Reservation-Based Timing Analysis
(A Partitioned Timing Analysis Model for Distributed Real-Time Systems)

Submitted for the degree of Doctor of Philosophy

Alan Grigg

Department of Computer Science,

University of York

August 2002

ABSTRACT

A real-time system is one where timeliness is intrinsic to its correct operation. Verification of a real-time system involves *a priori* specification of its timing requirements and a combination of analysis and testing to show that these have been met. The focus of the thesis is on the use of analytical methods, so-called *timing analysis*.

Timing analysis models for modern applications involving distributed systems are prone to being *holistic* - the emergent timing behaviour of the system can only be assessed with knowledge of the timing attributes of the entire system. This problem occurs due to the combined impact of functional integration (ordering constraints and data communication/sharing) and physical integration (sharing of physical computing resources) between concurrent software components, giving rise to cyclic inter-dependencies within the timing model. Whilst holistic timing models can be solved using iterative techniques, there are problems concerning their practical application. Fundamentally, it is difficult or impossible to analyse the timing behaviour of parts of the system in isolation from other parts. Correspondingly, localised changes to the system, *eg.* to timing requirements, software implementation details or target hardware details, can have a widespread impact on its emergent timing behaviour as predicted by the analytical model. In practical terms, significant costs can be incurred due to the scope of re-verification effort required after making such changes late in the development process or later during in-service upgrades.

The problem is compounded by current trends in industrial real-time systems applications, typified by those in the aerospace domain - the target application area for this research. There is a desire to adopt more *modular* and *open* architectures and to achieve more *evolvable* system development and upgrade. To this end, major research and development programmes are ongoing in both civil and military aerospace domains, generally referred to as *integrated modular avionics* (IMA).

This thesis describes the development of a timing analysis approach, referred to as *reservation-based timing analysis*, that addresses these issues. The approach provides :

- the ability to model the timing behaviour of parts of a distributed real-time system independently from other parts and, correspondingly, limit the scope of re-analysis required in the event of localised changes;
- the ability to assess the timing behaviour of the system throughout its development process, based on the latest information available at each stage.

CONTENTS

CHAPTER 1. Introduction 1

1.1 Problem Description 2

1.2 Central Hypothesis 5

1.3 Thesis Structure 5

CHAPTER 2. Timing Analysis of Real-Time Systems - A Survey 7

2.1 Basic Concepts 7

2.2 Structure and Contents of the Review 17

2.3 Cyclic Solutions 18

2.3.1 Classical Results 18

2.3.2 Basic Characterisation 21

2.3.3 Further Advances 22

2.3.4 Summary 25

2.4 Static Priority-Based Solutions 25

2.4.1 Classical Results 26

2.4.2 Basic Characterisation 29

2.4.3 Further Advances 30

2.4.3.1 Synchronous Communication and Precedence Constraints 30

2.4.3.2 Distributed Systems 33

2.4.3.3 Temporal Partitioning 35

2.4.4 Summary 37

2.5 Dynamic Priority-Based Solutions 38

2.5.1 Classical Results 39

2.5.2 Basic Characterisation 40

2.5.3 Further Advances 41

2.5.3.1 Synchronous Communication and Precedence Constraints 41

2.5.3.2 Distributed Systems 42

2.5.3.3 Temporal Partitioning 44

2.5.4 Summary 49
2.6 Industrial Trends and Perspectives 50
2.7 Review Conclusions 53

CHAPTER 3. Rationale and Basic Method of Solution 55

3.1 Requirements-Driven Derivation of the Timing Analysis Model 55
3.2 Partitionable Timing Analysis for a Distributed System 56
3.2.1 Partitioning in the Functional Domain 57
3.2.2 Partitioning in the Physical Domain 59
3.2.3 Partitioning in both Functional and Physical Domains 61
3.3 Timing Analysis through the System Life-Cycle 61
3.3.1 Timing Analysis without System Integration and Implementation Details 63
3.3.2 Timing Analysis for an Evolving System Definition 64
3.3.3 Related Work 65
3.4 Summary 66

CHAPTER 4. Reservation-Based Timing Analysis 69

4.1 Representation and Refinement of System Timing Properties 69
4.1.1 Transaction Model Structure 69
4.1.2 Transaction Timing Properties 73
4.2 End-to-End Timing Analysis 75
4.2.1 Analytical Framework 75
4.2.2 Initialisation and Finalisation of the End-to-End Timing Model 81
4.2.3 Evaluation of the End-to-End Timing Model 82
4.2.3.1 Example 82
4.3 Localised Timing Analysis 90
4.3.1 Rate-Based Execution Model 90
4.3.2 Example 95
4.4 Summary 96

CHAPTER 5. Extensions to the RBA End-to-End Timing Model.....97

5.1 Transaction Level Jitter Control Mechanisms.....97

5.1.1 Impact on Localised Timing Analysis 99

5.1.2 Example 1 100

5.1.3 Example 2 102

5.2 Determining Event Phase Relationships.....103

5.2.1 Phase Relationships Within a Single Transaction.....104

5.2.2 Phase Relationships Across Multiple Transactions 106

5.2.3 Phase - A Generalisation of Jitter 109

5.3 Summary.....109

CHAPTER 6. RBA and System Implementation.....111

6.1 Introduction to GAP Case Study 111

6.2 Basic Cyclic Scheduling Implementation of RBA 113

6.2.1 Example 117

6.3 Cyclic Server Implementation of RBA 123

6.3.1 Example 126

6.4 Implementation of RBA for Civil IMA APEX Applications 130

6.5 Summary.....133

CHAPTER 7. Comparative Assessment 135

7.1 Timing Analysis Models Under Assessment.....135

7.2 Assessment Stage 1 : Time-Triggered Transactions - Responsiveness and Bandwidth Utilisation 137

7.2.1 Test-Set Generation and Example Results.....137

7.2.2 Multiple Test-Set Results.....148

7.3 Assessment Stage 2 : Event-Triggered (Jitter-Controlled) Transactions - Responsiveness and Bandwidth Utilisation 154

7.4 Assessment Stage 3 : Sensitivity to Change - Time-Triggered and Event-Triggered (Jitter Controlled) Transactions.....158

7.4.1 Example Test-Set Results 159

7.4.2 Multiple Test-Set Results 160
7.5 Summary 169

CHAPTER 8. Extensions to the Rate-Based Execution Model and Cyclic Implementation Scheme 171

8.1 Prioritised Activities 171
8.1.1 Introducing High Priority Activities - Dealing with Interference Effects 171
8.1.2 Example 172
8.1.3 Introducing Low Priority Activities - Reclaiming Over-Allocated Bandwidth 175
8.1.4 Example 176
8.2 Synchronous Communication 177
8.2.1 Example 178
8.3 Further RBA Implementation Options 180
8.4 Generic RBA Implementation Model 181
8.5 Summary 182

CHAPTER 9. Research Contribution and Further Work 185

9.1 Problem Description 185
9.2 Central Hypothesis 185
9.3 Research Contribution - Reservation-Based Timing Analysis 186
9.4 Further Work 188

REFERENCES 189

APPENDIX 201

A1. Delay and Jitter Inheritance - Special Cases 201
A2. Derivation of Sufficient-but-Not-Necessary Test *FP-SbNN1* 205
A3. GAP Example - Introducing Prioritised Activities 207
A4. GAP Example - Introducing Synchronous Communication 211

FIGURES

1-1 Real-Time Flight Control Application 1

1-2 Holistic Timing Analysis 2

2-1a Analysis of Worst-Case Behaviour 10

2-1b Analysis of Best-Case Behaviour 10

2-2 Network Abstraction Models 15

2-3 Example Time-Triggered Schedule 18

2-4 Example Static Priority-Based Schedule 27

2-5 Example Static Priority-Based Schedule 27

2-6 Generic Open Architecture Model 50

3-1 Jitter Inheritance 56

3-2 Functionally Partitioned Model 57

3-3 Arrival Jitter Removal through Functional Partitioning 58

3-4 Physically Partitioned Model 59

3-5 Timing Analysis through the System Life-Cycle 64

4-1 Example Transaction 68

4-2a Example Transaction - Nesting Partially Resolved 70

4-2b Example Transaction - Nesting Fully Resolved 70

4-3 Timing Properties of Transaction/Activity $\lambda_{i,..,k}$ 72

4-4 Periodics, Aperiodics and Sporadics 73

4-5 Delay Relationships for Transaction/Activity $\lambda_{i,..,k}$ 74

4-6 Accrued Delay Relationships for Transaction/Activity $\lambda_{i,..,k}$ 75

4-7 Delay Inheritance - One-to-Many Precedence Relationship 75

4-8 Delay Inheritance - One-to-Many Nesting Relationship (Descending) 76

4-9 Delay Inheritance - One-to-Many Nesting Relationship (Ascending) 76

4-10 Delay Inheritance - Many-to-One Precedence Relationship 77

4-11 Delay Inheritance - Many-to-One Nesting Relationship (Descending) 77

4-12 Delay Inheritance - Many-to-One Nesting Relationship (Ascending) 78

4-13 Example Transaction - Timing Requirements 80

4-14 Example Transaction - Level 0 Model 81
4-15 Example Transaction - Partial Assignment of Level 0 Timing Attributes 82
4-16 Example Transaction - Final Assignment of Level 0 Timing Attributes 83
4-17 Example Transaction - Level 1 Model for λ_1 84
4-18 Example Transaction - Partial Assignment of Level 1 Timing Attributes for λ_1 84
4-19 Example Transaction - Final Assignment of Level 1 Timing Attributes for λ_1 85
4-20 Example Transaction - Final Assignment of Level 2 Timing Attributes for λ_1 87
4-21 Consistency of Best- and Worst-Case Timing Attributes 91
4-22 Timing Attribute Assignment as $r_{i,..,k} \rightarrow R_{i,..,k}$ 92
5-1 Jitter Control at $\lambda_{i,..,k}$ 96
5-2a Example Transaction - Jitter Control at λ_4 (Positive) 99
5-2b Example Transaction - Jitter Control at λ_4 (Negative) 99
5-3 Example Transaction - Final Assignment of Level 2 Timing Attributes for $\lambda_{1,1}$ 101
5-4 Example Transaction - Phase Calculation for Events $e^{in}_{1,1,2}$, $e^{in}_{1,1,3}$, $e^{in}_{1,2,3}$, in λ_1 103
5-5 Uncertainty in Multi-Transaction Phase 105
5-6 Example Transaction - Phase Calculation for Events $e^{in}_{1,1,2}$, $e^{in}_{1,1,3}$, $e^{in}_{1,2,3}$, $e^{in}_{7,2}$, $e^{in}_{9,1}$ 106
6-1 Example Task in Cyclic Schedule 113
6-2 GAP Example - Cyclic Schedule Implementation (Execution Rate Parameters) 117
6-3 GAP Example - Cyclic Schedule Implementation (Response Time Parameters) 118
6-4 GAP Example - Cyclic Schedule Implementation (Computation Time Parameters) ... 119
6-5 GAP Example - Cyclic Server Implementation (Execution Rate Parameters) 125
6-6 GAP Example - Cyclic Server Implementation (Response Time Parameters) 126
6-7 GAP Example - Cyclic Server Implementation (Computation Time Parameters) 127
7-1 Example Test-Set - Transaction Topology 138
7-2 Example Test-Set - Resource Topology 139
7-3 Multiple Test-Set Results - Responsiveness 145
7-4 Multiple Test-Set Results - Resource Utilisation for *RBA-Cyclic* Scheme 146
7-5 Multiple Test-Set Results - Responsiveness 147
7-6 Multiple Test-Set Results - Resource Utilisation for *RBA-Cyclic* Scheme 148

7-7 Multiple Test-Set Results - Responsiveness....149
7-8 Multiple Test-Set Results - Resource Utilisation for *RBA-Cyclic* Scheme....150
7-9 Multiple Test-Set Results - Responsiveness....152
7-10 Multiple Test-Set Results - Resource Utilisation for *RBA-Cyclic* Scheme....153
7-11 Multiple Test-Set Results - Responsiveness....154
7-12 Multiple Test-Set Results - Resource Utilisation for *RBA-Cyclic* Scheme....155
7-13 Multiple Test-Set Results - Changes in Responsiveness....158
7-14 Multiple Test-Set Results - Changes in Responsiveness....160
7-15 Multiple Test-Set Results - Changes in Responsiveness....162
7-16 Multiple Test-Set Results - Changes in Responsiveness....164
8-1 Generic Implementation Model....177
A1-1 Delay Inheritance - Many-to-One ('Any') Precedence Relationship....196
A1-2 Delay Inheritance - Many-to-One ('Any') Nesting Relationship (Descending)....197
A1-3 Delay Inheritance - Many-to-One ('Any') Nesting Relationship (Ascending)....197
A1-4 Delay Inheritance - Many-to-One ('All') Precedence Relationship....198
A1-5 Delay Inheritance - Many-to-One ('All') Nesting Relationship (Descending)....198
A1-6 Delay Inheritance - Many-to-One ('All') Nesting Relationship (Ascending)....199
A3-1 GAP Example with Prioritised Activities (Execution Rate Parameters)....202
A3-2 GAP Example with Prioritised Activities (Response Time Parameters) 203
A3-3 GAP Example with Prioritised Activities (Computation Time Parameters)....204
A4-1 GAP Example with Prioritised Activities (Execution Rate Parameters)....206
A4-2 GAP Example with Prioritised Activities (Response Time Parameters) 207
A4-3 GAP Example with Prioritised Activities (Computation Time Parameters)....208

TABLES

2-1 Notation 17

2-2 Example Task Set 18

2-3 Example Task Set and Priority Assignment 26

2-4 Example Task Set Response Times 27

2-5 Example Task Set Response Times 27

4-1 Example Transaction - Localised Timing Attributes for λ_1 86

4-2 Example Transaction - Worst-Case Execution Rates and Resource Requirements 93

6-1 GAP Example - Timing Requirements (CPU Resource) 110

6-2 GAP Example - Cyclic Schedule Implementation (Execution Rate Parameters) 117

6-3 GAP Example - Cyclic Schedule Implementation (Response Time Parameters) 118

6-4 GAP Example - Cyclic Schedule Implementation (Computation Time Parameters) ...119

6-5 GAP Example - Cyclic Server Implementation (Execution Rate Parameters) 125

6-6 GAP Example - Cyclic Server Implementation (Response Time Parameters) 126

6-7 GAP Example - Cyclic Server Implementation (Computation Time Parameters) 127

7-1 Example Test-Set - Timing Requirements 138

7-2 Example Test-Set - Allocation and Localised Response Time Analysis 141

7-3 Example Test-Set - Resource Utilisation (Rquired *v* Allocated) 142

7-4 Example Test-Set - *RBA-Cyclic* Results for Resource 6 142

7-5 Example Test-Set - End-To-End Response Time Analysis 144

7-6 Example Test-Set - Localised Analysis Results for Resource 6 (After Change) 156

7-7 Multiple Test-Set Results - Sensitivity (All Changes) 157

7-8 Multiple Test-Set Results - Sensitivity (Responsiveness Increases Only) 159

7-9 Multiple Test-Set Results - Sensitivity (All Changes) 160

7-10 Multiple Test-Set Results - Sensitivity (Responsiveness Increases Only) 160

7-11 Multiple Test-Set Results - Sensitivity (All Changes) 161

7-12 Multiple Test-Set Results - Sensitivity (Responsiveness Increases Only) 163

7-13 Multiple Test-Set Results - Sensitivity (All Changes) 163

7-14 Multiple Test-Set Results - Sensitivity (Responsiveness Increases Only) 164

8-1 GAP Example - RBA Minimum Execution Rates (With and Without Interference) ...170
8-2 GAP Example - Low Priority Activity Timing Requirements.....172
8-3 GAP Example - Low Priority Activity Response Times.....172
8-4 GAP Example - Synchronous Communication Requirements.....174
8-5 GAP Example - RBA Minimum Execution Rates (With and Without Blocking).....175
A3-1 GAP Example with Prioritised Activities (Execution Rate Parameters).....202
A3-2 GAP Example with Prioritised Activities (Response Time Parameters) 203
A3-3 GAP Example with Prioritised Activities (Computation Time Parameters).....204
A4-1 GAP Example with Prioritised Activities (Execution Rate Parameters).....206
A4-2 GAP Example with Prioritised Activities (Response Time Parameters) 207
A4-3 GAP Example with Prioritised Activities (Computation Time Parameters).....208

NOMENCLATURE

Symbol	*Meaning*
λ	activity
τ	task
e	event
a	minimum inter-arrival time
C	worst-case execution time
c	best-case execution time
D	deadline
d	minimum I/O separation
d^{in}	minimum I/O separation (inherited from predecessors)
d^{out}	minimum I/O separation (exported to successors)
J	arrival/input jitter
J^{in}	arrival/input jitter (inherited from predecessors)
J^{out}	response/output jitter (exported to successors)
Q	conditional release function
R	worst-case response time
r	best-case response time
T	execution period
U	worst-case utilisation
V	maximum execution rate
v	minimum execution rate
w	execution ‘weighting’
δ	execution time (within a scheduling cycle)
Δ	maximum duration of scheduling cycle
ω	maximum separation between specified input events
Ω	maximum separation between specified events

Note that subscripts may be appended to these symbols to provide additional context in the main text.

