Formalising Process Scheduling Requirements for an Aircraft Operational Flight Program

Jin Song Dong Neale Fulton Lin Zucconi John Colton

Mathematical and Information Sciences Division
Commonwealth Scientific and Industrial Research Organisation (CSIRO)
Canberra, ACT 2601, Australia
jsdong@cbr.dit.csiro.au

Abstract

Formal methods are considered to be an important technique towards achieving the levels of assurance needed for high integrity systems. Formal specification is the essential part of the formal development process. The use of formal specification techniques on critical systems has shown significant growth in the last few years. In particular, there are number of successful applications of using formal specification techniques in the aviation industry. Safety critical systems, such as aviation systems controlled by software, often have hard real-time requirements. Producing the correct result at the right time is the fundamental goal of such systems. Formally specifying the system functions and the timing requirements is the crucial step towards achieving such a goal. Furthermore, aviation systems often need to be modified or upgraded on a regular basis, i.e. functionality and timing constraints may be altered. Therefore, the formal specification of such systems needs to be easily maintained and modified. In this paper, we are interested in applying formal object modelling techniques to specify scheduling requirements of the multi-parallel processes of an aircraft operational flight program (OFP). Our aim is not only to formalise the scheduling requirements for a particular aircraft, but more importantly to demonstrate an incremental and extendible modelling approach such that our model can be readily reused to specify other aircraft OFP scheduling requirements.

Keywords: formal object modelling, real-time specification, aviation system, Object-Z.

1 Introduction

The essence of formal methods is the application of mathematics and logic to provide assurance of system product quality. Formal methods are well accepted techniques for the development of safety-critical systems. Formal specification is considered to be the essential part of the formal development process. The use of formal methods on critical systems has been growing significantly in the last few years[13, 1, 16]. In particular, there are number of successful applications of using formal specification techniques in the aviation industry. For example, Leveson's group produced a formal requirements specification for the Traffic Collision Avoidance System (TCAS) II by using RSML[15]. The software company Praxis used VDM[14] and CCS[19] to deliver to the UK Civil Aviation Authority the CCF Display Information System (a central part of the new air traffic management system for London's airspace).

Safety critical systems, such as aviation systems controlled by software, usually have hard timing requirements. Producing the correct result at the right time is the fundamental goal of such systems, where formally specifying the system functions and the timing requirements is the crucial step towards achieving such a goal. The lack of an adequate Mission Computer (MC) Operational Flight Program (OFP) timing specification was identified as a major concern for the Canadian Armed Forces[12, 2, 9]. Maintenance of the MC OFP involves the modification of the baseline requirements for the purpose of enhancing performance or adding new capabilities. Therefore maintenance requirements further compounding the specification management problem. In [12, 2, 9], the lack of a precise and adequate MC OFP timing specification was identified as a problem. To address this problem, some mathematics were used in the documentation.

However, the results are not satisfactory because

- there is no uniform notation across the document. Some mathematical terms have different semantics in different sections.
- There is no adequate techniques for describing OFP processes classification.
- The description model of the schedule is unstructured. For example, there is no precise way to describe the synchronisation relationships between the two MC schedulers.

We are interested in applying developed formal object real-time modelling techniques[1] to specify functional and timing requirements of an aircraft OFP with emphasis on a synchronous timing schedule. In this paper, we first use the Object-Z[8, 4] generic class constructs to build a general task scheduling model, then instantiate this generic model to capture the multi-parallel processes scheduling requirements for the OFP. The reason that we chose Object-Z over Z is because the extra object-oriented techniques embedded in Object-Z allow us to produce a more structured scheduling model. Furthermore, the Object-Z inheritance construct is well suited for modelling the classification of the OFP processes.

The aim of this paper is not only to formalise the scheduling requirements for a particular aircraft, but more importantly to demonstrate an incremental and extendible modelling approach such that the formal model can be readily reused to specify evolving design requirements on the OFP schedule and potential application to OFPs in other aircraft.

Section 2 presents a formal generic scheduling model which captures essential scheduling constraints. Section 3 informally describes the multi-parallel processes scheduling requirements for the OFP, then instantiates the generic model of Section 2 to present the formal model of the process scheduling requirements for the OFP. Section 4 concludes the paper and identifies the need for further research.

