64.
- [3] A. Pnueli and E. Harel, Application of Temporal Logic to the specification of real-time systems, in: M. Joseph, ed., *Proc. Symp. Formal Techniques in Real-Time and Fault-Tolerant Systems*. Lecture Notes in Computer Science 331 (Springer, Berlin, 1988) pp. 84-98.
- [4] William G. Wood, A Specification of the Cat and Mouse Problem, in: *Real-Time: Theories in Practice*. Lecture Notes in Computer Science (Springer, Berlin, 1992) pp. 677-686.
- [5] Ron Koymans, (Real) Time: A philosophical Perspective, in: *Real-Time: Theories in Practice*. Lecture Notes in Computer Science (Springer, Berlin, 1992) pp. 353-370.
- [6] Hayes, Ian, editor, *Specification Case Studies*. Prentice-Hall, Englewood Cliffs, NJ, 1987.
- [7] Iain Houston and Steve King, CICS Project Report: Experience and results from the use of Z in IBM, in: *VDM '91 Formal Software Development Methods*. Lecture Notes in Computer Science 551 (Springer, Berlin, 1991) pp 588-596.
- [8] R.M. Keller, Formal Specification of parallel programs. *Communications of ACM*, 19(7): 371-384, 1976.
- [9] Robert L. Glass, *Real-time Software*, Prentice-Hall, 1983.
- [10] Zohar Manna and Amir Pnueli, *The Temporal Logic of Reactive and Concurrent Systems*, Springer-Verlag, 1992, USA.
- [11] W. Brauer, G. Rozenberg and A.Salomaa, *Temporal Logic of Programs*, Springer-Verlag, 1987, USA.
- [12] J.D. Aron, Real-time systems in perspective, *IBM Systems Journal* 6(1), 1967.
- [13] Cliff B. Jones, *Systematic Software Development Using VDM*, Prentice-Hall International 1986.

- [14] J.M. Spivey, *The Z Notations -- A Reference Manual*, Prentice-Hall International, 1989.

- [15] Martin D. Fraser, Kuldeep Kumar and Vijay K. Vaishnavi, Informal and Formal Requirements Specification Languages: Bridging the Gap, in: *IEEE Transactions in Software Engineering* 17(5), May 1991.
- [16] J.M. Wing, A specifier's introduction to formal methods, *Computer* 23(9), pp 8-24, 1990.
- [17] A.M. Lister, *Fundamentals of Operating Systems*, Third Edition, Macmillan 1984.
- [18] Zhou Chaochen, Duration Calculi: An Overview, *UNU/IIST Report* No. 10, June 1993.
- [19] Susan Owicki and David Gries, "Verifying Properties of Parallel Programs: An Axiomatic Approach", in: *Communication of ACM* 19(5), May 1976.
- [20] Victor Yodaiken and Krithi Ramamritham, "Mathematical Models of Real-Time Scheduling", in: *Foundations of Real-Time Computing: Formal Specification and Methods*, Kulwer Academic Publishers 1991.
- [21] Zhou Chaochen, Michael R. Hansen, Anders P. Ravn and Hans Rischel, "Duration Specification for Shared Processors", in: *Formal Techniques in Real-time and Fault-Tolerant Systems*, Lecture Notes in Computer Science 571, pp 21-32.
- [22] Jonathan S. Ostroff, "Formal Methods for the Specification and Design of Real-time Safety Critical Systems", in: *The Journal of Systems and Software*, April 1992, pp 33-60.
- [23] Gurdip Singh, "Real-time leader election", in: *Information Processing Letters* 49, pp. 57-61.
- [24] Anderson & Honsy, "A Continuous Media I/O Server and its Synchronization Mechanism", *IEEE Computer*, Oct 1991.
- [25] Micheal R. Hansen, and Zhou Chaochen, "Semantics and Completeness of Duration Calculus", in: *Formal Techniques in Real-time and Fault-tolerant Systems*, Lecture Notes in Computer Science 571, pp 210-225.
- [26] Ron Koymans, and Ruurd Kuiper, "Paradigms for real-time systems", in: , M. Joseph, ed., *Proc. Symp. Formal Techniques in Real-Time and Fault-Tolerant Systems*. Lecture Notes in Computer Science 331, pp 160-174.