DECLARATION

Certain parts of this thesis have appeared previously in the following publications (where * indicates principal author) :

Grigg, A.*, Audsley, NC., “Timing Analysis of the ARINC 629 Databus for Real-Time Applications,” *Proceedings of ERA Avionics Conference and Exhibition*, 1996; republished in *Microprocessors and Microsystems Journal* 21(1), 1997.

Grigg, A.*, Audsley, NC., “Towards a Scheduling and Timing Analysis Solution for Integrated Modular Avionics,” *Proceedings of ERA Avionics Conference and Exhibition*, 1997; reprinted in *Aircraft Engineering and Aerospace Technology Journal* 70(4), 1998; also reprinted in *Microprocessors and Microsystems Journal* 22(8), 1999.

Audsley, NC., Bate, IJ., Grigg, A. “The Role of Timing Analysis in the Certification of IMA Systems,” *Proceedings of IEE Symposium on Certification of Ground/Air Systems*, 1998.

Audsley, NC., Bate, IJ., Grigg, A., “Life-Cycle Timing Analysis of IMA Systems,” *Proceedings of IEE Colloquium on Systems Engineering of Aerospace Projects*, 1998.

Grigg, A.*, Audsley, NC., Fletcher, MA., Wake, AS. “A Method for Design and Analysis of Next Generation Aircraft Computer Systems,” *Proceedings of 9th INCOSE Conference on Systems Engineering*, 1999.

Grigg, A., Henderson, N., “A Systematic Method for Development of Reactive Real-Time Systems,” *Proceedings of 10th INCOSE Conference on Systems Engineering*, 2000.

Grigg, A.*, Audsley, NC., “Reservation-Based Timing Analysis - A Practical Engineering Approach for Distributed Real-Time Systems,” *Proceedings of 8th IEEE International Conference on Engineering Computer-Based Systems*, 2001.

ACKNOWLEDGEMENTS

I would like to thank my DPhil supervisor Dr. Neil Audsley for his encouragement and assistance and my internal examiner Professor Andy Wellings for his constructive criticism over the course of my research. I would also like to thank Professor John McDermid, who provided the funding for my post out of the BAE SYSTEMS funded Dependable Computing Systems Centre (DCSC) contract.

Thanks also to the customers in BAE SYSTEMS who supported my research and to Mike Burke (DCSC manager) for helping to promote my work within the company. And thanks to Dr. Iain Bate for his comments on a draft version of this thesis.

CHAPTER 1. Introduction

There are many real-world computing applications that are referred to as *real-time*. Examples of such applications include :

- vehicle control systems, *eg.* aircraft flight control;
- traffic management systems *eg.* air traffic control;
- plant control systems eg. nuclear power plant control;
- telecommunication systems, *eg.* video-conferencing;
- automated manufacturing systems, *eg.* car production.

These application areas are diverse but, for all of these systems, timeliness is intrinsic to their correct operation. The ability to provide results or to perform particular actions within specified intervals of time can be just as important as performing the correct function.

Consider the flight control system of an unstable, fly-by-wire military aircraft [IMechE96], as illustrated in figure 1-1. The control system must continually react to inputs from sensors in the body of the aircraft and signals from the pilot's controls and, in turn, drive a set of actuators that control the orientation and speed of the aircraft. There are strict constraints on the time taken by the control system to perform the necessary calculations in order to keep the aircraft in controlled flight. Failure to meet these timing requirements will lead to an unacceptably high risk of losing the aircraft, since the aircraft is sufficiently unstable that the pilot alone cannot maintain controlled flight.

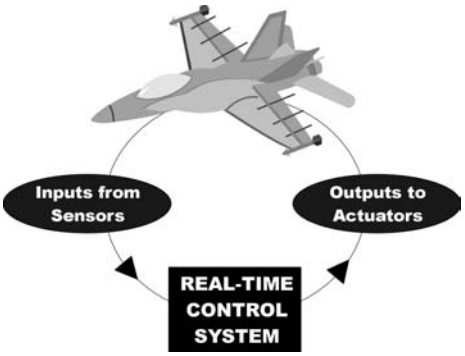


Figure 1-1 : Real-Time Flight Control Application

Just as the functional properties of a system can undergo varying degrees of verification prior to its in-service life, so can the timing behaviour of a real-time system. The degree of rigour with which verification is performed depends on the nature of the application involved. In particular, safety-critical applications (where incorrect operation of the system could jeopardise the safety of some user/operator or third-party, as in the above example) must undergo extensive verification before being allowed to enter service. These systems

are typically subjected to rigorous development standards such as DO-178B [DO-178B] in the civil aerospace domain and Defence Standard 00-55 [DS 00-55a, DS 00-55b] in the military aerospace domain. Defence Standard 00-55, for example, stipulates that '*The safety arguments shall include … analytical arguments. These shall include … analysis of performance, timing and resource usage.'* [DS 00-55a] and that '*The design of the scheduler should ensure that execution is predictable and that analysis can be applied to show that critical timing requirements will be met. Distributed systems inevitably introduce an element of concurrency and constraints on the behaviour of each node should be set to ensure that the behaviour of the overall system is predictable.'* [DS 00-55b].

In order to verify the timing behaviour of a real-time system, it is necessary to show that such behaviour always remains within acceptable limits for the system. These limits are typically described in terms of worst-case response times to particular events that can occur in the operating environment of the system. Verification can be performed by a combination of analysis and testing. The focus of this thesis is on the use of analytical techniques.

1.1 Problem Description

Verification by timing analysis involves the formulation of mathematical models to predict the extremes of temporal behaviour of the real-time system. These analytical predictions can then be assessed against the specified timing requirements for the system in order to determine whether or not these requirements have been met. Timing analysis models for modern applications involving distributed systems, however, are prone to being *holistic* [Tindell94] - the emergent timing behaviour of the system can only be assessed with knowledge of the timing attributes of the entire system. The holistic analysis problem occurs due to the combined impact of functional and physical integration, giving rise to cyclic inter-dependencies within the timing model. The problem will be investigated in detail in chapter 3 but is introduced below via a relatively simple example.

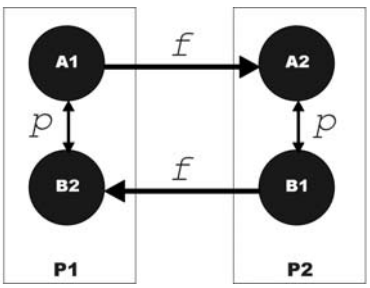


Figure 1-2 : Holistic Timing Analysis

Consider the scenario depicted in figure 1-2 of a two processor system, P1 and P2, with each processor allocated two concurrent tasks, (A1, B2) and (A2, B1), respectively. In this context, each task represents a logical thread of control implemented in software. Two sets of functional dependencies, *ie.* task ordering/precedence relationships, f, are depicted; A1→A2 (read as 'A1 must complete execution before A2 begins execution') and B1→B2.

Two sets of physical inter-dependencies, *ie.* resource sharing relationships, p, are also depicted in the example; A1↔B2 (on processing resource P1) and A2↔B1 (on processing resource P2).

The net result, without imposing constraints on the means by which functional and physical dependencies are managed, is that the analyses of all four tasks are intrinsically inter-dependent. It is not possible to analyse the timing behaviour of any one task, or even the set of tasks on a given processor, in isolation from the rest of the system.

It is easy to see how the holistic effect arises readily in the analysis of larger scale distributed systems when the combined impact of functional and physical integration effects (precedence relationships and shared resources) are taken into account across the whole system. In practice, precedence relationships often involve the communication of data between the parties involved. Whilst this adds to the logical chain of events that need to be modeled (and implemented) and also increases the potential for sharing of resources, *ie.* communication media, it actually introduces no new fundamental contributory factors to the holistic analysis problem.

Holistic timing models can be solved, to predict worst-case timing behaviour, using iterative techniques. Solutions are typically non-optimal but can be subjected to minimum acceptance criteria, such as the need to meet specified timing requirements, and in that respect can be fit for purpose. In [Tindell94] for example, a holistic response time model for fixed-priority scheduled tasks and inter-task communications is presented that can be solved by simple iteration. Tindell's response time analysis is integrated with a generic global optimisation algorithm (simulated annealing) to solve the more general problem of determining a feasible allocation of tasks and inter-task communications to underlying processor and communication resources, *ie.* one where specified timing requirements are met.

The main problem with holistic models is that it is difficult or impossible to analyse the timing behaviour of parts of the system in isolation from other parts. Consequently, the impact of making localised changes to the system on its emergent timing behaviour is typically widespread. This limits the applicability of such analysis techniques with respect to large-scale, industrial real-time systems, where change requirements are encountered frequently (and concurrently) throughout the development of the system. There can be significant costs incurred due to the scope of re-verification effort required after making localised changes, *eg.* to timing requirements, software implementation details or target hardware details, late in the development process or later during in-service upgrades.

The holistic analysis problem is increasingly relevant in the development of appropriate timing analysis models for next generation industrial applications, given current trends in both product architectures and development processes. These trends are typified by current civil and military *integrated modular avionics* (IMA) programmes [AEEC91, Multido98], aimed at defining the characteristics of next generation aircraft on-board computing

systems. IMA provides the target industrial application area for this research, although the principles involved are more generally applicable in the wider context of distributed real-time systems.

From the perspective of the IMA product architecture, there is a move towards increasing integration. Functional integration is sought through increased sharing of functions and data between traditionally independent aircraft sub-systems. Physical integration is sought through increased sharing of computing platform resources - processing and communication media. Integration is the fundamental cause of the holistic timing analysis problem. At the same time, the overall scale of the problem will increase due to anticipated quantitative growth in application software and the underlying computing platform.

From the perspective of the IMA system development process, the scale and complexity of engineering effort will increase in line with the product architecture trends outlined above. The process is further complicated by a move towards more collaborative, multi-national development teams and the desire to achieve more *evolvable* system development and upgrade, *ie.* the ability to develop and upgrade the product in a more incremental manner. This means coping with changes that are smaller in scale but more frequent in occurrence than has previously been the case, where change has invariably been less frequent but more radical, typified by mid-life update programmes that often involve as much effort and cost as the original development programme.

For the timing analysis model to support this evolvable development strategy, it should support the notion of evolvable timing analysis. That is, the model should be able to provide consistent predictions about the timing behaviour of the system based on the latest information available at successive stages of development. This is also consistent with the fact that timeliness is an intrinsic, non-functional property of a real-time system and cannot be added at the end of the development phase - the penalty for late consideration of system timing properties can be significant in terms of re-work and its associated costs. By evolving the timing analysis mode alongside the development of the system, the chances of re-work arising from detection of timing problems late in the development process may be reduced.

In summary, the timing analysis model for a distributed real-time system should provide :

- the ability to model the timing behaviour of parts of the system independently from other parts and, correspondingly, limit the scope of re-analysis required in the event of localised changes; this is referred to as *temporal partitioning*;
- the ability to assess the timing behaviour of the system throughout the development process, based on the latest information available at each stage.

Note that similar requirements have been stated in recent literature on IMA and industrial avionic systems in general. In particular, it is suggested in [Borky98] that, after the definition of a common core computing architecture, the most important requirement of IMA is the ability to limit the impact on timing behaviour when localised changes are made

to the system, such as software implementation changes or upgrades to hardware components. This requirement is driven by both the need to reflect the modular nature of the architecture and the increasing desire to share processing and communication resources between applications of different safety integrity levels. The implication is that the scheduling and timing analysis solution for IMA needs to provide temporal isolation between components of such applications to the extent that any 'misbehaviour' of lower integrity components cannot adversely affect those of higher integrity. In the context of avionic systems in general, Nilsson *et al* have recognised a need for some means of assessing/comparing the real-time performance of different design options, involving decomposed end-to-end timing requirements, long before system implementation details are available [Nilsson98].

1.2 Central Hypothesis

The development of an appropriate timing analysis model to meet the requirements stated above will be pursued throughout the course of this thesis. The central hypothesis is thus stated as follows :

> *An appropriately partitioned timing analysis model can support the stated development objectives for future real-time systems : evolvable development of real-time applications for a distributed computing platform that is functionally and physically integrated.*

1.3 Thesis Structure

The rest of the thesis is structured as follows :

Chapter 2 presents the results of a literature survey in the area of timing analysis techniques for real-time systems, concentrating on the problems and solutions relating to distributed systems;

Chapter 3 gives the rationale for the timing analysis model proposed in this thesis, referred to as *reservation-based timing analysis* (RBA), taking the problem description contained in Chapter 1 as the starting point.

Chapter 4 describes the RBA model in detail with illustrative examples;

Chapter 5 gives some extensions to the RBA model that are necessary to support particular application timing requirements or system engineering preferences;

Chapter 6 describes an implementation (scheduling) approach that supports the goals of the RBA model;

Chapter 7 presents the results of a comparative assessment exercise involving RBA and a number of other timing analysis models (and associated scheduling policies);

Chapter 8 gives some extensions to the proposed implementation (scheduling) approach for RBA.

Chapter 9 outlines the main results of this research, discusses the extent to which the original objectives have been met and highlights potential areas of future work.

CHAPTER 2. Timing Analysis of Real-Time Systems - A Survey

This chapter provides a review of the literature available on timing analysis techniques for real-time systems with a focus on the main results that are pertinent to this thesis. The review is necessarily broad, however, given that the development of a timing analysis model for distributed real-time systems must consider a wide range of issues and that the nature of the problem being addressed is quite general at this stage, *eg.* there is no particular form of scheduling solution assumed.

By way of introduction, the main review is preceded by a description of the basic terminology and general concepts involved. The structure of the rest of the chapter then reflects that of the literature, where a large proportion of results are categorised according to the form of scheduling solution involved.

2.1 Basic Concepts

Fundamentally, the development of a real-time system should involve the definition of system timing requirements and, at some later stage, verification of the timing properties of the final system against those requirements. In order for verification to occur by analysis, the temporal behaviour of the system must be *predictable*. Predictability refers to the ability to determine bounds on the timing behaviour of the system and is a fundamental characteristic of real-time systems [Stankovic90, Burns91]. Two further issues are immediately apparent :

- the time at which the timing analysis can be performed - *offline* (prior to run-time) or *online* (at run-time);
- the stringency of the timing requirements and associated degree of confidence required in the predictions (guarantees) obtained from the timing analysis model - *hard* versus *soft* timing requirements.

These issues are discussed below along with some other basic principles as follows :

- timing properties of the system;
- basic approach to timing analysis;
- accuracy (and complexity) of the model;
- the scheduling problem;
- recurrent and pre-conditioned execution (of concurrent components);
- communication issues;
- resource allocation;
- uncertainty in system state and time.

Firstly, however, the concept of *temporal partitioning* that was introduced in chapter 1, and which is central to this thesis, is reiterated as a lead into the descriptions of basic timing analysis concepts.

Temporal partitioning :

In chapter 1, it was proposed that there is a need to provide some form of partitioning of the timing analysis model for more effective application to real-time systems development in practice. The following definition can be stated :

> *A partitioned timing analysis model allows the system to be analysed in part as well as in whole.*

Temporal partitioning, therefore, is a focal point for the literature review that follows. A number of approaches will be discussed that have been proposed for providing varying degrees of partitioning within a distributed real-time system. These approaches are often referred to as *bandwidth reservation* methods or *bandwidth preserving servers*.

The temporal partitioning approaches described in the literature review are later generalised in chapter 3 as part of the establishment of a rationale for the approach proposed in this thesis.

Offline versus online analysis :

The time at which the timing analysis can be performed depends on the nature of the system and the environment within which it is expected to operate. Timing analysis can safely be performed offline if (and only if) the assumptions regarding the run-time behaviour of the system upon which it is founded are static, *ie.* fixed for all run-time. Such details typically include :

- the timing requirements of the system;
- the decomposition of these requirements (the mapping of timing requirements onto executable software components);
- the mapping of software components onto hardware resources.

Such details are normally fixed offline for safety-critical real-time applications. Analytical evidence of system timing behaviour generated during system development can therefore be used along-side offline testing results as part of the system certification process.

For more dynamic systems, where sufficient information is not available to perform timing analysis prior to run-time, the analysis must be performed online. This is the case for systems that operate in less than fully predictable environments. Moreover, it may be necessary for the analysis to be repeatedly applied at run-time to reflect changing circumstances. For example, the system may be designed to reconfigure itself and/or change the order of execution of application software components depending on its perception of the state of its operating environment.

Hard versus soft timing requirements :

Timing requirements that must be *guaranteed* to be met by the final implemented system are referred to as ‘hard’ requirements. In such applications, the inability to meet stipulated timing requirements is considered to be a failure of the system - this is normally the case for safety critical applications. The associated timing analysis model must be constructed such that the predictions made regarding the timing behaviour of the system will *always* hold true at run-time (provided that the assumptions hold upon which the analysis is based, and the analysis itself is sound).