For those readers not familiar with Object-Z, a glossary of the notation used in this paper, together with a brief introduction to Object-Z is given in an appendix. Further details are given in [8].

2 A Generic Scheduling Model

The major elements of the scheduling model are the system resources and application processes. In brief, the model should consider the following general areas:

[1]In [4, 7], various Z approaches for specifying real-time requirement[10, 18] is integrated into the Object-Z notation[8]

- system resources
- process timing specifications
- process synchronisation constraints

In Hard-Real-Time applications a process can be classified as **periodic** or **demand** (asynchronous) based on the nature of its arrival time[20]. A periodic process must be executed at fixed time intervals. The time between two successive arrivals is referred to as the period of the process. For each periodic process, there is an upper bound on computation time and a deadline for completion with respect to the beginning of the period. The process execution time must be completed within the deadline, where deadline must be less than or equal to the period. A demand process can be invoked at any time and consequently has random arrivals. Demand processes cannot be scheduled statically. However if the number of demand processes is small and their execution time is short, it is possible to transform demand processes into pseudo-periodic processes, thereby making their scheduling possible before run-time.

Process synchronisation constraints include:

- **Precedence Constraints**
  A precedence constraint ensures that a process which produces data for another process will complete before that data is required.
- **Exclusion Constraints**
  An exclusion constraint ensures that if either one of the two processes has started and is not yet finished, the other process cannot be started.

Any two processes on which a precedence or exclusion relation is defined must have the same periodicity.

A schedule can be classified as **preemptive** and **non-preemptive**. Preemption refers to the suspension of one process to permit the execution of another process. In a non-preemptive schedule, once a process has started on a resource it cannot be preempted by another process. A non-preemptive scheduling is choosen because the OFP treats the demand processes as pseudo-periodic processes such that they can be pre-scheduled. The length of a schedule corresponds to the least common multiple (LCM) of the processes' periods of a schedule. Given a process, a schedule solution (outcome) should tell us which time interval is allocated with which resource. The most important requirement constraints that need to be ensured in the scheduling model are:

- **non-collision** : no two processes compete for the same resources at the same time.

- **non-preemption**: once a process has acquired a resource it cannot be preempted by another process until it has released the resource.

As an example, Figure 1 illustrates a schedule model. In the following we will present a formal model of a generic schedule.

Formal Model of a Generic Schedule

The following formal model of a generic schedule is used as a library component to build the model of an aircraft OFP scheduling requirements.

First let

$$\mathbb{T} == \mathbb{N}$$

represent the discrete time domain.

The discrete time model is used for the following reasons:

- An integer increment of 1 is chosen to represent the smallest granularity of time in which we are interested and therefore also represents the precision of representing time within the system. The practical limitation in choosing the finest resolution is usually the period of the central processor clock cycle which in the systems of our interest is one micro-second.
- The aircraft airframe dynamics have periods less than 50 milliseconds so a one micro-second resolution represents a fractional measure of time of 1:50,000
- High precision weaponry may require timing of order 50 micro-seconds so this represents a fractional measure of 1:50, again adequate resolution.
- The choice of 1 microsecond can also be related to telemetry measurement standards for use on an aircraft test range.

A time interval is modelled as a schema:

$$\begin{array}{|l}
\multicolumn{1}{l}{\textit{TimeInterval}} \\
\hline
\textit{interval} : \mathbb{P}\,\mathbb{T} \\
\textit{start}, \textit{end} : \mathbb{T} \\
\hline
\textit{interval} = \textit{start} \mathrel{..} \textit{end} \\
\hline
\end{array}$$

A resource is associated with a *TimeInterval* by the following schema:

$$\begin{array}{|l}
\multicolumn{1}{l}{\textit{TimeWithResource}[U]} \\
\hline
\textit{TimeInterval} \\
u : U \\
\hline
\end{array}$$

As we only need to know the resources identity in this level of specification, it is introduced as a generic type parameter U to the specification. The type parameter U will be instantiated to define a particular schedule with a specific type of resources.