Timing requirements that need not be guaranteed to be met with such a degree of confidence are commonly referred to as ‘soft’ requirements. In such cases, the assumptions upon which the timing analysis model is founded and/or the form of analysis itself need not be sufficient to ensure that the predictions made regarding the timing behaviour of the system will hold true under all circumstances. Soft timing requirements are associated with applications where a degree of risk (either quantified or unquantified) in the failure to meet timing requirements is acceptable or, quite simply, timing requirements are not explicitly specified. The latter case refers to systems that are implemented according to a ‘best effort’ strategy, *ie.* to make maximum use of available resources given a total workload that cannot entirely be scheduled or to maximise responsiveness at any given time. Quantification of risk for soft real-time applications, *eg.* via probabilistic timing models, is an ongoing research issue - notable contributions include [Atlas98, Lehoczky97, Nissanke01].

System timing properties :

Ultimately, the goal of timing analysis is to assess timing behaviour at the system level with respect to system level timing requirements. The key focus of timing analysis is *responsiveness* - this is simply the time taken by the system to perform operations that can be traced to the timing requirements of the system. For example, the analysis could be applied to predict the maximum time taken by the system to respond to some detected change in its operating environment or to perform a sequence of calculations in response to some triggering event from an internal system clock.

Worst-case responsiveness must typically be determined for all real-time applications in order to show that the system will ‘operate quickly enough'. For many applications, the *best-case* must also be determined in order to show that the system will 'not operate too quickly' if such behaviour is recognised as undesirable. The prediction of best-case responsiveness is clearly required to accurately determine the worst-case variability in response times, referred to as *response jitter*. For example, with reference to the automatic flight control system example cited in the previous chapter, it is necessary to limit the response jitter associated with the main control loop in order to maintain stability of the aircraft.

The true system level timing requirements of a real-time system are sometimes referred to as *end-to-end* requirements - although this is actually more a reference to the physical

implementation of the system than the requirements themselves. This terminology is used since, for any non-trivial system, system level requirements must be dispatched via sequences of diverse computational, communication and I/O operations executed on a physically distributed set of system hardware resources. Such sequences of operations are often referred to as end-to-end *transactions*.

Basic approach to timing analysis :

The timing analysis problem is invariably decomposed into two sub-problems :

- 'unit' level analysis - to determine the resource requirements of concurrent system elements (typically referred to as *tasks* or, analogously in communication terminology, *messages*);
- 'integration' level analysis - to determine the emergent timing behaviour of integrated sets of concurrent system elements with a view to determining whether system level timing requirements have been met, often referred to as *schedulability* or *feasibility* analysis.

This thesis is concerned with the latter and this is reflected in the content of the literature survey presented in this chapter. Integration level analysis for distributed systems is normally performed using composite models comprised of separate processor scheduling and communication analysis models. Often, these models are decomposed in turn to reflect the parallel operation of multiple processing and communication media within the system, possibly reflecting different *scheduling* schemes (see below). These lower level models are aimed at determining task level or message level response times. In turn, the results from lower level models must be combined in some way by the total system level model in order to determine the end-to-end behaviour of the system as an integrated whole.

Accuracy (and complexity) of the model :

A timing analysis model is said to be *sufficient* if the predictions it provides represent true bounds on the actual behaviour under observation, such as the worst-case responsiveness. The difference between the predictions made by a sufficient model and the actual behaviour under observation is referred to as the *pessimism* of the model (when applied to worst-case behaviours; otherwise, for best-case behaviours, this is referred to as *optimism*). If the predictions made do not bound the actual behaviour under observation but represent requirements that must be fulfilled, then the model is said to be *necessary*. The model is said to be *exact* if the difference between predicted and actual behaviour is nil, *ie.* the model is both sufficient and necessary. Figures 2-1a and 2-1b illustrate this terminology in the context of observing worst-case and best-case behaviours, respectively.

In practice, forms of analysis are often referred to as exact when, strictly speaking, the model is either necessary or sufficient. For example, a model may be necessary rather than exact if it ignores run-time overheads such as context switching or run-time operating

system overheads. Alternatively, a model can be sufficient rather than exact if any one of its underlying assumptions is even slightly pessimistic.

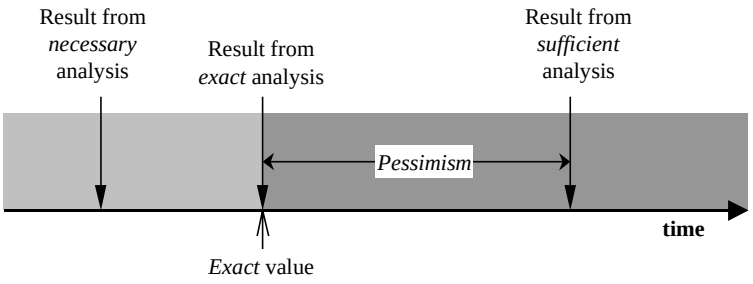


Figure 2-1a : Analysis of Worst-Case Behaviour

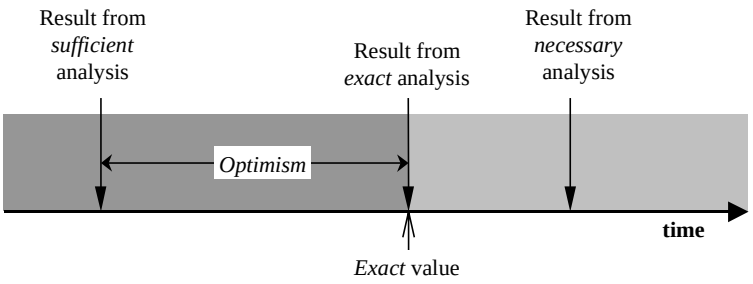


Figure 2-1b : Analysis of Best-Case Behaviour

Exact models are, for all practical applications of real-time systems, computationally intractable to solve - a good discussion on complexity results for scheduling problems is given in [Audsley93b]. For this reason, sufficient models that are relatively simpler to solve are invariably developed and applied in such applications. This highlights a trade-off between model complexity and accuracy.

Scheduling :

Scheduling refers to the means employed for controlling access to a *shared* resource (typically a processor) over time. In communications terminology, scheduling is more specifically referred to as *medium access control* (MAC) but there are strong analogies between the two problems and their associated solutions - see below for more on communications issues.

Just as timing analysis can be classified according to whether it is performed offline or online, so can scheduling. Offline scheduling is referred to as *static*; online scheduling is referred to as *dynamic*. The main classes of scheduling solution described in the literature on real-time systems are as follows :

- *cyclic* schemes [Kopetz97], where processor access is pre-defined in time (or, alternatively, in order with bounded maximum access time).
- *static priority-based* schemes [Burns01], where resource access is ordered according to pre-defined priority values;

- *dynamic priority-based* schemes [Stankovic98], where resource access is ordered according to priority values that are determined (and continually modified) at run-time.

Specific details of these methods will become apparent in the main review, which is structured around this classification of scheduling techniques. This is a reflection that the form of timing analysis applicable to a system is normally a derivative of the scheduling approach(es) adopted.

Scheduling may be *pre-emptive* or *non-pre-emptive*. Pre-emptive scheduling allows the interruption (suspension) of the currently executing task in order to begin or continue the execution of some other task. The process of suspending one task and preparing another for execution is referred to as a *context switch*. Non-pre-emptive scheduling does not permit the ongoing execution of a task to be interrupted and so task execution, once started, will continue to completion. Non-pre-emptive schemes where the currently executing task can relinquish control of the shared resource earlier than its natural completion time, at some designated point in its execution, are referred to as *co-operative* schemes.

Recurrent and pre-conditioned execution :

The recurrent nature of the computations and communications that real-time systems need to perform is modelled in terms of *periodic*, *sporadic* and *aperiodic* operations. More precisely, these terms refer to the maximum frequency of *arrival* of the concurrent operations performed by the system, *ie.* the frequency with which the associated computation or communication can potentially become ready (arrive) for execution.

The different forms of behaviour can be distinguished as follows :

- periodic - with fixed arrival rate (referred to as the *period*);
- sporadic - with bounded minimum time between successive arrivals (referred to as the *minimum inter-arrival time*);
- aperiodic - with unbounded arrival rate.

These properties are typically used during feasibility analysis, in conjunction with ‘unit’ level analysis results, to bound the maximum resource utilisation requirements over time.

Any constraints on the arrival times of individual tasks or messages are normally a derivative of higher level constraints governing the execution of sequences of related operations as is the case within end-to-end transactions. Execution ordering requirements across multiple tasks or messages are referred to as *precedence constraints*. This gives rise to the notion of *time-triggered* and *event-triggered* transactions.

In time-triggered transactions, tasks and messages are constrained to arrive (and subsequently released for execution) at pre-defined times. These times are defined to allow for the worst-case behaviour of all designated ‘predecessors’ as inferred by the precedence constraints within the transaction. In event-triggered transactions, the arrival of successive

operations is triggered directly by the completion of designated predecessors. The time-triggered transaction model is, in fact, a specific instance of the event-triggered model, where the arrival events are constrained to occur at fixed times. In the general event-triggered case, the times at which arrival events occur will be subject to a degree of variability (jitter). The precise form of the transaction model will normally have significant impact on the form and properties of the timing analysis model for a distributed real-time system.

Communication issues :

Timing analysis of any real-time system that involves data communication at some level must address the prediction of the delays involved. Even in uniprocessor systems, delays are normally associated with communication between tasks. Delays associated with synchronous communication via shared memory are typically referred to as *blocking* delays - one task can be blocked from accessing shared data due to the actions of other task(s). In distributed systems, communication delays are more obvious in that there is typically a need to physically transport data from one processing node to another. This transfer can occur via either direct point-to-point connections or otherwise a multi-access databus[†]. In all cases, the predictability of communication delays is dependent upon an appropriate *protocol* being observed, *ie.* a set of rules that constrain the behaviour of the applications concerned and the form and underlying implementation of the data transfer.

The nature of the timing analysis model used to determine communication delays is influenced by the form of protocol and the physical architecture of the communication medium, *ie.* databus or network. A databus is characterised by a single physical communication channel that is shared between multiple terminals (data transmission end-points). Only a single terminal may transmit at any given time and, at that time, all other terminals can potentially receive the transmitted data - the effect of two (or more) users transmitting simultaneously is termed a *collision* which typically results in the corruption of both (or all) transmissions. From a scheduling perspective, this gives rise to the *multi-access problem* [Kurose84], *ie.* how to schedule access to the single physical channel between multiple terminals. MAC protocols must inherently address this problem and specific solutions will be discussed throughout the main review sections. In general, MAC protocols can be categorised as follows [Kurose84] :

- *cyclic* schemes, where access to the underlying medium is pre-defined to occur at fixed times, termed *time-division multiple access* (TDMA), or merely in a fixed order (with bounded maximum access time); an example scheme in this class is the *time-triggered protocol* (TTP) [Kopetz94];

- *demand-adaptive* schemes, which allow run-time contention (multiple terminals can be ready to transmit at any one time) but try to resolve this according to rules which aim to prevent collisions; an example scheme in this class is the *controller area network* (CAN) protocol [Burns95a];
- *contention-based* schemes, which allow both run-time contention and collisions but provide mechanisms for collision detection and recovery; an example scheme in this class is the *ethernet* protocol [Metcalfe76].

Predictability of demand-adaptive and contention-based protocols requires the ability to bound the maximum time required to resolve contention over the physical medium. This is relatively straight forward in the case of demand-adaptive protocols where this time is fixed and related to the number of terminals involved rather than the timing attributes of the workload. For contention-based schemes, however, the time required to resolve contention is dependent upon the instantaneous workload.

The above classification refers to a single (multi-access) communication medium. In practice, distributed systems are often implemented around a *multi-hop network* containing multiple communication media (links, switches and buffers). Some strategy is required to manage the associated suite of network resources in order to deliver predictable end-to-end message delays. The *asynchronous transfer mode* (ATM) protocol [Boisseau95] is a good example of a network protocol that can be configured to deliver predictable delays, as described in [Raha96] for example. Raha's approach is typical in that end-to-end message deadlines are decomposed linearly across multiple network nodes according only to the number of hops from source, *ie.* independent of the actual distribution of load within the network. The intermediate worst-case delays are then calculated for each message at each network resource to determine feasibility.

ATM is also an example of a multi-level protocol, as are often associated with communication networks. The most well-known is the ISO *open systems interconnection* (OSI) model [Day83] aimed at defining a generic standard for open systems. ISO-OSI defines 7 levels of abstraction - the highest level being the application protocol and the lowest being the physical transmission protocol. Rodd *et al* have proposed a sub-set of the ISO-OSI model (reduced to 4 layers) for applicability to real-time applications in the manufacturing domain, known as RTMMS [Rodd90]. Arvind *et al* have also proposed the same abstraction model for a local area network based on a single multi-access databus or ring [Arvind91]. Figure 2-2 illustrates the reduced 4 layer model alongside the full ISO-OSI model.

---

† Multiprocessor systems can also be built around centralised shared memory. From a timing analysis perspective, this represents an intermediate scenario between the uniprocessor and physically distributed multiprocessor cases.

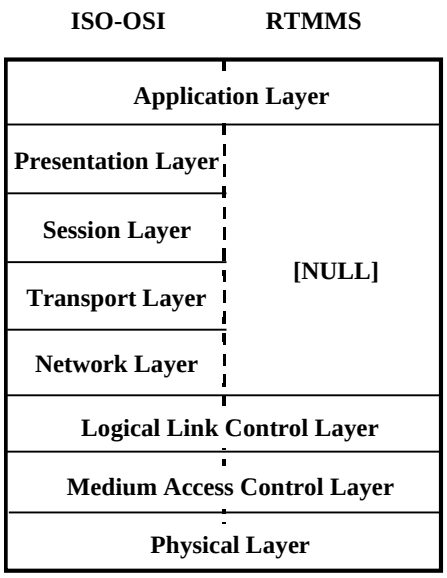


Figure 2-2 : Network Abstraction Models

The significance of the MAC layer has already been noted - this is the communications analogy to scheduling. The operation of the *logical link control* (LLC) layer is also important from a timing analysis perspective. The functions of the LLC layer include the following :

- to provide adaptation between the application level message format and that of the underlying transmission protocol, *eg.* to segment the message into fixed length *packets* (and re-assemble these at the destination);
- to control the release times of packets into the MAC layer for scheduling, *eg.* to support a time-triggered transaction model or, more generally, to control the total communications workload over a given time interval in order to permit delay predictability and ultimately prevent transient overload.

Resource Allocation :

Allocation refers to the assignment of workload (tasks and messages) to system resources (processing and communication media). Like scheduling and timing analysis, allocation can be performed offline or online. Offline allocation, also referred to as static allocation, is required to support static scheduling solutions and, likewise, static offline timing analysis. Online allocation, also referred to as dynamic allocation, encourages the use of dynamic online scheduling and online feasibility analysis.

Static allocation is normally the preferred approach for systems where the total workload can be bounded at system integration time, *ie.* where the timing attributes of tasks and messages are fixed for all run-time. Dynamic allocation tends to be applied in systems where the workload is either truly unbounded or just difficult to predict without unacceptable pessimism. Limited dynamic allocation can also be applied in the context of a predominately static system as a means of reacting to faults or transient overload.

Unless otherwise stated, it is assumed throughout the literature review (and in the remainder of the thesis) that resource allocation is performed offline. For reference, automated techniques that have been proposed for performing allocation offline include simulated annealing [Tindell92] and genetic algorithms [Nicholson98].

Uncertainty in system state and time :

A fundamental problem in distributed systems is how to achieve a consistent and up-to-date understanding of system state and time across multiple processing nodes. A number of primary sources of this uncertainty are apparent :

- *clock drift* - the accuracy to which the time at one processing node can be determined from any other processing node is limited by the maximum drift between the local clocks involved;
- *communication delays* - the speed with which state information at one processing node can be relayed to any other processing node is limited by the communication delay between the nodes involved;
- *behavioural inconsistency* - the ability to predict the future behaviour of one processing node from another is limited by inherent variability in task arrival times, execution times, other resource usage and, in turn, the consequential impact on scheduling behaviour, resultant response times and communication delays.

A consequence of this inherent inability to establish a completely consistent view of system state and time is an inability to determine a *total ordering* of all possible events within the system. Total ordering of events means that any event in the system can be designated as happening either strictly before or strictly after any other event, *ie.* there are no two events which cannot be ordered relative to each other. The best than can be achieved in practice is a *partial ordering* of events - a total ordering of a designated subset of all possible system events. The provision of a global time reference and a total ordering of scheduling events is, however, invariably an implementation choices rather than actual system level requirement. The true timing requirements of the system impose only a partial ordering of application level events and do not explicitly infer global time.

One of the first papers to address the problem of global time and event ordering in the context of multiprocessor systems was [Lamport78], presenting a distributed algorithm for clock synchronisation and *total ordering* of designated events. Lamport's approach is still commonly referenced and often used as a basis for developing more advanced techniques, *eg.* that can tolerate failures, and so a brief summary is given below. A good overview of global time and event ordering techniques is given in [Kopetz97].

Lamport suggests that the notion of time itself is derived from the more basic concept of event ordering. The event ordering algorithm associates a logical clock with each concurrent task, which is updated whenever the task instigates an event itself, *eg.* completes its

execution, or receives a message event from some other task. All message events are time-stamped at source and receiver tasks must update their logical clocks to register a time equal to or later than this time-stamp upon receipt. It is assumed that communications are well behaved - the communication system must either guarantee that messages are received in the order that they are transmitted or, otherwise, messages are assigned unique identifiers and all receipts are acknowledged. Where tasks require mutually exclusive access to shared data resources, typically on a first-come-first-served basis, the algorithm requires all tasks to time-stamp and broadcast all resource requests (and subsequent releases) to all other tasks in the system. The approach is clearly communication intensive and is prone to failure upon the misbehaviour of any one task.

Lamport's method provides a logical notion of common time - there is no common physical time reference. Systems that contains multiple physical clocks can also be synchronised explicitly at the physical clock level by providing a bound on the maximum drift between clocks. This may be achieved via a centralised time server (a single physical clock to which all others are synchronised) or some distributed algorithm aimed at achieving a consensus view of time across the system, *eg.* as implemented in Kopetz's *time-triggered architecture* (TTA) [Kopetz97] (as discussed in section 2.3.1).

2.2 Structure and Contents of the Review

The structure of the literature review reflects that of the literature and the real-time systems research community, where a large proportion of results are categorised according to the form of scheduling solution (or communication MAC protocol) involved. Hence, the main sections of the review are :

- cyclic solutions;
- static priority-based solutions;
- dynamic priority-based solutions.

For each of these classes of solution, the literature is considered for uniprocessor systems, communication networks and distributed systems (although this progression is not reflected directly in the structure of each main review section). The emphasis is on the fundamental properties of the solutions involved rather than on specific details of how the solutions may or may not best be implemented. In particular, the ability to support temporal partitioning is discussed for each class of solution - such approaches are often referred to as *bandwidth reservation* methods or, in the specific context of priority-based solutions, *bandwidth preserving servers*. A good overview of server-based approaches for priority-based solutions is given in [Buttazzo97]. Bandwidth reservation or server-based approaches have been suggested for :

- handling sporadic (and aperiodic) tasks in a way that has no adverse impact on guaranteed periodic tasks;

- handling soft tasks in a way that has no adverse impact on guaranteed hard tasks;
- providing temporal isolation between guaranteed hard tasks;
- providing temporal isolation between message transmissions across multi-hop communication networks.

It is the work in the latter context that has shaped many solutions for temporal isolation of end-to-end task/message sequences (transactions) in distributed systems, including solutions for multimedia applications, as will be discussed in the review.

The final section of the review gives an industrial perspective on the timing analysis problem. Current industrial practice in real-time systems development is summarised but the emphasis is on current trends and perceived problems in future industrial real-time systems.

For convenience, table 2-1 introduces some common notation that is used throughout the review. Let τ_i denote any given task with the following timing attributes :

Symbol	*Meaning*
C_i	worst-case execution time
T_i	period (or minimum inter-arrival time)
D_i	deadline
R_i	worst-case response time
U_i	utilisation
J_i	(release) jitter

Table 2-1 : Notation

2.3 Cyclic Solutions

Cyclic scheduling solutions and associated timing analysis models, often referred to as *time-triggered* solutions, are frequently criticised for being 'inflexible' [Locke92, Stankovic93, Audsley93]. This section of the review will hopefully give an appreciation that, whilst this criticism is valid for more basic cyclic methods, this is not the complete picture. For this reason, this section of the review is sub-divided into two parts, the first describing the traditional view of cyclic scheduling methods and the second covering more recent advances.

2.3.1 Classical Results

The most basic form of cyclic solution is fully deterministic, consisting of a completely time-triggered schedule, *ie.* one where tasks are scheduled to execute at fixed times. The schedule is defined offline (at system integration time) and remains fixed at run-time. The schedule is cyclic in nature, typically with a cycle time equal to the least common multiple (LCM) of all task periods - termed the *major* cycle of the schedule. Each major cycle may be divided up into a number of *minor* cycles, where individual tasks are scheduled to

execute, normally to completion, *ie.* non-pre-emptively. Sporadic tasks are typically polled periodically according to their minimum inter-arrival rate. Verification of time-triggered solutions is by 'schedule construction' rather than analysis, *ie.* feasibility is an integral part of the derivation of an acceptable static schedule.

To illustrate the time-triggered method by example, consider the set of independent tasks and associated timing attributes $\{\tau_i = (C_i, T_i); i = 1,..,4\}$ as given in table 2-2.

i	C_i	T_i
1	5	20
2	7	50
3	3	25
4	4	33⅓

Table 2-2 : Example Task Set

The LCM of the task periods is 100 and so this gives the duration of the major scheduling cycle. The minor cycle duration is set to 20, the shortest period of any one task. Assuming that the arrival and deadline of each task respectively coincide with the start and end of the task period, a feasible schedule is given in figure 2-3.

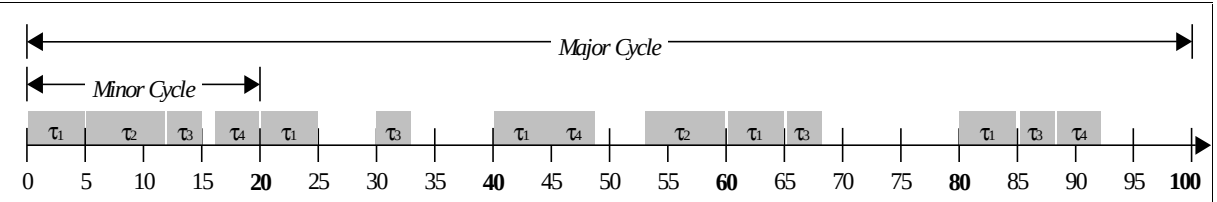


Figure 2-3 : Example Time-Triggered Schedule

This particular example illustrates the use of idle time in the schedule, *ie.* time when some task(s) may be able to run but are kept waiting - such schedules are often referred to as *non-work-conserving*. This is a means of achieving the desired periodic behaviour of the tasks with an acceptable level of jitter - rather than executing tasks at increased rates for the sake of simpler schedule definition.

A principal promoter of time-triggered solutions is Kopetz [Kopetz97], who has stated that event-triggered solutions are unsuitable for hard real-time systems since they result in unmanageable complexity due to the nature of interactions between system components [Kopetz95]. In the same paper, the following benefits of the time-triggered solutions are proposed :

- *partitionable design* - temporal partitioning is inherent from design-time due to the fact that processing nodes are afforded autonomous temporal control; this enables independent design and verification of each sub-system;

- *independence between functional and temporal control structures* - the temporal behaviour of the system is consistent regardless of any run-time variability of applications, *eg.* values of data structures;
- *replica determinism* (for systems which employ active redundancy) - given a global time-base, replica determinism is assured by the fact that replicated processing nodes will operate on the same logical data items at the same time (within the accuracy of the global time mechanism).

Kopetz has overseen the development of the MARS (MAintainable Real-time System) architecture [Kopetz89, Damm89, Kopetz95a], aimed at fault-tolerant real-time control systems with hard timing requirements. This work was instigated in the late 1980's and has since become one of the most notable efforts in the field.

The MARS architecture is partitioned into *clusters*, each containing a number of *nodes* inter-connected by a multi-access databus, governed by the MARS *time-triggered protocol* (TTP). Originally, as described in [Kopetz89, Damm89], each node was a single board computer but, more recently, as described in [Kopetz95a], nodes have become fail-silent sub-systems comprised of two replica boards and software, each decoupled from the main cluster bus via shared memory, termed the *message base interface* (MBI). Inter-cluster communication is via *gateway* nodes, which are logically members of two clusters. Soft real-time applications are isolated in separate clusters and can be implemented by other (non-MARS-compliant) methods, so long as any interfaces to hard real-time applications are sound [Kopetz98].

MARS is characterised by a static resource allocation and time-triggered scheduling solution, underpinned by a global time-base - a 'sparse, global time lattice' [Kopetz92]. Clock synchronisation is achieved by periodic message exchange between nodes. Each node calculates a drift correction value based upon averaging of differences between its own clock reading and those received from other components. For fault tolerance, a number of the highest and lowest readings are discarded before averaging. The maximum drift between clocks is boundable. To prevent non-deterministic behaviour due to clock drift between nodes, all run-time control actions and state information are time-stamped. An explicit agreement protocol is provided to ensure system-wide consistency of control information time-stamps. Moding and fault tolerance are supported by switching between pre-defined static schedules.

The MARS TTP scheme for inter-node communication [Kopetz94] is an example of a TDMA (time-division multiple access) protocol (as categorised in section 2.1). TDMA protocols involve medium access time being partitioned into *frames*, which are in turn partitioned into *time-slots*. Individual time-slots are allocated to individual databus terminals at system integration time in accordance with predicted worst-case transmission requirements (the frequency and maximum length of periodic messages and the minimum

inter-arrival rate and maximum length of sporadic messages). Under TTP, all normal run-time communication is in the form of fixed length periodic messages that are valid only for a defined time interval and subsequently over-written.

A tool known as the MARS *cluster compiler* [Kopetz95a] is employed to generate static schedules offline. The tool supports a number of different heuristics (rules of thumb) that are used to traverse a search tree, potentially containing all valid schedules, in search of an acceptable (sub-optimal) solution. A typical heuristic would seek to minimise the overall length of the cyclic schedule, *ie.* minimise the major cycle time. The method assumes that the resultant schedules for different processing nodes can be synchronised at run-time to the granularity of the smallest time-slot.

2.3.2 Basic Characterisation

The basic form of cyclic solution supported by MARS has been widely criticised for being inflexible [Locke92, Stankovic93, Audsley93]. The final scheduling solution is not amenable to incremental modification and any changes in application requirements or availability of resources must be supported by a new schedule. Such changes can occur readily as part of the normal system development process or be required to achieve some future system upgrade. Further criticisms aimed at pure time-triggered solutions have included :

- *Compromised application timing requirements* :
  The true timing requirements of individual applications may be compromised due to the need to accommodate a range of applications within a common, cyclic schedule. In turn, this contributes to higher resource utilisation than is actually required, without any increase in functional capability for the application.

- *Resource idle time* :
  This may be significant since the actual completion of individual activities may be much earlier than their worst-case would suggest. Sporadic activities pose a particular problem. Where these are *polled*, resource usage may be ineffective due to sporadic arrivals occurring at a much lower rate than their predicted worst-case inter-arrival rate. It is possible, however, to reduce resource idle time by scheduling sporadics to execute in the *slack* time in each minor cycle, *ie.* the time unused by periodics, supported by some appropriate cycle over-run mechanism in case of overload [Baker89, AEEC94].

- *Pessimistic worst-case estimates* :
  Any deviation in actual run-time behaviour beyond predicted worst-case behaviour has the potential to result in information loss or cycle over-run, depending on whether outstanding task executions or fixed cycle durations are deemed more important. This factor encourages pessimism in design-time predictions of worst-case behaviour (in terms of both computation times and arrival rates). According to [Locke92], virtually every system employing cyclic scheduling will experience cycle over-run during its in-

service life and this will normally occur due to unanticipated high load stress - a time at which correct operation can be most critical.

- *Lack of responsiveness* :
  End-to-end transaction response times may be large (compared to the total computation time involved) due to the release of successive activities based on worst-case behaviour of all predecessors. Further, any pessimism in the estimation of worst-case execution times may be directly manifest at run-time as an actual delay in end-to-end response times.

- *Lack of visibility of true precedence relationships* :
  In the final scheduling solution, there is no distinction between precedence relationships reflecting true system requirements and those resulting from the resource sharing policy. Unless true precedence relationships are otherwise documented at design-time, this can result in problematic upgrade of the system beyond the original design.

A notable benefit of time-triggered solutions is highlighted in [Locke92] in that it is simpler to implement synchronous communication requirements (than it is using more flexible scheduling solutions). In a same vein, Baker and Shaw suggest that time-triggered solutions offer significant benefits in distributed systems where synchronisation is required between multiple processing nodes at run-time, *eg.* to support consensus decision-making [Baker89].

Time-triggered solutions have been widely employed in industrial real-time applications. This includes the vast majority of safety critical control systems currently deployed in the aerospace and automotive fields. This is largely due to their deterministic nature, allowing the system timing behaviour to be verified with a high degree of confidence - the final schedule can be inspected manually and extensively tested[†]. For communications TDMA-based protocols have been widely employed, *eg.* [MIL-STD-1553] in the military domain. The lack of flexibility of time-triggered solutions, as currently implemented, means that alternative methods are sought for future industrial applications. Some candidate cyclic solutions are outlined below.

2.3.3 Further Advances

A number of more flexible cyclic scheduling methods have been developed more recently. These methods are more flexible in the sense that the run-time cyclic schedule is adaptable online to react to changing application demands and/or resource availability.

A cyclic scheduling scheme aimed at providing independent execution of hard and soft real-time applications in a multiprocessor workstation environment is described in [Jones97]. Real-time applications are characterised by deadline constraints and resource requirements

[†] The term ‘extensive’ is used rather than ‘exhaustive’ due to the problem of reproducing the true worst-case execution time of code by testing.

expressed in the form of *x* CPU time units in any interval of *y* time units. These requirements can vary at run-time to allow application to adapt their execution behaviour. At run-time, a static schedule is computed per-CPU on the basis of the current application timing and resource requirements and updated each time these change. As far as possible, changes to the schedule are accommodated incrementally rather than by computing a completely new schedule each time. A heuristic search method that adds reservations in order of increasing period is used to construct and amend the schedule. Applications that contain multiple threads must execute these according to a simple round-robin policy. The approach has been implemented in the Rialto operating system [Jones96] executed on top of Windows/NT. The same approach has been implemented in the Rialto/NT operating system [Jones01] executed on top of Windows 2000, where it was found that tasks with CPU reservations could be implemented alongside other non-real-time tasks executed via a priority-based scheduler with priorities assigned dynamically (to maximise performance) using heuristics.

Numerous cyclic scheduling solutions have been proposed recently for distributed systems based upon the *weighted round-robin* (WRR) scheduling approach. This is a pre-emptive scheme whereby tasks that become ready for execution join a first-in-first-out (FIFO) queue and are executed in turn for a defined (maximum) number of CPU 'time-slices' - a standard unit of execution time, typically of the order of tens of milliseconds. Each time a task fails to complete execution within its current time-slice allocation, it is pre-empted and re-joins the back of the queue. Since the duration of a time-slice is normally shorter than the execution time of a task, tasks will normally traverse the FIFO queue several times before completing.

The WRR scheme is *work-conserving*, *ie.* the CPU will be kept busy as long as there is work to be done. A simple form of timing analysis can be stated for the case of a set of independent periodic tasks $\{\tau_i;\ i = 1,..,n\}$ executed under WRR (from [Liu00]). Firstly, a necessary condition to guarantee feasibility of a periodic task τ_i (with deadline equal to period) is that a weight w_i must be assigned such that :

$$w_i \geq \left\lceil \frac{C_i}{\left\lfloor T_i / \Delta \right\rfloor} \right\rceil \qquad (2\text{-}1)$$

where Δ is the maximum cycle time of the WRR schedule (the actual cycle time will vary with load). Note that w_i is a duration rather than a number of time-slots, *ie.* w_i is a multiple of the unit time-slot size. The resulting worst-case response time of task τ_i can be bounded as follows :

$$R_i \leq \left\lceil \frac{C_i}{w_i} \right\rceil \Delta \qquad (2\text{-}2)$$

Clearly, an overall utilisation constraint on the WRR schedule is that the sum of the weight values is no greater than the cycle time :

$$\sum_{i=1}^{n} w_i \leq \Delta \qquad (2\text{-}3)$$

It can be seen from equations *(2-1)* and *(2-2)* that the worst-case task response time depends not only on the weight assigned to the task but also on the maximum schedule cycle time. The maximum cycle time, however, can be fixed independently from the workload and so WRR (and this form of analysis) supports temporal partitioning between tasks sharing a resource.

Since the actual cycle time of the schedule will vary at run-time dependent upon actual workload, task response jitter can be large. For this latter reason, fixed cycle time WRR schemes have been proposed, such as [Golestani91] and [Kalmanek99]. These schemes are actually aimed at scheduling real-time traffic in packet-switched communication networks - an area where WRR has been applied extensively since the low implementation complexity of the scheme supports high throughput and context-switching overheads are minimised due to hardware-implemented multiplexing between packet streams (sometimes referred to as *virtual channels*). In this context, the duration of a time-slice is typically equal to the time required to transmit a single packet of data.

Parekh and Gallager have adopted the idealised WRR model, where the resource bandwidth allocation is infinitesimally fine-grained, as the starting point for their *generalised processor sharing* (GPS) algorithm [Parekh93, Parekh94]. GPS is a work-conserving scheme developed in the context of providing traffic control at communication network interface nodes. As in the case of WRR, minimum execution rates are guaranteed based on 'weight' parameters. In GPS, however, weight values vary at run-time to give a fair share of the resource bandwidth that reflects the actual total demand at any given time (in WRR, the weights remain fixed but the duration of the scheduling cycle will vary). Consequently, implementation schemes proposed for GPS are typically based on *dynamic priorities* rather than cyclic WRR and so are discussed later in section 2.5.3.3. Under a GPS-compliant scheduling scheme, a traffic stream (or task) is guaranteed a fair share of the resource bandwidth in *any* time interval dependent upon the total workload on the resource. With analogy to the WRR analysis stated above, the proportion of bandwidth allocated to a given traffic stream is equivalent to the ratio of its assigned 'weight' to the sum of all weight values for currently active traffic streams. Clearly, no GPS implementation scheme can deliver an idealised implementation but merely an approximation based on some finite granularity of bandwidth allocation.