A periodic process is modelled as an object of the following class:

$$\begin{array}{|l}
\multicolumn{1}{l}{\textit{Proc}} \\
\hline
\begin{array}{|ll}
\hline
t : \mathbb{T} & \text{[period]} \\
c : \mathbb{T} & \text{[computation time]} \\
d : \mathbb{T} & \text{[deadline]} \\
\hline
0 < c \leq d \leq t & \\
\hline
\end{array} \\
\hline
\end{array}$$

where the attribute t is the period of the process; c is an upper bound on the process computation time; d is the process deadline. The relationships between these attributes are captured in the class invariant. The process identity is implicitly captured by the reference semantics of Object-Z.

Given a set of processes, the length of a schedule for all these processes is the least common mutiple of all the periods of the process. It is defined as:

$$\begin{array}{|l}
\textit{length} : \mathbb{P}\downarrow\textit{Proc} \rightarrow \mathbb{T} \\
\hline
\forall\, \textit{procs} : \mathbb{P}\downarrow\textit{Proc} \bullet \\
\quad \textit{length}(\textit{procs}) = \\
\qquad \textit{min}\{m : \mathbb{T} \mid \forall\, i : \textit{procs} \bullet m \bmod i.t = 0\}
\end{array}$$

Note that $\downarrow\textit{Proc}$ denotes a set of references to objects of class *Proc* or any (inheritance) derivative of *Proc*.

Processes synchronisation relationships include precedence and exclusion. It can be specified as:

$$\begin{array}{|l}
\multicolumn{1}{l}{\textit{ProcSynchronisation}} \\
\hline
\begin{array}{|l}
\hline
_ \prec _ : \downarrow\textit{Proc} \leftrightarrow \downarrow\textit{Proc} \\
\hfill \text{[precedence relation]} \\
_ \otimes _ : \downarrow\textit{Proc} \leftrightarrow \downarrow\textit{Proc} \\
\hfill \text{[exclusion relation]} \\
\hline
\forall (p_1, p_2) : (_ \prec _ \cup _ \otimes _) \bullet p_1.t = p_2.t \\
\hline
\end{array} \\
\hline
\end{array}$$

Schedules are modelled as a generic class that consists of a set of processes with precedence or exclusion constraints among some of the processes and a set of resources which execute the processes.

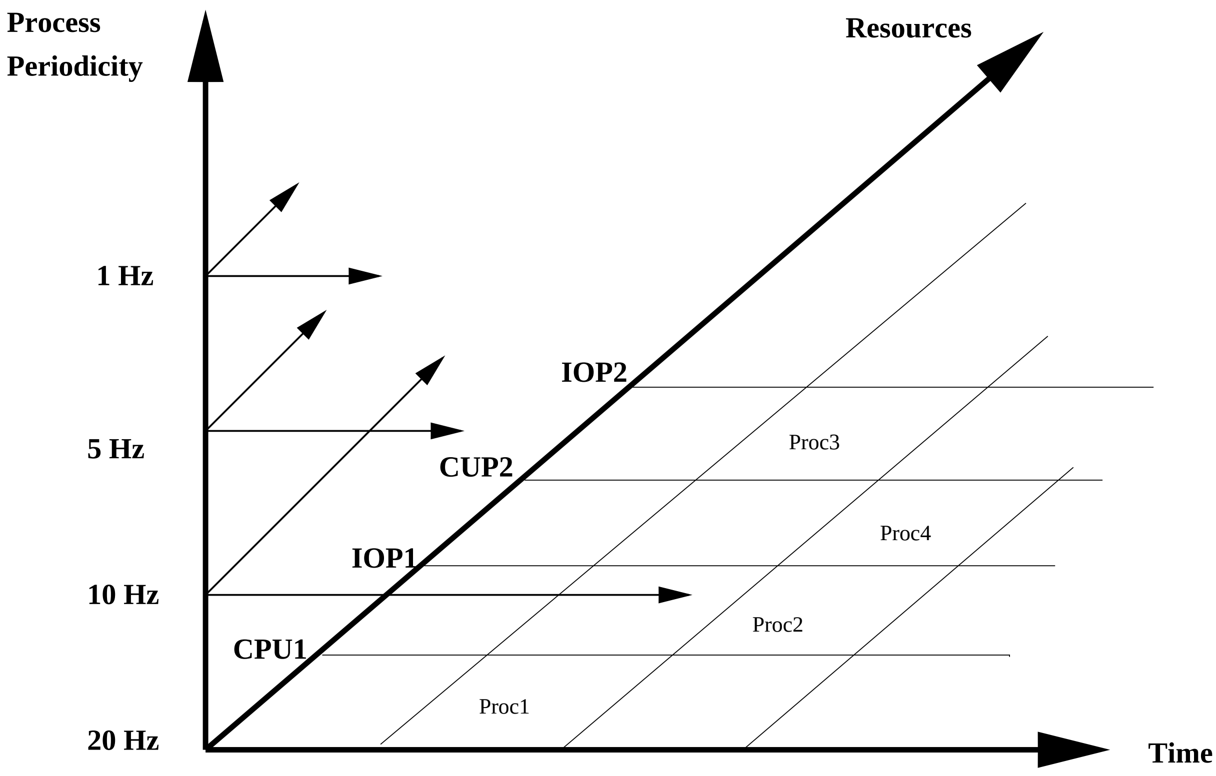

Figure 1: A schedule solution

$$
\begin{array}{|l}
\hline
\textit{Schedule}[U] \\
\textit{ProcSynchronisation} \\
\quad st : \mathbb{T} \qquad \text{[start time of a schedule]} \\
\hline
\quad ps : \mathbb{P}\downarrow \textit{Proc} \qquad \text{[periodic processes]} \\
\quad us : \mathbb{P}\, U \qquad \text{[resources]} \\
\quad \Delta \\
\quad \mathcal{S} : \downarrow \textit{Proc} \pfun \operatorname{seq} \textit{TimeWithResource}[U] \\
\hline
\quad \operatorname{dom}(\_ \prec \_ \cup \_ \otimes \_) \cup \operatorname{ran}(\_ \prec \_ \cup \_ \otimes \_) \subseteq ps \\
\quad \operatorname{dom} \mathcal{S} = ps \\
\quad \{ twu : \operatorname{ran}(\operatorname{ran} \mathcal{S}) \bullet twu.u \} = us \\
\quad \forall p : ps \bullet \# \mathcal{S}(p) = \textit{length}(ps) \operatorname{div} p.t \\
\qquad \forall i : \operatorname{dom} \mathcal{S}(p) \bullet \\
\qquad\quad \mathcal{S}(p)(i).\textit{interval} \subseteq \\
\qquad\qquad ((i-1) * p.t + st) \mathrel{..} (i * p.t + st) \\
\quad \forall p_1, p_2 : ps \bullet \qquad \text{[non-collision]} \\
\qquad \forall twu_1 : \operatorname{ran} \mathcal{S}(p_1);\ twu_2 : \operatorname{ran} \mathcal{S}(p_2) \bullet \\
\qquad\quad twu_1.\textit{interval} \cap twu_2.\textit{interval} \neq \varnothing \\
\qquad\qquad \Rightarrow twu_1.u \neq twu_2.u \\
\quad \forall p : ps \bullet \qquad \text{[non-preemption]} \\
\qquad \forall twu : \operatorname{ran} \mathcal{S}(p) \bullet \\
\qquad\quad twu.\textit{end} - twu.\textit{start} + 1 = p.c \\
\quad \forall (p_1, p_2) : \_ \prec \_ \bullet \\
\qquad \forall i : \operatorname{dom} \mathcal{S}(p_1) \bullet \\
\qquad\quad \mathcal{S}(p_1)(i).\textit{end} \leq \mathcal{S}(p_2)(i).\textit{start} \\
\quad \forall (p_1, p_2) : \_ \otimes \_ \bullet \\
\qquad \forall i : \operatorname{dom} \mathcal{S}(p_1) \bullet \\
\qquad\quad \mathcal{S}(p_1)(i).\textit{end} \leq \mathcal{S}(p_2)(i).\textit{start} \vee \\
\qquad\quad \mathcal{S}(p_2)(i).\textit{end} \leq \mathcal{S}(p_1)(i).\textit{start} \\
\hline
\end{array}
$$

The declaration $ps : \mathbb{P}\downarrow Proc$ introduces ps as a set of references to objects of class $Proc$ or any (inheritance) derivative of $Proc$. The attributes $\_ \prec \_$ and $\_ \otimes \_$ (inherited from the class $ProcSynchronisation$) denote the precedence and exclusion relations between processes respectively. The attribute $us : \mathbb{P}\, U$ specifies a set of resources of a generic type U which will be instantiated to a specific type of resources when defining a particular schedule.