GPS assumes that the 'bursty' (irregular) arrival patterns associated with network traffic (or, analogously, aperiodic tasks) can be limited by some appropriate means - in the case of network traffic, the *leaky bucket* algorithm [Turner86] is often applied to provide this function, generally referred to as *traffic shaping*. Leaky bucket actually implements WRR without the 'weights', *ie.* basic round robin, at either the byte or packet level to regulate average flow-rate and (in the appropriate configuration) peak flow-rate.

The initial development of GPS was actually based on another WRR-oriented scheme known as *weighted fair queuing* (WFQ), proposed for servicing sporadic/aperiodic traffic at communication network interface nodes [Demers89]. WFQ is sometimes referred to as *packetised GPS*. WFQ applies WRR at the byte level to determine expected completion times of packets awaiting transmission at a network node or switch. The calculation is weighted such that particular virtual communication channels receive a specified proportion of the available bandwidth. Like GPS, WFQ implementation schemes aim to deliver an approximation to idealised WRR and are typically based on *dynamic priorities* and so are discussed later in section 2.5.3.3.

WRR has also been investigated in the context of providing predictable end-to-end delays in an ATM network [Raha95]. Raha *et al* consider the provision of independent minimum delay guarantees per ATM 'virtual channel' (VC) by application of WRR at the ATM packet (cell) scheduling level in network nodes and intermediate switches. VCs are allocated statically to network resources on the basis of maximum buffer sizes predicted by the model. Worst-case delays can then be determined for each VC at each network node or switch and, consequently, the overall end-to-end delays determined by summation. The WRR scheme is found to offer performance improvements over the standard FIFO policy implemented in many ATM hardware components, given a slight increase in implementation complexity.

2.3.4 Summary

Cyclic (static, time-driven) scheduling solutions are often criticised for lacking flexibility [Locke92, Stankovic93, Audsley93]. Whilst this criticism is valid for the most basic forms of cyclic scheduling solution, more flexible solutions have been developed more recently. These solutions have been motivated by the need to balance flexibility with simplicity of implementation in modern applications such as packet-switched communication networks. In particular, a range of cyclic solutions that can support adaptive run-time behaviour, *ie.* changing timing and resource requirements and resource availability, have been proposed based on the WRR scheme, as described in section 2.3.3.

More significantly, from the perspective of this research, WRR-based schemes demonstrate a move towards greater consistency in the form of scheduling and analysis applied to processing and communication resources. Furthermore, the worst-case response time analysis for WRR-based schemes can be applied independently for independent tasks when a fixed maximum cycle time is imposed independent of the workload. The analysis, however, assumes that tasks are periodic time-triggered, *ie.* there is no significant arrival/release jitter involved and also that the deadline of each task is equal to its period.

2.4 Static Priority-Based Solutions

Static priority-based solutions are characterised as follows :

- a run-time dispatching policy of '*execute the highest priority runnable task*' (normally with pre-emption enabled);

- a method for offline assignment of task priorities;
- a timing analysis model for offline assessment of feasibility.

There are many good sources of information on the wide range of static priority-based solutions that have been developed over the last 30 years or more. [Burns01] gives a good overview of timing analysis models and, in particular, provides extensive coverage of programming language and run-time system definition and implementation issues. [Liu00] and [Buttazzo97] both give good description of more flexible static priority-based implementation techniques, *eg.* for dealing with sporadics, as well as considering priority-based communication network protocols. In order to keep this review focused, the emphasis is on key results that underpin static priority-based methods and contribute to their emergent properties and upon more recent extensions for distributed system applications.

2.4.1 Classical Results

A historical perspective on the development of static priority-based solutions is given in [Audsley95]. This paper describes some very early material that pre-dates the extensively referenced Liu and Layland paper [Liu73]. Most notably, in [Fineberg67], Fineberg and Serlin considered the case of two independent periodic tasks on a uniprocessor, with each task deadline equal to its period and computation time no greater than its deadline. They observed that assigning the higher priority to the task with the shorter period was more likely to result in a feasible schedule. They also presented a sufficient (pessimistic) utilisation-based schedulability test for this simple case of two concurrent tasks :

$$U \leq 2(\sqrt{2}-1) \approx 82.8\% \qquad (2\text{-}1)$$

where U is the combined utilisation of the task set, which can be defined more generally as follows for the case of n tasks :

$$U = \sum_{i=1}^{n} \frac{C_i}{T_i} \qquad (2\text{-}2)$$

Two publications appeared several years later, [Serlin72] and the now widely recalled Liu and Layland paper [Liu73], proposing a generalisation of this simple test to cover for the case of n independent periodic tasks $\{\tau_i;\ i = 1,..,n;\ C_i \leq D_i = T_i\}$ where tasks with shorter periods are assigned higher priorities :

$$U \leq n(2^{1/n}-1) \qquad (2\text{-}3)$$

The test converges to $log_e(2)$ for very large n, which translates to a utilisation of approximately 69.3%. It was also observed that all tasks will experience their worst case response time when they are released simultaneously with all other tasks of higher priority. Liu and Layland termed this the *critical instant* and named the method of priority assignment *rate monotonic*. They proceeded to show that rate monotonic priority assignment is optimal in the sense that any set of independent periodic tasks $\{\tau_i;\ i = 1,..,n;\ C_i$

$\leq D_i = T_i\}$ that cannot feasibly be scheduled under rate monotonic priority assignment cannot feasibly be scheduled by any other fixed priority assignment method either.

Inherent in these early studies were a number of assumptions, including the following :

- task release occurs immediately upon arrival;
- no task may voluntarily suspend its own execution prior to completion;
- context switching overheads are ignored.

The next major development came almost a decade later. Relaxation of the rate monotonic constraint $D_i = T_i$ to deal with the more general case of $D_i \leq T_i$ results in the loss of optimality. For such a set of independent periodic tasks $\{\tau_i; i = 1,..,n; C_i \leq D_i \leq T_i\}$, *deadline monotonic* priority assignment [Leung82] is optimal. Under the deadline monotonic approach, tasks with shorter deadlines are assigned higher priorities. Clearly, for the special case where every task deadline is equal (or otherwise proportional to) its period, deadline monotonic and rate monotonic priority assignments are equivalent.

The ability to determine task worst-case response times was another significant advance in the theory of static priority-based solutions. Harter initially proposed *response time analysis* (RTA) involving systematic calculation of a worst-case response times at successively decreasing priority levels [Harter84] (later extended in [Harter87]). Joseph and Pandya developed the idea to provide an exact (sufficient and necessary) expression for calculation of worst case response times for a set of fixed priority tasks $\{\tau_i; i = 1,..,n; C_i \leq D_i \leq T_i\}$ [Joseph86] :

$$R_i = C_i + \sum_{\tau_j \in hp(\tau_i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j \qquad (2\text{-}4)$$

where the function $hp(\tau_i)$ is the set of all tasks in the task set with higher priority than task τ_i. Audsley later described an iterative method for solution of this equation as a recurrence relation [Audsley91]. RTA has continued to provided the basis for development of the majority of static priority-based methods up to the present day.

To illustrate the basic method involved, consider the task set from section 2.3.1. The timing attributes of the task set and the resultant rate monotonic priority assignment $\{P_i; i = 1,..,4\}$ are as follows :

i	C_i	T_i	P_i
1	5	20	1
2	7	50	4
3	3	25	2
4	4	33⅓	3

Table 2-3 : Example Task Set and Priority Assignment

The total utilisation of the example task set is 63%. This value is less than the maximum schedulable utilisation bound for rate monotonic of 69.3% (from equation *(2-3)*) and so the task set can be shown to be feasible for the case where all $D_i = T_i$. Worst-case response times can be calculated by iterative application of equation *(2-4)*; results are as follows :

i	R_i
1	5
2	19
3	8
4	12

Table 2-4 : Example Task Set Response Times

The resultant 'worst-case' schedule, *ie.* for the case when all tasks exhibit worst-case behaviour following a critical instant release, is shown in figure 2-4.

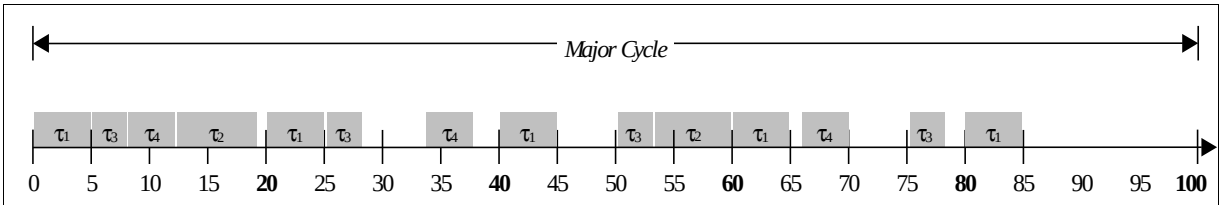


Figure 2-4 : Example Static Priority-Based Schedule

It turns out that, due to the particular timing attributes involved, there is no need for pre-emption to occur. The schedule is different, however, for tasks τ_2, τ_3 and τ_4, from the time-triggered schedule (for the same set of tasks) shown in figure 2-3. To illustrate pre-emption in the schedule, consider the effect of an increase in the computation time of the lowest priority task τ_2 such that $C_2 = 12$. This change does not affect the priority assignment but the new results of response time analysis are as follows :

i	R_i
1	5
2	32
3	8
4	12

Table 2-5 : Example Task Set Response Times

The impact of the change has been to disproportionately increase the response time of task τ_2 as a result of pre-emption due to higher priority tasks. This can be seen in the resultant 'worst-case' schedule shown in figure 2-5 where the times that task τ_2 is resumed after pre-emption indicated by darker shading.

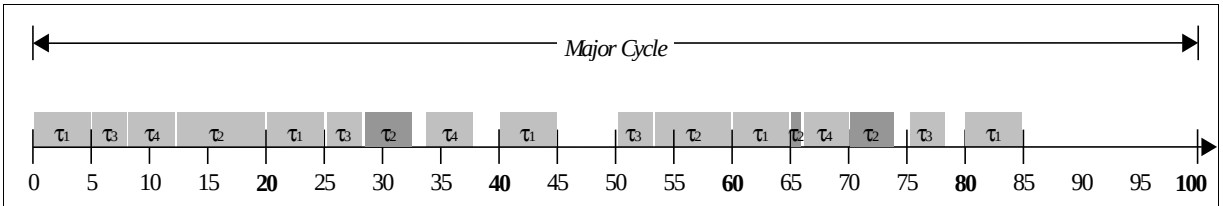


Figure 2-5 : Example Static Priority-Based Schedule

The overall utilisation of the task set after the change is 73%. This would cause the utilisation-based analysis to fail (since 73% is greater than the maximum schedulable utilisation bound of 69%). This serves to illustrate that the utilisation-based test is sufficient rather than exact, *ie.* when the test is passed the task set is always schedulable but when the test fails the task set is not necessarily unschedulable.

Note that the term 'major cycle' can still be applied to the resultant schedule since a critical instant will occur at each multiple of the LCM. In general, however, the actual schedule will be different within each major cycle due to variation in task execution times.

2.4.2 Basic Characterisation

There has been extensive development to static priority-based methods since the fundamental uniprocessor results described above were published. The initial formulation of the basic method and response time analysis, however, is sufficient to determine some important properties of this class of solution.

Static priority-based solutions are work-conserving - whenever there is work to do, *ie.* a task to execute, then the processor (or some other resource) will be kept busy. Resources will only remain idle when there is no work to perform. Hence, for a given set of tasks, average-case response times will typically be shorter under a static priority-based solution than under a time-triggered solution, where resources can occasionally remain idle even when there are tasks ready to execute. Locke's comparison between time-triggered and static priority-based solutions [Locke92] concluded that static priority-based solutions tend to experience more jitter than time-triggered solutions.

A related but undesirable property is apparent in the form of the exact response time analysis given in equation *(2-4)*. The response time calculation for each task τ_i is dependent upon the timing attributes of all higher priority tasks - the set $hp(\tau_i)$. Hence, using this exact form of analysis, it is not possible to determine the response time of a given task in isolation from the rest of the task set (other than for the highest priority task). In turn, changes to the timing attributes of a given task in the task set will generally impact on the response times of all other lower priority tasks (where the change impacts on the priority assignment, all tasks with lower priority in the new priority ordering will be affected).

This observation is backed up by Punnekkat *et al*, who investigated the sensitivity of static priority-based solutions to changes in task worst-case execution times [Punnekkat97]. A method is proposed to determine the extent that computation times can be increased in a

feasible task set without becoming infeasible or reduced in an infeasible task set to become feasible. It is observed that, in general, fixed priority task sets are very sensitive to change, with tasks at the lower priority end of the scale being particularly vulnerable.

These characteristics would appear to represent a constraint on the application of the basic method in practice. Definitive response time analysis is not possible until the task timing attributes of all tasks are stable. Any subsequent changes arising from system upgrade will typically invalidate previous timing analysis results at least in part and potentially in whole. Some means for enforcing temporal isolation within fixed priority task sets is required to reduce or ultimately eliminate this problem - some techniques that aim to achieve this are discussed in the next section of the review.

2.4.3 Further Advances

Key advances to the original theory that have mainly been achieved more recently and that relate to this thesis include :

- support for synchronous communication and precedence constraints between tasks;
- solutions for distributed systems;
- mechanisms for enforcing temporal partitioning.

2.4.3.1 Synchronous Communication and Precedence Constraints

Liu and Layland's original constraint of task independence is too restrictive for most practical real-time system applications where some form of communication is normally required between concurrent tasks. A number of developments have occurred for static priority-based methods to support communication via synchronous shared data resources. Access to such resources must be controlled such that no more than one task has control of the resource at any given time, at which time the given task is said to *hold* the resource. A task is said to be *blocked* on a shared data resource if it requires access at a time when the resource is held by another task. Some means of controlling the extent of blocking is required so that associated delays may be bounded. In such circumstances, the response time analysis given in equation *(2-4)* can be trivially extended as follows :

$$R_i = C_i + B_i + \sum_{\tau_j \in hp(\tau_i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j \qquad (2\text{-}5)$$

where B_i is the maximum blocking time of the task τ_i.

A side-effect of blocking is the potential for *priority inversion* to occur - whereby a higher priority task is prevented from accessing a shared data resource because it is already being held by a lower priority task (due to the latter being pre-empted before it has released the resource). Mok suggested that a way to avoid this problem was to execute all shared resource accesses as *critical sections* - non-pre-emptable sections of code [Mok83].

A technique for tolerating blocking, known as *priority inheritance*, is described in [Sha87, Sha90]. The priority of a blocked (higher priority) task is temporarily inherited by a blocking (lower priority) task in order that the latter may complete its execution and subsequently release the resource in question. A controlled form of priority inheritance is proposed, the *priority ceiling protocol* (PCP) whereby a fixed priority value (the ceiling priority) is associated with each shared resource. The ceiling priority value is equal to the highest priority of any task requiring access to that resource. At run-time, any task wishing to gain access to a shared resource can only do so if its priority is higher than the ceiling priorities of all shared resources currently held by other tasks. Any task holding a resource will later inherit the priority of any higher priority task it blocks on that resource over the remainder of the time for which the resource is held. PCP guarantees that any task can be blocked at most once (per execution) by any lower priority task. For any given task τ_i, the worst case blocking time is equal to the longest critical section of any lower priority task sharing resource access with τ_i or any higher priority task.

A simpler approach is provided by the *ceiling semaphore protocol* (CSP), proposed in [Rajkumar89, Sha90]. Under CSP, the ceiling priority of a resource is assigned to a task immediately access has been granted. This results in all blocking potential being restricted to the beginning of a task's execution, at the expense of increased average response times. CSP now forms the basis for the Ada95 *protected record* [ANSI95].

Burns and Wellings have introduced *synchronous sessions* - periods of execution during which pre-defined groups of tasks may collectively undertake synchronous communication [Burns97]. All tasks within each communicating group must be released simultaneously and may not be blocked during a synchronous session by any other task that is not a member of the group. A consequence of this latter restriction is that sessions may not overlap, although they may be nested. Sessions are assigned ceiling priorities and each task in the communicating group executes at this priority level whilst the session is active. In order to minimise priority inversions, ceiling priorities are inherited before entry into a session, from either the point of release or upon exit from some previous session. In order for tasks to participate in more than one session without sessions interfering with each other, the ceiling priorities of successive sessions must be non-increasing. An expression for calculating worst-case response times is given as follows :

$$R_i = \sum_{j=1,..,n}^{n} \left\lceil \frac{R_i}{T_j} \right\rceil C_j^i \qquad (2\text{-}6)$$

where C^i_j is the computation time involved for task τ_j at or above priority level i.

Co-operative scheduling schemes [Burns95, Simpson90] provide an alternative means of limiting worst-case blocking times. Tasks are partitioned into non-pre-emptable sections, at the end of which they offer themselves to be descheduled. If a higher priority task is able to then execute, a context switch is instigated, otherwise the original task is allowed to

continue execution of its next non-pre-emptable section. Co-operative scheduling is, therefore, said to involve *deferred pre-emption*. According to [Burns95], a benefits associated with this approach is that the worst-case execution time of each non-pre-emptable section may be predicted with reduced pessimism when the target is a modern processor since pipelines and caches cannot be invalidated by pre-emption during the execution of each section. [Simpson90] also proposes an asynchronous approach to communication via shared data resources, called the *four slot mechanism*. This is aimed at blocking avoidance in the limited case of two tasks (one reader and one writer) sharing a single resource.

The second area of concern addressed in this section is the related issue of dealing with precedence relationships between tasks. Harbour *et al* discussed the direct use of priority levels to deal with synchronous communications, non-pre-emptable sections and precedence constraints [Harbour91]. Tasks with critical sections are decomposed into precedence-constrained sub-tasks. Priorities are then assigned such that each sub-task has an equal or higher priority than any predecessor. Each sub-task has an associated worst-case computation time and a deadline relative to the arrival time of the original compound task. It is observed that, under these conditions, schedulability can be improved when the final sub-task in each compound task is assigned a high priority.

Burns and Wellings categorised methods for the enforcement of precedence constraints in static priority-based solutions into two main classes [Burns95] :

- *deadline transformation* [Audsley93a], whereby task deadlines are increased by the sum of all predecessor task deadlines and then priorities are assigned deadline monotonically; hence the tasks may be released simultaneously but the normal priority-based scheduling mechanism will ensure precedence constraints are enforced;
- *release offsets* [Audsley93], whereby task release times are statically defined relative to a common transaction release time such that each task is only released after all predecessor tasks are guaranteed to have completed.

The deadline transformation approach would appear to be more desirable in terms of flexibility, relying on the standard priority ordering mechanism to enforce precedence rather than imposing explicit constraints on the release times of tasks.

A combination of release offsets and reliance on priority ordering is proposed in [Audsley93c]. Each task is assigning a priority which is lower than that of any of its predecessors and it is ensured that no task is released before the latest release time of any of its predecessors. An extension to equation *(2-5)* is given for calculating response times by introducing release jitter as follows :

$$R_i = C_i + B_i + \sum_{\tau_j \in hp(\tau_i)} \left\lceil \frac{R_i + J_j}{T_j} \right\rceil C_j \qquad (2\text{-}7)$$

Gutierrez *et al* proposed that the accuracy of the response time analysis could be improved by accurate calculation of best-case response time and, in turn, jitter [Gutierrez98] - an iterative form of best-case response time analysis is proposed accordingly. The approach involves examining the best-case phasing of higher priority tasks such that interference is minimised.

Gerber *et al* have proposed a method known as *periodic calibration* for the derivation of task timing attributes from specified end-to-end timing requirements and precedence relationships for a set of static priority-based tasks on a uniprocessor [Gerber95]. The method employs a heuristic-based constraint solver to generate attribute assignments from a series of simultaneous equations involving task priorities, periods, deadlines and offsets. Nilsson *et al* have applied the method to the assignment of timing attributes for an avionics control system [Nilsson98]. The work has since been extended for application to distributed systems given a static allocation of tasks to processing resources [Saksena96].

2.4.3.2 Distributed Systems

Some notable observations regarding distributed (multiprocessor) systems were published in early papers on static priority-based methods. Graham proposed that multiprocessor systems with a single, centralised priority-ordered queue are prone to timing anomalies [Graham69], whereby the response time of some tasks can actually be *increased* by decreasing its execution time, increasing its priority or the addition of another processor. This can arise due to transient impact on the dynamic allocation method such that the task in question becomes allocated to a different processor than would otherwise have occurred. Serlin also examined priority-based solutions in multiprocessor systems [Serlin72] and concluded that, for a given set of independent periodic tasks released at a critical instant, a homogeneous multiprocessor system offers no performance advantages over a uniprocessor solution of equivalent total processing capability.

Mok and Dertuozos later showed that algorithms that are optimal for uniprocessor systems are not optimal for multiprocessor systems [Mok78]. Graham then showed that the problem of optimal scheduling of multiprocessor systems is NP-hard [Graham79]. Chang and Livny later commented that the centralised, priority queue model does not scale well for multiprocessor systems [Chang85] due to the demands on the scheduler becoming increasingly unmanageable as the number of processors in the system grows large.

Chang and Livny also commented on multiprocessor solutions with multiple, physically distributed priority queues, stating that the expected waiting time of a priority class on one processor should be equivalent to that on another processor to give priority a global meaning. Also, tasks of higher priority should be able to pre-empt tasks of lower priority on a system-wide basis. State information exchange and task migration would be required to implement this strategy.

Sha *et al* proposed a common scheduling abstraction for multiprocessor systems [Sha93], based on a static priority-based model with rate-monotonic priority assignment. All task and message releases within the distributed system are time-triggered, allowing individual processors and communication media to be analysed independently via the utilisation-based feasibility test given in equation *(2-3)*. Resources are assumed to be pre-emptable - at the task level for processing resources, acknowledging critical section restrictions, and at the message level for communication media, acknowledging atomicity of individual packets. The technique is applied to the *fibre distributed data interface* (FDDI) cyclic token-based communication protocol, which is modelled as a priority-based resource with bounded priority inversion on the assumption that messages can be priority-ordered locally at each network interface. There is no explicit notion of transactions nor end-to-end deadlines, all scheduling and feasibility testing is localised.

Strosnider has also examined the extension of static priority-based solutions into the communications domain in the context of the IEEE 802.5 *token ring* protocol [Strosnider89]. This is a two-phase demand-adaptive protocol; the first phase (the reservation phase) involves the assignment of the communication medium to a single terminal and the second phase (the transmission phase) allows the exclusive use of the medium by that terminal to transmit its data. The reservation phase involves a 'token' message being circulated between terminals, that are physically connected in a ring, where the token holder ordering is determined either by physical position in the ring or, in the case of Strosnider's work, a priority ordering rule common to all terminals. Priorities are assigned to (periodic) messages according to the rate monotonic scheme. Static priority-based communication models have also been developed in conjunction with *controller area network* (CAN) protocol. CAN is also a form of two-phase demand-adaptive protocol - the reservation phase is priority-based with message level priorities. A detailed timing analysis for CAN was first presented in [Burns95a]. More generally, a key problem with this class of protocol is the maintenance of consistent state information between terminals during reservation phases, particularly in the presence of transmission errors. Inconsistent state will lead to collisions during subsequent transmission phases.

A central part of any scheduling and timing analysis solution for distributed real-time systems is a means for dealing with precedence relationships between tasks and messages, *ie.* how distributed transactions are handled. In [Audsley93], end-to-end transactions are decomposed into component tasks and messages via release offsets relative to the start of the transaction, allowing for worst-case response time behaviour at each stage. The method is applicable to multiprocessor systems provided that a global time reference exists and the communication network can provide bounded message delivery times. This gives rise to time-triggered transactions that span multiple processors but allows priority assignment and response time analysis to be performed on an individual processor basis.

In [Audsley93c], the effects of variability in completion times of tasks on one processor upon the release times of dependent tasks on another processor are examined. The resultant form of response time analysis is the same as that given in equation *(2-7)*. It is again assumed that maximum clock drift is bounded. Significantly, the analysis must be applied to the system as a whole, *ie.* holistically. A task on one processor with immediate predecessors on some other processor will inherit a release jitter (worst-case release delay relative to its arrival) equal to the worst-case completion time of that set of predecessors plus the subsequent worst-case communication delay. At the same time, interference effects (and, to a lesser extent, blocking) make the analysis of task response times inter-dependent on a single processor. The combined impact is that the response time analysis for the multiprocessor system is holistic.

The response time analysis given in [Audsley93c] is applied in the context of both TDMA-based and static priority-based communication protocols in [Tindell94]. Distributed transactions are decomposed into the appropriate combination of time-driven and event-driven tasks and messages. Individual messages inherit their release characteristics from their sender task, *eg.* period, and trigger the release of the destination task upon arrival of the last packet in the message. Where tasks require access to remote shared data objects, these are further decomposed into two separate tasks - the first to issue a request to the remote server and the second to process the data upon its return. Equation *(2-7)* is applied in its recurrence relation form, iterating for each task τ_i until either there is no change in R_i between iterations, *ie.* convergence is achieved, or the condition $R_i > D_i$ holds true, *ie.* the task τ_i is infeasible.

The theory and application of release offsets for distributed transactions is extended in [Bate97, Bate99]. Precedence constrained tasks are assigned offsets such that their release always occurs after all predecessor tasks (including those executed on other processors) and their intermediate communications are guaranteed to have completed. Beyond that, offsets are treated as free variables in the overall task attribute assignment. The offset assignment and corresponding response time analysis are essentially holistic - decisions regarding the offset assignment for one task can affect the feasibility of other tasks in other transactions.

2.4.3.3 Temporal Partitioning

A number of methods are proposed in the literature for providing varying degrees of temporal isolation between tasks and messages that share resources in uniprocessor and distributed systems. A relatively primitive technique known as *background scheduling* has been suggested for scheduling sporadics and aperiodics with soft deadlines. Such tasks are allowed to execute on a FCFS basis whenever there is no (hard deadlined) periodic ready to execute. A number of more advanced techniques have been developed, however, that can collectively be referred to as *bandwidth preserving servers*.

A range of server algorithms are described in [Lehoczky87]. The most basic form is the *polling server* whereby a periodic server task with fixed period and worst-case execution time (termed the server *capacity*) is executed as part of the normal scheduling regime. When the server executes, it checks for any sporadic or aperiodic requests that are ready to execute. If there is one or more such requests to execute, then the server deals with these (in any particular desired order, *ie.* not dependent upon the priority assignment for periodics), otherwise the server is suspended until its next periodic release. The *deferrable server* extends the polling server by allowing it to be reactivated within the same period if a new sporadic task arrives subsequent to the server's suspension (rather than wait to the next period to service that sporadic). The *priority exchange* algorithm allows the server to reduce its priority whenever there is no sporadic request pending - hence other periodics (of previously lower priority) may execute in its place and the server can execute later with a lower priority if required. Unlike the polling server and deferrable server, the priority exchange algorithm allows server tasks to accumulate capacity across period boundaries. For all of these approaches, a maximum server capacity (that can maintain pre-computed periodic guarantees) can be determined as a function of the utilisation of the periodic task set [Buttazzo97].

The *extended priority exchange* algorithm [Sprunt88] has since been proposed for improved sporadic response times, particularly when server capacity is low due to high periodic load. This algorithm does not actually involve the explicit creation of a server task but rather involves the offline assessment of residual processing capacity after a given set of periodic tasks have been guaranteed under rate monotonic analysis. The residual capacity at each priority level is then used to schedule sporadics without affecting periodic guarantees.

The *sporadic server* algorithm [Sprunt89] also offers a solution for handling sporadic tasks with hard deadlines without compromising timing guarantees for periodics. The method works alongside both rate monotonic and deadline monotonic priority assignment. A periodic server task with a hard deadline is created to reserve processing capacity at a given priority level which may then be consumed by an associated sporadic task whenever it arrives for execution. Server capacity is replenished only after it has been consumed. Kuo and Li investigated the use of sporadic servers within an RMA framework for partitioning the feasibility analysis of independent applications on a uniprocessor, where each application consists of multiple tasks [Kuo99]. The total bandwidth of the processor is made available for executing servers that are statically allocated to independent applications. The feasibility of each application can then be argued independently by comparison between its total utilisation requirement and a 'schedulable utilisation factor' calculated for its associated sporadic server.

Lehoczky and Ramos-Thuel developed the *slack stealing* algorithm [Lehoczky92] where the server is notional rather than explicit. The server is notional since it is merely a function that determines the maximum amount of slack time that can be used to execute

sporadic/aperiodic requests in a given interval of time. The algorithm is instigated offline by determining the maximum slack times available over the periodic task LCM, based on worst-case execution times. At run-time, these values are updated to account for actual execution time variations and slack usage. Davis and Burns later proposed an algorithm for determining the available slack time online whenever an aperiodic request arrives [Davis93]. Audsley *et al* have examined the use of slack time for executing *optional* tasks (or optional parts of tasks) without affecting the guarantees of other tasks with hard deadlines [Audsley94]. The basic priority mechanism is used to provide temporal isolation between hard and optional sub-tasks by assigning lower priorities to the latter.

Shin and Chang have proposed a bandwidth reservation approach for guaranteeing periodic tasks whilst maximising the use of residual processing capacity for servicing sporadics [Shin95]. Processing time is discretised into 'unit cycles' equal in length to the highest common factor of all (periodic) task periods. In turn, a 'major cycle' is defined as the interval of time equal to the LCM of the same tasks. A least upper bound for sporadic processing capacity in each unit cycle, over a period of a major cycle, can then be derived using an iterative algorithm by examination of worst-case processor utilisation due to the periodic task set in each unit cycle. The resulting bound can be used as a basis for establishing sporadic task feasibility. Periodic tasks are scheduled using rate-monotonic priority assignment. Sporadics are serviced on a FCFS basis.

Bernat and Burns have the use of capacity sharing between servers to improve average-case responsiveness and/or maximise the likelihood of aperiodic tasks meeting their deadline [Bernat01]. When the capacity of a server is exhausted and there is still work to do, it may steal unused capacity from another lower priority server whilst continuing to run at its original priority. Several different strategies are proposed for identifying which lower priority server from which a given higher priority server may steal capacity, including 'next highest priority first' and 'least capacity first'. Simulation results show that the capacity sharing approach offers better performance than the use of a single server for all aperiodics and the use of a single server per aperiodic.

2.4.4 Summary

There is extensive theory available for static priority-based methods for uniprocessor systems, covering a range of application requirements including periodic and sporadic tasks, hard and soft deadlines, inter-dependent tasks with precedence constraints and synchronous communication. There have been comparatively few results documented for distributed systems, the most notable of these being the treatment of end-to-end transactions through the application of release offsets and release jitter analysis. Jitter analysis, however, is known to be holistic when applied at the system level due to circular dependencies in the timing model. Offset analysis introduces limited partitioning into the timing model through the use of time-triggered transactions but the benefits are correspondingly limited in terms

of flexibility - the initial offset assignment and response time analysis are essentially holistic.

An alternative basis for more flexible, static priority-based solutions for distributed systems potentially exists in the guise of bandwidth preserving server algorithms, initially devised for supporting sporadics and tasks with soft deadlines. These approaches are able to introduce a degree of temporal isolation into the timing model for a uniprocessor task set. The degree of partitioning is currently restricted, however, to providing isolation between the server and the complete set of guaranteed periodics - there is no partitioning within the periodic task set itself.

More fundamentally, it has been suggested several times during the history of real-time systems research that the basic methods proposed initially for uniprocessor systems do not scale well for distributed applications and that new approaches need to be developed for such systems. Mok and Dertuozos made the suggestion in 1978 [Mok78], Chang and Livny reiterated the point in 1985 [Chang85] and still the point is being made in modern real-time systems texts such as by Burns and Wellings [Burns01]. This is not specifically a reflection on static priority-based solutions but rather the fundamental manner in which distributed systems are approached. Rather than taking a particular scheduling or timing analysis approach as a starting point and attempting to 'extend' that approach to deal with distributed applications, the starting point should be 'the system' and its associated end-to-end timing requirements. Consequently, questions should be asked as to what are the most appropriate abstractions for representing and developing the timing and resource characteristics of a distributed system.

2.5 Dynamic Priority-Based Solutions

Dynamic priority-based methods employ the same dispatching policy as static priority-based methods but, as the name suggests, priority values are determined online to reflect the current workload characteristics of the system. Dynamic techniques can be categorised according to the time when feasibility analysis (to establish 'hard' timing guarantees rather than some 'soft' or probabilistic notion of guarantee) is performed (if at all) :

- feasibility analysis performed offline (subject to all timing and resource requirements being bounded and known *a priori*);
- feasibility analysis performed online (subject to all timing and resource requirements being bounded and known at run-time);
- no feasibility analysis (often referred to as *best-effort* solutions).

Dynamic solutions are currently dismissed for real-time systems that contain safety-critical applications, primarily due to difficulties involved with testing, although many techniques do support a degree of offline timing analysis (as suggested in the classification above). The extent to which timing analysis can be performed will be discussed throughout the review.

There have also been several useful contributions in the context of dealing with end-to-end transactions and distributed systems and these will also be discussed. For an extensive discussion of dynamic solutions, see [Stankovic98]. For a more concise presentation of the most notable results, see [Burns01].

Stankovic advocates the use of dynamic online solutions to cope with uncertainty or unrealistic assumptions about behaviour of the environment in which the system will operate, since this may cast doubt over apparent 100% guarantees provided by more static methods [Stankovic92a]. Stankovic argues that static priority-based methods are inadequate for complex real-time systems due to their concentration on scheduling the CPU resource (with insufficient consideration of other resources in the system) and the inability to incorporate importance, urgency and availability (fault tolerance) requirements into a single, static priority value [Stankovic95]. Stankovic also proposes that complex distributed systems require new (dynamic) approaches that are multi-level (catering for different criticalities, hardness of deadlines and fault tolerance) and multi-dimensional (covering a range of system resource types). Jensen goes further and argues that, for the vast majority of (non-safety-critical) real-time systems, a dynamic *best-effort* approach is required, since traditional solutions for hard real-time systems are “inflexible” and “non-cost-effective” [Jensen92, Jensen97].