The attribute $\mathcal{S}$ specifies an outcome result of a schedule. It models that, given a process, it returns a list of periods that the process is scheduled in with the corresponding resources that the process is scheduled on. Note that there are a number of ways to specify the scheduling result. For example, the solution ($\mathcal{S}$) can be modelled as

$$\textit{TimeInterval} \pfun U \pfun \downarrow \textit{Proc}$$

However, we believe the repeat pattern of different processes and the synchronisation constraints can be effectively captured in this model:

$$\downarrow \textit{Proc} \pfun \operatorname{seq} \textit{TimeWithResource}$$

The attribute $\mathcal{S}$ is modelled as a secondary attribute[6] because the value of $\mathcal{S}$ depends on the state of primary attributes. Syntactically, the declarations of secondary attributes appear below the Δ separator placed in the declaration section of the state schema. In this class, the Δ separator also distinguishes between 'what the inputs to a schedule are' and 'what the output (result) of a schedule should be'.
In the next section, we will instantiate the above generic scheduling model to specify the OFP's scheduling requirements.

3 Modelling OFP Scheduling Requirements

We study a typical military avionics system which operates under the control of two mission computers, MC1 (for navigation) and MC2 (for weapon delivery). Each computer has a disjoint set of processes and resources. Processes are classified according to their periodicity and all processes have a periodicity of $50k, 100k, 200k, 1000k$ micro-seconds (corresponding to $20, 10, 5, 1$ Hz), except weapon-release process which has a periodicity of $10k$ micro-seconds (100 Hz) on MC2. Further detailed process classifications according to their functionalities are also documented in [12]. The resources of each mission computer consist of a CPU, three serial channels, and a parallel discrete channel. In the OFP, each process in a schedule can be allocated to one specific resource only. Each computer has its own executive program to direct scheduling and intercomputer communication[12]. However synchronisation constraints not only exist between processes within one mission computer, but also exist between the processes on the different mission computers. If there is a precedence relationship between processing performed in MC1 and processing in MC2 then MC1 process will always occur before MC2. The periodic processes for MC2 have a fixed delay with respect to the periodic processes of MC1. This delay is required to allow the inter-MC data transfer following navigation processing in MC1.
Other OFP requirements are limited in this paper because we aim to demonstrate the approach rather than give a complete detailed specification.

Formal Model of the OFP Schedule

The OFP resources are specified for the two different mission computers according to:

$$U_1 \mathrel{\widehat=} CPU1 \mid SER11 \mid SER12 \mid SER13 \mid DISC1$$
$$U_2 \mathrel{\widehat=} CPU2 \mid SER21 \mid SER22 \mid SER23 \mid DISC2$$

OFP processes are specified by the following classes which are defined by inheriting the general periodic process *Proc* (defined in Section 2). The classification is based on different mission computers and periodicities.

$MC1_{20}$
Proc
$t = 50k$

$MC1_{10}$
Proc
$t = 100k$

$MC1_{05}$
Proc
$t = 200k$

$MC1_{01}$
Proc
$t = 1000k$

WeaponRelease
Proc
$t = 10k$

$MC2_{20}$
Proc
$t = 50k$

$MC2_{10}$
Proc
$t = 100k$

$MC2_{05}$
Proc
$t = 200k$

$MC2_{01}$
Proc
$t = 1000k$

Processes are grouped by the following class-union[5, 3] for MC1 and MC2.

$$MC1Procs \mathrel{\widehat=} MC1_{20} \cup MC1_{10} \cup MC1_{05} \cup MC1_{01}$$
$$MC2Procs \mathrel{\widehat=} WeaponRelease \cup MC2_{20} \cup MC2_{10} \cup MC2_{05} \cup MC2_{01}$$

By using inheritance, the above processes can be further classified into more specific processes according to their functionalities. For example, the MC1 display programming transfer processes can be defined as:

$MC1XDPTX$

$MC1_{01}$

$des : String$

$c = 1k$
$d = 1000k$
$des = \text{`}MC1\ DISPLAY\ PRG\ TRANSFER\text{'}$

The $MC1XDPTX$ example demonstrates that all detailed process classification in [12] can be defined using this approach. The present aim is to demonstrate this approach rather than provide the complete formal description of the aircraft scheduler which is too lengthy for this paper.

The schedules for MC1 and MC2 are specified by inheriting the instantiated generic schedule model as:

$MC1Schedule$

$Schedule[U_1]$

$ps \in MC1Procs$
$\forall\, p : ps \bullet$
$\quad \forall\, i, j : \operatorname{ran} \mathcal{S}(p) \bullet i.u = j.u$

$MC2Schedule$

$Schedule[U_2]$

$ps \in MC2Procs$
$\forall\, p : ps \bullet$
$\quad \forall\, i, j : \operatorname{ran} \mathcal{S}(p) \bullet i.u = j.u$

The class invariant

$$\forall\, p : ps \bullet \\ \quad \forall\, i, j : \operatorname{ran} \mathcal{S}(p) \bullet i.u = j.u$$

captures that each process in the scheduling model can be allocated to one specific resource only.

An OFP schedule consists of the MC1 schedule and the MC2 schedule with a time delay. Precedence and exclusion constraints also exist between processes of the two MCs.

$OFPSchedule$

$ProcSychronisation[\_ \prec_2 \_/\_ \prec \_, \_ \otimes_2 \_/\_ \otimes \_]$

$Delay : \mathbb{T}$
[a delay between two MC schedules]

$s_1 : MC1Schedule$
$s_2 : MC2Schedule$

$s_1.st + Delay = s_2.st$
$\operatorname{dom} \_ \prec_2 \_ \subseteq s_1.ps \wedge \operatorname{ran} \_ \prec_2 \_ \subseteq s_2.ps$
$\operatorname{dom} \_ \otimes_2 \_ \subseteq s_1.ps \wedge \operatorname{ran} \_ \otimes_2 \_ \subseteq s_2.ps$
$\forall (p_1, p_2) : \_ \prec_2 \_ \bullet$
$\quad \forall\, k : \operatorname{dom} s_1.\mathcal{S}(p_1) \bullet$
$\quad\quad s_1.\mathcal{S}(p_1)(k).end \leq s_2.\mathcal{S}(p_2)(k).start$
$\forall (p_1, p_2) : \_ \otimes_2 \_ \bullet$
$\quad \forall\, k : \operatorname{dom} s_1.\mathcal{S}(p_1) \bullet$
$\quad\quad s_1.\mathcal{S}(p_1)(k).end \leq s_2.\mathcal{S}(p_2)(k).start \vee$
$\quad\quad s_2.\mathcal{S}(p_2)(k).end \leq s_1.\mathcal{S}(p_1)(k).start$

The attributes $\_ \prec_2 \_$ and $\_ \otimes_2 \_$ specify the process precedence and exclusion relation respectively between the two MCs.

This completes the specification of the OFP process scheduling requirements.

4 Conclusion

In this paper, we have first used the Object-Z generic class constructs to build a general task scheduling model. This generic model has then been used as a specification library component which is instantiated to capture the multi-parallel processes scheduling requirements for an OFP. Object modelling techniques, such as inheritance and class-union, are used as mechanisms for the specification of the classification of MC processes. This work not only formalises the scheduling requirements for a particular aircraft, but more importantly, demonstrates an incremental and extendible modelling approach such that the formal model can be readily reused to specify other aircraft OFP scheduling requirements.