2.5.1 Classical Results

Liu and Layland discussed dynamic priority-based solutions in the form of the *deadline-driven* algorithm in [Liu73]. This is a pre-emptive scheduling scheme whereby tasks are dynamically assigned priorities at run-time such that the task with the earliest deadline will be executed at any given time - this approach is more commonly referred to as *earliest deadline first* (EDF) algorithm. The scheme is optimal for a uniprocessor task set in the sense that if the task set can be scheduled by any algorithm then it can be scheduled by EDF. An exact utilisation-based feasibility test (for a task set that conforms to Liu and Layland’s original assumptions - see section 2.4.1) is stated as follows :

$$U = \sum_{i=1}^{n} \frac{C_i}{T_i} \leq 1 \qquad (2\text{-}8)$$

Dertouzos showed that EDF is still optimal when the assumption of critical instant release is relaxed, *ie.* task release times are arbitrary [Dertouzos74]. Leung and Merrill later proposed an exact feasibility test for such task sets involving the construction of a worst-case schedule over the LCM of all task periods [Leung80]. Leung and Merrill also briefly considered multiprocessor systems and concluded that their particular approach was scaleable only if some optimal priority assignment approach became available for such systems.

Mok and Dertouzos considered the optimality of dynamic solutions for multiprocessor systems [Mok78]. A dynamic algorithm is optimal if it always produces a feasible schedule whenever a static algorithm (with complete knowledge of all tasks) can do so. The general

observation is made that for most real-world applications optimal algorithms do not exist. It was later observed that dynamic algorithms that have no *a priori* knowledge of task arrival times, computation times and deadlines cannot guarantee optimal performance [Dertouzos89]. Ramamritham *et al* have stated that it is impossible to find an optimal schedule for a dynamic distributed system given the inherent communication delays in the system [Ramamritham89].

Dynamic best-effort solutions are driven by an online *value* function, early work on which was performed by Jensen and Locke [Jensen85]. Locke has since proposed an approach that employs EDF to schedule independent tasks during normal system operation but switches to a policy of 'maximising value' during intervals of transient overload [Locke86]. This approach was motivated by the observation that straight forward EDF solutions may miss deadlines in an unpredictable manner during transient overload - to the extent that the effective throughput of the system can theoretically approach zero. Later, in [Jensen92], a wide range of value (or *benefit*) functions have been discussed, including examples for tasks with hard and soft deadlines, cases of negative value and combined value functions which cater for both urgency and importance.

2.5.2 Basic Characterisation

For the type of applications envisaged by this thesis, *ie.* which require *a priori* timing guarantees to be available, dynamic best-effort solutions and dynamic solutions with online feasibility analysis are clearly inappropriate. Such solutions are aimed at a different class of systems where total workload is unbounded *a priori*. Dynamic solutions that do support offline feasibility analysis are potentially appropriate from a purely theoretical viewpoint, even though these represent a significant practical challenge in terms of certification for safety-critical applications.

Offline analysis for dynamic scheduling schemes is generally presented in the form of utilisation-based tests - there is no concise form of response time analysis available for EDF and related dynamic schemes. The simple test given in equation *(2-8)* is a sufficient and necessary condition for feasibility of the whole task set but, unfortunately, all it tells us about the worst-case response time of an individual task is that it is less than the corresponding deadline. More recent work has shown that determining worst-case response times in dynamic priority-based systems is non-trivial [Stankovic98]. This is because the worst-case does not necessarily occur for all tasks in the first 'busy interval' after the task set is released synchronously, *ie.* the time from the first critical instant release until the processor is first idle again, as is the case for static priority-based solutions. The problem arises due to the fact that the relative task priorities vary over time, depending on their relative deadlines (in the case of EDF) and, in turn, on their relative arrival times. Consequently, the relative priority orderings that occur during the first busy interval are not necessarily those which give rise to the worst-case response times for all tasks. That said,

worst-case response times *can* be determined for dynamic scheduling schemes by observing different critical instants for each task [Stankovic98].

Another concern associated with dynamic solutions is that the run-time overhead resulting from determining priorities online can be significant. Stankovic argues that this problem can be overcome by the provision of specific hardware support for the scheduler [Stankovic95].

Even though there appear to be some significant drawbacks associated with dynamic priority-based solutions for applications that require offline feasibility guarantees, some further advances are discussed below. This is in recognition that some useful contributions have been made in the context of dealing with end-to-end transactions and distributed systems. The emphasis is entirely on solutions that support a degree of offline feasibility analysis. Further information regarding best-effort and online techniques can be found in several publications - notable results include heuristic-based multiprocessor scheduling solutions [Ramamritham84, Zhao87, Cheng86, Stankovic91, Stankovic95]. See [Stankovic98], [Liu00] and [Buttazzo97] for extensive information on dynamic solutions.

2.5.3 Further Advances

Significant advances have been made in the theory and application of dynamic priority-based methods since its foundations were laid in the 1970s. As in section 2.4.3 for static priority-based solutions, these results are categorised in terms of :

- support for synchronous communication and precedence constraints between tasks;
- solutions for distributed systems;
- mechanisms for enforcing temporal partitioning.

Much of the recent work on dynamic solutions has paid significant attention to this latter issue - temporal partitioning - and this is reflected in the relative lengths of the review sections.

2.5.3.1 Synchronous Communication and Precedence Constraints

A range of priority inheritance and priority ceiling protocols have been developed for dealing with synchronous communication in dynamic priority-based solutions that are analogous (though generally more complex due to dynamically changing priorities) to those developed for static priority-based solutions. For example, the *priority inheritance protocol* (PIP) [Sha90] has been adapted for EDF scheduling, as described in [Stankovic98]. Dynamic PIP offers a form of task set level feasibility analysis (for a task set that otherwise conforms to Liu and Layland's original assumptions - see section 2.4.1) as follows :

$$\forall i = 1,\ldots,n; \sum_{j=1}^{i-1} \frac{C_j}{D_j} + \frac{C_i + B_i}{D_i} \leq 1 \qquad \textit{(2-9)}$$

The test is sufficient but not necessary and can be applied offline. Similarly, in [Chen90], the *priority ceiling protocol* (PCP) [Sha90] is adapted for EDF scheduling, giving the following feasibility test which looks like an intuitive extension of the exact feasibility test for (non-blocking) EDF, as given in equation *(2-8)* :

$$\sum_{i=1}^{n} \frac{C_i + B_i}{T_i} \leq 1 \qquad (2\text{-}10)$$

Once again, the test is sufficient but not necessary and can be applied offline. Whilst the net results of these dynamic protocols are analogous to their static counterparts in the manner of achieving mutual exclusion over shared resources, it is considerably more difficult to determine worst-case blocking times. [Stankovic98] describes some such techniques.

A scheme that is applicable to dynamic and static priority-based systems, the *stack resource policy* (SRP), is proposed in [Baker91]. According to [Burns01], the scheme provides analogous behaviour in dynamic solutions to that resulting from the *ceiling semaphore protocol* (CSP) [Sha90] in static priority-based solutions in that a task can be blocked at most once and this is restricted to the start of its execution. The scheme also provides a task set level feasibility test is sufficient but not necessary and can be applied offline

Jeffay examined EDF scheduling of sporadic tasks that communicate synchronously through shared data resources [Jeffay92a]. Each sporadic task is assigned an overall deadline equal to its minimum inter-arrival time and then decomposed into a linear sequence of resource requests, each with an associated maximum computation time. Priority inversion effects are minimised through the dynamic modification of deadlines to encourage early release of resources.

Precedence constraints can also be treated in EDF schemes in a manner that is similar to that proposed for static priority-based schemes, via the transformation of task deadlines [Stankovic98]. It is also assumed that no precedence-constrained task may be released prior to any of its predecessors. The approach can be applied in conjunction with the synchronous communication protocols described above.

2.5.3.2 Distributed Systems

A wide range of dynamic solutions have been proposed for scheduling multi-access communication media, point-to-point packet-switched networks and, ultimately, distributed real-time systems. This section describes some notable results in each of these areas in turn.

Dynamic solutions that have been proposed for controlling communication medium access include forms of the contention-based protocols introduced in section 2.1 (although, strictly speaking, these do not directly embody the notion of priorities). This includes forms of the *carrier sense multiple access* (CSMA) group of protocols that employ dynamic policies for resolving collisions (multiple terminals attempting to transmit at the same time) at run-time. There have been numerous protocols proposed in this class over the last 30 years (see

[Tannenbaum96] for details) but the basic *ethernet* protocol [Metcalf76] is perhaps the best known. The basic ethernet protocol enforces random delays at each terminal following the detection of a collision prior to retransmission attempts. The maximum retransmission delay is bounded by a value that increases each time a collision is detected but, since there is no guarantee that a collision will be avoided for any particular transmission attempt, worst-case communication delays are unbounded. Bounded re-transmission strategies have since been proposed, most notably based on the sub-division of transmission time windows.

For example, the *virtual time* CSMA protocol [Molle81] makes use of a system-wide 'virtual clock' maintained at each terminal, being activated whenever the medium is sensed free and halted when the medium is sensed busy. The time associated with the communication medium is slotted and a terminal is enabled at the start of a slot if data has become ready for transmission within a given time window (calculated based on currently active communication requests). Transmission is attempted for all enabled terminals and if a collision is detected then the time-window is repeatedly sub-divided with only the terminals in one of the resulting partitions remaining enabled, until a single enabled terminal is isolated and allowed to transmit if necessary.

A significant amount of attention has been paid in recent times to packet-switched network technologies and their potential application to real-time systems. The simple *first-come-first-served* (FCFS) strategy is widely applied in such networks for scheduling packets at network entry points and intermediate switch nodes and is a form of dynamic scheme - the actual arrival times of packets at network entry points and intermediate switch nodes dictates the order in which these are serviced. Although there is no explicit recognition of delay requirements (deadlines) at run-time, such schemes are often predictable in the worst-case and offline analysis can be performed to determine whether or not worst-case delays are acceptable. The analysis of such networks is, however, far more complex than that of a single FCFS queue, where the worst-case delay is equal to the product of the maximum queue size and the per-packet time. The analysis must take into account the fact that each intermediate switch will provide multiplexing between several input and output links, each served by a separate FCFS queue. A key factor in the model will normally be the arrival jitter associated with packets as they traverse the network - this impacts on the maximum queue length (from which delays are determined) and, in turn, given that queue size is limited by physical buffer size, governs the extent of packet *congestion*. Congestion is said to occur whenever the queue size exceeds that which is physically available and, as a consequence, packets can be lost or allowed to back up at previous switches and ultimately at source.

The approach to synchronous communication described in section 2.5.3.1, from [Jeffay92a], has been applied in the context of end-to-end video-conferencing applications in a distributed system [Jeffay92b]. The overall solution uses an event-triggered transaction model but with periodic 'servers' to poll for communications arriving at packet-switch

network nodes. Consequently, this gives rise to relatively stable end-to-end delays in the presence of variable system load at the expense of increased latencies. Poor resource utilisation also results due to the need to specify pessimistic minimum inter-arrival times for the periodic servers to allow for the effect of network congestion on packet delivery.

DiNatale and Stankovic have proposed a method for providing end-to-end timing guarantees in a dynamic distributed system [DiNatale94]. Transactions are decomposed into a number of precedence-constrained tasks and messages, which are then statically allocated to *time-windows* on individual system resources accordingly, *ie.* transactions are time-triggered. Local schedulers are then free to execute the portions of the transaction for which they are responsible within their given time-window, without explicit recognition of precedence constraints. In this case, scheduling is non-pre-emptive. Each transaction is assigned a 'master' task which makes the scheduler of the processor to which it is assigned responsible for triggering the arrival of the transaction on other processors. As is typical of time-triggered transaction models, it is assumed that there is a known upper bound on clock drift between processors.

2.5.3.3 Temporal Partitioning

Numerous techniques have been proposed for providing various degrees of temporal partitioning in dynamic priority-based solutions. The first approaches described below are aimed at providing partitioning between groups of tasks - periodic *v* sporadic tasks or critical *v* non-critical tasks.

Chetto and Chetto devised an online feasibility test for acceptance or rejection of independent sporadic tasks in the presence of independent periodic tasks on a uniprocessor [Chetto89]. The test is based on calculating the slack time available between the arrival time of the sporadic and its deadline and so the acceptance of any new sporadic task does not affect the schedulability of periodics. In [Chetto90], the test is extended to precedence-related groups of sporadic tasks on a uniprocessor. The release times and deadlines of the precedence-related task groups are, in fact, initially transformed such that the sporadic tasks involved may be scheduled as independent. In order to improving the likelihood of acceptance, the test proceeds on the basis of *remaining* computation times for previously guaranteed tasks.

Buttazzo and Stankovic have developed a version of EDF with online feasibility testing that can guarantee deadlines will be met for designated 'critical' tasks even during transient overload conditions. The algorithm is known as *robust earliest deadline* (RED) [Buttazzo95]. All tasks are, in fact, treated as aperiodic, *ie.* minimum inter-arrival times are unknown *a priori*. Whenever a new task arrives for execution at run-time, the algorithm calculates the slack time available for each task, based on a pre-defined computation times and deadlines, along with a profile of the expected processor workload up to the latest deadline. This gives an indication of which, if any, tasks can expect to miss their next

deadline and allows for some recovery mechanism to be instigated early if required. Expected workload and task feasibility are re-evaluated each time a task completes execution, hence reclaiming any unused processor capacity. Critical tasks can maintain their timing guarantees throughout periods of overload since tasks with lower *value* - a pre-assigned indicator of relative importance - can be removed to the required extent.

Clearly, the degree of temporal partitioning provided by the above approaches is limited in that individual tasks are not isolated from each other within the defined groupings. A number of dynamic solutions aimed at providing partitioning between individual tasks have also been developed. Notable results are discussed below. Whilst none of these schemes provides any form of response time analysis, they are interesting in the context of this research from the following perspectives :

- the manner in which task execution requirements are specified;
- the manner in which the basic EDF scheduling scheme is adapted to provide independent execution rate control;
- the extent to which schemes have been applied consistently to the scheduling of both CPU resources and communication media.

Mercer *et al* have applied EDF scheduling in conjunction with a utilisation-based admission test for reserving processor capacity for individual real-time tasks in a multimedia environment - referred to as *processor capacity reservation* (PCR) [Mercer94]. Bandwidth is allocated to each task according to a utilisation-based specification of *x* computation time units over a period of *y*. This is a generalisation of the standard periodic execution model, although the approach does not directly ensure task feasibility but relies on task timing requirements being reflected in the utilisation-based specification of bandwidth requirements. Lee *et al* have applied the PCR method to multimedia applications with end-to-end timing requirements executing in a distributed system [Lee96]. Distributed applications are decomposed into sequences of periodic tasks, allocated statically and reserved processor capacity at each stage. Worst-case end-to-end response times are equal to the sum of the periods of the periodic tasks in each application. The real-time Mach operating system [Lee 96a] has been used as a test-bed for demonstrating the PCR approach as part of the construction of dependable real-time services on top of a COTS hardware/software platform [Abdelzaher99].

Jeffay *et al* have proposed a slightly different form of periodic specification for reserving processor capacity [Jeffay95, Jeffay99] - *x* task executions every *y* time units, referred to as a *rate-based* specification. Each task is released upon the occurrence of some pre-assigned event, typically involving the arrival of a message, and each has a specified computation time and deadline. Tasks are designated as either *source* or *internal* based on the nature of their release events - either external to the system or internal. Internal tasks ultimately inherit their rate-based specification parameters from source tasks, possibly via other

predecessors. The motivation behind event-driven release of tasks is to minimise average end-to-end response times. The actual execution rate of each task is dynamic and depends on the actual arrival rate of source events from the environment - task release is therefore prone to jitter. Whenever the specified maximum arrival rate is exceeded, this results in either the execution rate of the task being increased, if resources are available, or the effective loss of events. Given an EDF scheduling policy, feasibility of the task set as a whole can be analysed by determining the worst-case loading on the CPU as a function of the rate specifications [Jeffay99]. The Real-time Mach operating system [Lee 96a] has again been used as a test-bed.

Stoica, Jeffay *et al* have proposed a form of online reservation-based scheduling solution known as *earliest eligible virtual deadline first* (EEVDF) [Stoica96, Stoica97]. Tasks are allocated a fixed proportion of the overall capacity of each required resource and can be guaranteed to make progress at a defined rate. EDF scheduling is applied at an abstract level based on *virtual* time, which is progressed one 'tick' after all tasks in the currently active set have each been granted access to the processor a specified number of actual time slots. Processor capacity can thus be reserved for each task by allocating a fixed proportion of actual time slots within each virtual time unit. A time unit on a processing resource may be defined as some fixed interval of time for which an executing task is granted uninterrupted access to the processor - although it may release the task voluntarily before the end of that interval. Where tasks communicate via shared data resources, processor time units may be assigned a duration equal to the maximum length of any critical section of any task, in order to be assured of avoiding pre-emption during such times. A time unit on a physical communication link may be defined as the interval required to transmit a fixed length packet of data. Task execution rates can be adapted online and dynamic task creation and destruction are also supported.