This work sets up the formal base model for verifying the correctness of a scheduling algorithm. One of the next tasks would be to verify an existing algorithm against the formal specification. Alternatively, by using real-time refinement techniques[11, 18], it is possible to derive a scheduling algorithm from the model presented in this paper. So far, we have modelled static scheduling requirements. We believe that it is possible to present a formal model to capture the dynamic scheduling aspects by adding appropriate operations in the corresponding class definitions. Therefore, one of the possibilities for further work would be to choose a case study which has dynamic scheduling requirements to examine the extendibility of the model presented in this paper. We also plan to use Timed Communicating Object Z (TCOZ)[17], a recently developed multi-threaded notation, to specify other physical or software components and their concurrent real-time interactions with the scheduler.

Acknowledgements

We would like to thank Colin Fidge and Brendan Mahony for many helpful discussions on the issues raised in this paper. We also wish to thank the anonymous referees for many helpful suggestions.

References

[1] J. Bowen and V. Stavridou. The Industrial Take-Up of Formal Methods in Safety-Critical and Other Areas: A Perspective. In J. C. P. Woodcock and P. G. Larsen, editors, *FME'93: Industrial Strength Formal Methods*, volume 670 of *Lect. Notes in Comput. Sci.* Springer-Verlag, April 1993.

[2] D.W. Campbell. Timing Analysis in Hard-Real-Time Systems with Application to the CF-188 . Master's Thesis, Dept of Electrical and Computer Engineering, Royal Military College of Canada, 1991.

[3] J.S. Dong. Living with Free Type and Class Union. In *The 1995 Asia-Pacific Software Engineering Conference (APSEC'95)*, pages 304–312. IEEE Computer Society Press, December 1995.

[4] J.S. Dong, J. Colton, and L. Zucconi. A Formal Object Approach to Real-Time Specification. In *the 3rd Asia-Pacific Software Engineering Conference (APSEC'96)*, Seoul, Korea, December 1996. IEEE Computer Society Press.

[5] J.S. Dong and R. Duke. Class Union and Polymorphism. In C. Mingins, W. Haebich, J. Potter, and B. Meyer, editors, *Proc. 12th International Conference on Technology of Object-Oriented Languages and Systems. TOOLS 12 & 9*, pages 181–190. Prentice-Hall, November 1993.

[6] J.S. Dong, G. Rose, and R. Duke. The Role of Secondary Attributes in Formal Object Modelling. In Alex Stoyenko, editor, *The First IEEE International Conference on Engineering Complex Computer Systems (ICECCS'95)*, pages 31–38, Ft. Lauderdale, USA, November 1995. IEEE Computer Society Press.

[7] J.S. Dong and L. Zucconi. A Framework for Adding Time into Formal Object Models. In *the 3rd IEEE International Workshop on Object-oriented Real-time Dependable System (WORDS'97)*, pages 26–31, Newport Beach, California, February 1997. IEEE Computer Society Press.

[8] R. Duke, G. Rose, and G. Smith. Object-Z: a Specification Language Advocated for the Description of Standards. *Computer Standards and Interfaces*, 17:511–533, 1995.

[9] J.D.G. Falardeau. Schedulability Analysis in Rate Monotonic Based Systems with Application to CF-188 . Master's Thesis, Dept of Electrical and Computer Engineering, Royal Military College of Canada, 1994.

[10] C. Fidge, P. Kearney, and M. Utting. Quartz: An integrated formal development method for real-time software. Technical Report 94-26, Software Verification Research Centre, The University of Queensland, Australia, September 1994. (To appear in IEEE Software 1997).

[11] C. Fidge, M. Utting, P. Kearney, and I. Hayes. Integrating Real-Time Scheduling Theory and Program Refinement. In *Proceedings of FME'96: Industrial Benefit of Formal Methods*, pages 325–346, Oxford, March 1996. Springer-Verlag.

[12] J.A.M. Gagne. A Pre-Run-Time Scheduling Algorithm with Application to the CF-188 Aircraft. Master's Thesis, Dept of Electrical and Computer Engineering, Royal Military College of Canada, 1989.

[13] S. Gerhart, D. Craigen, and T. Ralston. Experience with Formal Methods in Critical Systems. *IEEE Software*, January 1994.

[14] C. Jones. *Systematic Software Development Using VDM.* International Series in Computer Science. Prentice-Hall, 1986.

[15] N. Leveson, M. Heimdahl, H. Hildreth, and J. Reese. Requirements specification for process-control systems. *IEEE Trans. Software Eng.*, 20(9), September 1994.

[16] S. Liu, V. Stavridou, and B. Dutertre. The Practice of Formal Methods in Safety-Critical Systems. *J. SYSTEM SOFTWARE*, 28:77–87, 1995.

[17] B. Mahony and J.S. Dong. Blending Object-Z and Timed CSP: An introduction to TCOZ. Technical Report 97-22, Mathematical and Information Sciences, Commonwealth Scientific and Industrial Research Organisation (CSIRO), Australia, 1997.

[18] B. P. Mahony. *The Specification and Refinement of Timed Processes.* PhD thesis, The University of Queensland, 1992.

[19] R. Milner. *Communication and Concurrency.* International Series in Computer Science. Prentice-Hall, 1989.

[20] J. Xu and D. L. Parnas. On Satisfying Timing Constriants in Hard-Real-Time Systems. In *Proceedings of ACM SIGSOFT'91 Conference on Software for Critical Systems*, New Orleans, December 1991.

Appendix: Glossary of Notation

Sets, Functions and Relations

$\mathbb{N}$	The set of natural numbers, i.e. $\{0, 1, 2, ...\}$
$\mathbb{P}\, X$	Powerset: the set of all subsets of X
$\# X$	Size (number of members) of a finite set
$\{D \bullet t\}$	The set of values of the term t, e.g. $\{n : \mathbb{N} \bullet 2 * n\}$ is the set of even numbers
$X \leftrightarrow Y$	The set of all relations from X to Y;
$x \underline{R}\, y$	x is related by the relation R to y
$X \nrightarrow Y$	The set of partial functions from X to Y;
$\mathrm{dom}\, R$	The domain of a relation
$\mathrm{ran}\, R$	The range of a relation
$S \lhd R$	The relation R with domain restricted to S
$R \rhd T$	Range restriction to T
$m \mathrel{..} n$	The set of integers between m and n.
$min\, S$	Minimum of a set; for $S : \mathbb{P}_1\, \mathbb{Z}$, $min\, S \in S \wedge (\forall x : S \bullet x \geq min\, S)$.

Object-Z Overview

Object-Z is an extension of the Z formal specification language to accommodate object orientation. The main reason for this extension is to improve the clarity of large specifications through enhanced structuring.

Classes

A class is a template for *objects* of that class: for each such object, its states are instances of the class' state schema and its individual state transitions conform to individual operations of the class. An object is said to be an instance of a class and to evolve according to the definitions of its class.

A generic queue example

Queue[*T*]

$items : \mathrm{seq}\, T$
Δ
$size : \mathbb{N}$
———
$size = \# items$

Join
$\Delta(items)$
$item? : T$
———
$items' = items \frown \langle item? \rangle$

Leave
$\Delta(items)$
$item! : T$
———
$items \neq \langle\, \rangle \wedge items = \langle item! \rangle \frown items'$

The state schema is nameless and contains declarations (the attributes) above the short dividing line and a predicate (class invariant) below the line. In this example, it has one attribute *items* denoting a sequence of elements of the generic type T.
The remaining two schemas are operation schemas. Operation schemas have a Δ-list of those attributes whose values may change. By convention, no Δ-list means no attribute changes value. Every operation schema implicitly includes the state schema in un-primed form (the state before the operation) and primed form (the state after the operation).
In this example, operation *Join* appends a given input *item*? to the existing sequence of items. Operation *Leave* outputs a value *item*! defined as the head of sequence *items* and reduces *items* to the tail of its original value.
Note that the attribute *size* representing the size of *items* is a secondary attribute. Syntactically, the declarations of secondary attributes appear below the Δ separator placed in the declaration section of the state schema. The values of the secondary attributes are always subject to change. This is achieved by implicit inclusion of secondary attributes in the Δ-list of every operation. In this case, the value of *size* is always subject to change whenever operation *Join* or *Leave* is invoked. For a detailed discussion on secondary attributes see [6].
Further details on Object-Z are given in [8].