A number of dynamic priority-based schemes have been suggested for implementing the generalised processor sharing (GPS) model discussed earlier in section 2.3.3 [Parekh93, Parekh94] that was originally proposed for scheduling independent traffic flows in packet-switched networks. The first adaptation of GPS for processor scheduling was actually *probabilistic* - this was developed for multimedia applications and referred to as *lottery scheduling* [Waldspurger94]. As the name suggests, a number of ‘lottery tickets’ are allocated to each task according to its CPU requirements. The CPU is then allocated in fixed time-quanta to a lottery winner selected at random. Lottery tickets can be re-allocated dynamically to reflect changing application requirements. On average, each task will be allocated an amount of CPU time that reflects its currently allocated proportion of lottery tickets. A deterministic variant was later proposed, termed *stride scheduling* [Waldspurger95], whereby the ‘stride’ value of a task is inversely proportional to its current allocation of lottery tickets. The task with the smallest stride value is scheduled in each CPU

time quanta. Stride scheduling gives a more accurate implementation of GPS than lottery scheduling.

A scheme referred to as *earliest completion time GPS* (EGPS) is proposed in [Kuo98]. EGPS supports both hard and soft timing requirements and allowing dynamic control of execution rates. Tasks are prioritised as they arrive according to predicted earliest completion times under pure GPS. Predicted completion times are dependent on the current load on the CPU and so must be re-calculated for all active tasks upon arrival of any new task.

Deng *et al* have developed a GPS emulation scheme known as the *constant utilisation server* [Deng97, Deng99] aimed at scheduling independent real-time applications in a uniprocessor workstation/PC-based open system (Windows NT OS). At the higher level, CPU bandwidth is allocated to constant utilisation servers, one for each real-time application and one for all other (non-real-time) applications. Servers are scheduled according to an EDF policy. At the lower level, each application can implement its own internal task scheduling mechanism (RM, DM, EDF, with local resource protocols like PCP). The budget of each constant utilisation server is replenished periodically on the basis of predicted earliest arrival times of events within the application and the appropriate intermediate task execution requirements.

The constant utilisation server is actually a restricted version of a previously developed server approach known as the *total bandwidth server* [Spuri96] - originally proposed for partitioning aperiodics from hard periodic tasks. In the former scheme, the server's bandwidth is replenished periodically when the server's deadline occurs. In the latter scheme, the server's bandwidth is replenished as soon as it has been consumed if there are tasks ready to run. Thus, the total bandwidth server is more demand-driven and can exploit unused/unreserved processor time whereas the constant utilisation server cannot. Consequently, the total bandwidth server generally provides better average-case responsiveness. The constant utilisation server is not applicable if any shared synchronous communication resources exist or, analogously, if any application contains non-pre-emptable code. The total bandwidth server is not subject to this restriction but the ability to partition application feasibility analysis is lost - the analysis of each application depends upon the maximum non-pre-emptable section duration determined across all applications.

The constant utilisation server has since been adapted by Abeni and Buttazzo to maintain server independence when tasks over-run their predicted worst-case execution times, giving rise to the *constant bandwidth server* (CBS) scheme [Abeni98, Abeni98a]. A further adaptation of the constant utilisation server has also been proposed for scheduling traffic in packet-switched networks [Zhang99]. Other server-based schemes have been proposed based on EDF - see [Buttazzo97] for further details.

A non-pre-emptive dynamic scheme, sometimes referred to as *packetised GPS* but more often as *weighted fair queuing* (WFQ), has been proposed for servicing sporadic/aperiodic traffic at communication network interface nodes [Demers89]. In its simplest form, WFQ applies WRR at the byte level to determine expected completion times of packets awaiting transmission at a network node or switch. The calculation is weighted such that particular virtual communication channels receive a specified proportion of the available bandwidth. Expected completion times are then interpreted as deadlines and the packets transmitted in EDF order. Ermdahl *et al* have compared the performance of a WFQ implementation for an ATM network with one employing priority-based scheduling with Audsley's optimal priority assignment [Ermdahl97]. The priority-based implementation was found to be superior in terms of responsiveness and effective bandwidth utilisation. The WFQ implementation was superior in terms of bandwidth partitioning and simplicity of feasibility analysis, including the ability to rapidly accept or reject new connection requests online for reconfiguration or dynamic applications.

A similar approach, the *virtual clock* algorithm, has been proposed in [Zhang91] aimed at providing independent delay control for virtual communication channels in a point-to-point packet-switched network. Associated with each virtual channel is a 'virtual clock' which ticks every time a packet arrives for transmission on that channel. The duration of the tick is equal to the size of the packet divided by a specified 'reservation rate' value for the channel (a fraction of the total bandwidth available from the associated physical link). Packets are time-stamped with the corresponding virtual time and subsequently transmitted in increasing virtual time order. Yau and Lam have adapted this approach to provide a dynamic scheduling solution for multimedia applications that supports the notion of independent guarantees of minimum execution rates for individual tasks [Yau96]. Tasks are guaranteed at least kr_iT_i CPU time in any interval of time kT_i; where r_i is the reservation rate of the task ($0 < r_i \leq 1$), T_i is its period and k is a number of task periods ($k = 1,2,\ldots$). Guarantees are maintained even when other tasks exhibit behaviour beyond initial assumptions, *eg.* arrive more frequently than anticipated or require more CPU capacity - such tasks can only invalidate their own guarantees. Tasks are scheduled according to EDF but with deadlines repeatedly recalculated to reflect expected completion times based on elapsed time, elapsed execution times and specified execution rates.

Kandlur and Shin have described a related solution for scheduling traffic in packet-switched communication networks [Kandlur91]. Expected completion times are calculated for each packet as it arrives for transmission using static priority-based response time analysis [Joseph86, Audsley91]. Packet transmission then occurs in order of expected completion times. Priorities are re-assigned (based loosely on deadlines associated with each channel and on the current workload) each time a packet arrives for transmission. The approach has been applied in the HARTS project [Shin92]. A similar approach has also been applied in

the Tenet project [Ferrari90, Ferrari95], which has also considered other schemes including best-effort and statistical guarantee approaches for non-real-time traffic.

Baruah *et al* [Baruah96] have proposed a scheduling approach known as *pfair* which allows periodic tasks to make progress at approximately a uniform rate, irrespective of other tasks that share resources. Both task execution times and CPU resource time are quantised into fixed-length unit time-slots. Intermediate deadlines are calculated for all unit-length sub-tasks so that the feasible execution of sub-tasks will deliver an approximately uniform rate of execution for the parent task. Sub-tasks are then scheduled in deadline order (when sub-task deadlines are equal, ties are broken by examining task level deadlines). A parameter referred to as the *lag* is determined for each task as the difference between the specified and actual execution rates. The resulting schedule is designated as 'pfair' if the lag for all tasks can be shown to be bounded by the interval [-1, 1]. Whilst the theoretical model has some desirable properties in terms of task isolation, a key practical draw-back is identifiable. One of the key assumptions that underpins the scheduling model and associated analysis is that any task (or sub-task) can be pre-empted on one processor and re-started on another at any time. This practical issue has since been investigated to a limited extent in [Moir99] which examines the impact of fixing the allocation of some (but not all) tasks to specific processors. This work has concluded that it is still possible to provide a pfair scheduling solution in such cases at the expense of increasing the complexity of the algorithm (from polynomial to exponential).

2.5.4 Summary

A range of dynamic priority-based approaches have been discussed that have been developed for distributed real-time systems, many of which support distributed transactions and temporal partitioning (at some level). Target applications are also invariably dynamic in the sense that it is not practical to impose *a priori* bounds on system timing and resource requirements. Many of these schemes, however, do support utilisation-based feasibility analysis and could, theoretically at least, be applied for systems that require offline timing guarantees where timing and resource requirements can be bounded *a priori*. In practical terms, however, these approaches may still be inappropriate for safety-critical applications due to difficulties in testing, a lack of predictability under fault condition (such as when actual execution times or arrival rates exceed stated requirements) and, more generally, a lack of predictability during normal operation when compared to cyclic and static priority-based solutions.

From the perspective of this research, it is interesting to observe the use of a 'virtual' or 'abstract' scheduling model, such as GPS [Parekh93, Parekh94] or virtual clock [Zhang91], to provide an idealistic execution model which is then approximated by some actual underlying scheduling implementation. This idea provides a basis for consistent scheduling and analysis across processing and communication resources in a distributed system.

It is also interesting to note that a number of proposed dynamic schemes provide fine-grained bandwidth allocation *without* resorting to fine-grained time-slicing (as exhibited by WRR-based cyclic schemes). These include the constant utilisation server [Deng97], total bandwidth server [Spuri96], proportional share [Stoica96], weighted fair queuing [Demers89] and virtual clock implementations [Zhang91, Yau96].

2.6 Industrial Trends and Perspectives

Current industrial practice in the development of real-time systems, especially those that contain safety critical functionality, is predominantly to adopt cyclic 'time-triggered' solutions as described in section 2.3.1. In the avionics domain, these solutions are applied as part of a *federated* architecture design approach where specific functionality is allocated to (and often developed alongside) dedicated hardware subsystems. Problems with complicated and expensive system upgrade programmes, however, have prompted the search for more flexible solutions for future avionic systems, generally referred to as *integrated modular avionics* (IMA). IMA represents the primary target application domain for this research, although the basic principles involved are generally applicable to a wider range of real-time systems applications in industry. The rest of this section provides a brief history of IMA development efforts.

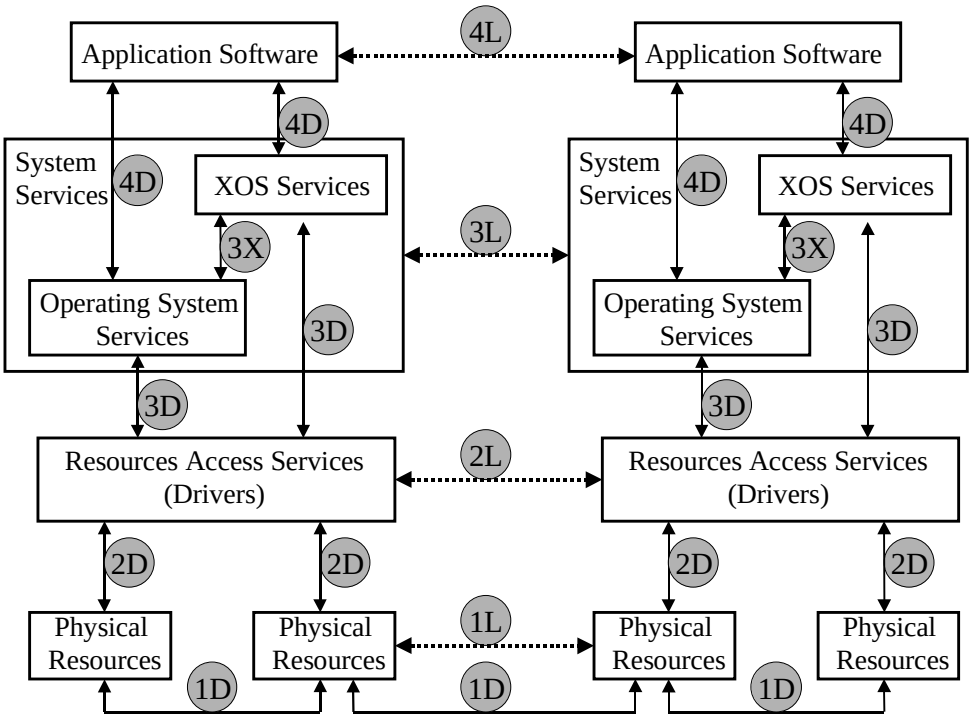
The first (large-scale) programme to investigate the potential for moving away from federated architectures towards IMA was the US Air Force *PAVE PILLAR* 'technology demonstrator' programme [AFAL87]. The project concluded that an open systems approach could potentially reduce the life-cycle cost of an avionics system by up to 50% through the provision of a common core processing system with standardised hardware and software interfaces. This represents a significant saving in view of more recent studies into aircraft life-cycle costs where it has been suggested that development and maintenance/upgrade of avionics and related computer-based systems account for around one third of combat aircraft life-cycle costs [Lachenmaier97]. Some of the IMA architecture principles that arose from PAVE PILLAR, particularly the hardware aspects, influenced the development of the F-22 Raptor avionics system, giving rise to the F-22 *common integrated processor* (CIP) architecture [Benedict98] - a common core processing architecture for F-22 avionics aimed at exploiting COTS hardware.

Awareness of the IMA-related activities in the US prompted the UK MoD to begin to fund IMA-related research programmes in the mid-1980's, most notably the *advanced avionic architecture programme* (A^3P). There is little information in the public domain regarding A^3P but see [Fabian95] for a brief MoD perspective. The overall objective of the programme was to investigate the technical and practical feasibility of IMA. A more specific objective was to reduce the number of standard hardware modules specified for the core architecture compared to that for PAVE PILLAR. Before A^3P could deliver much in the way of practical demonstrations or module standards, however, European-wide collaborative IMA programmes were instigated, most notably the *allied standard avionics architecture council*

(ASAAC) programme [Edwards94, Multido98], which continues to be active to the present. [Edwards94] gives the rationale for adopting an open systems approach and describes the basic modular architecture that emerged from the first stage of the programme. One of the key architectural requirements is identified as hardware *interchangeability* - the ability to replace an existing system hardware module with another of the same specification but different implementation (possibly from a different supplier). [Grigg94] identifies a number of issues arising from this objective including the need for more modular/partitionable approaches to timing and safety analyses. The second (current) phase of ASAAC has continued to develop the architecture definition for IMA, most recently described in [Multido98]. A range of standards is proposed as follows :

- hardware modules - including modules for general purpose processing, signal processing, graphics and sensor/actuator interfacing;
- communication network - a technology-independent network model, instantiated in the current 'architecture baseline' via ATM;
- layered software model [Wake96, Field97, Kemp00] - consisting of an application layer, real-time operating system and hardware interface layer;
- other standards - including power management, packaging and cooling.

The *generic open architecture* (GOA) programme [SAE96] is a standardisation programme that was instigated outside the aerospace domain, by the US Society of Automotive Engineers (SAE), but is now being examined as an alternative framework for military IMA architectures. A number of parties have become involved since the programme commenced, including the US Army, Navy and Air Force, NASA, the UK MoD and French and German military establishments, numerous industrial prime contractors and suppliers and a small number of US academic establishments. The GOA model (as illustrated in figure 2-6) is comprised of *architecture layers* and *interface classes* which provide a framework for classification (and comparison) of other open system models. The GOA model was, in fact, initially developed as an adaptation of the *space generic open avionics architecture* (SGOAA) [LESC94], instigated in 1991 by NASA with the aim of developing a family of architectures that could support a wide range of future space missions. SGOAA programme drivers were similar to IMA - to reduce life-cycle costs, shorten project schedules and improve technical performance. More specific design objectives for SGOAA were to allow interoperability and interchangeability of system components and mission procedures across a range of space vehicles and programmes, to minimise obsolescence costs through technology transparency and to support increased physical integration.



KEY

D - *direct* interface

L - *logical* interface

X - *extended* interfaces

Figure 2-6 : Generic Open Architecture Model

GOA *logical* interfaces describe peer level abstract communication mechanisms that can be configured with specific details of sender/receiver identification, data formats and timing requirements for a given system. *Direct* interfaces identify key implementation stages for logical interfaces, *ie.* they describe how the information flow specified in the logical interfaces is implemented, *eg.* via message passing or shared memory protocols. The *application software* layer contains application-specific functionality/code and communication requirements. The *system services* layer, sub-divided into *operating system* (OS) and *extended operating system* (XOS) services, provides an abstract/virtual machine model upon which the applications are executed. The difference between OS and XOS services is merely the degree of generality - OS services are genuinely generic whereas XOS services may be system-specific. Services in this layer include 'fault management', 'communication management', 'database management' and 'system control'. The *resource access services* layer provides a set of physical device interfaces (drivers) that are ultimately hardware-dependent. The *physical resources* layer encompasses the entire set of system hardware components and their physical interconnections.

An adaptation of the GOA model, the *technical reference model* (TRM) [DoD99], is currently being developed by the US DoD for military applications. In turn, the TRM itself

is being extended by collaboration between US defence organisations to cover military 'system of systems' applications, giving rise to the *joint technical architecture* (JTA) [DoD99a]. Interestingly, the JTA is explicitly aimed at being entirely forward-looking, focussing on emerging standards without regard to legacy systems. The GOA model and its successors in the form of TRM and JTA have subsequently influenced the Joint Strike Fighter (JSF) *avionics architecture definition* (JAAD) [JSF97], coordinated by the JSF Program Office.

The primary IMA standardisation programme that exists in the civil aerospace domain is coordinated by a consortium known as *aeronautical radio incorporated* (ARINC). The goals of ARINC are similar to those of ASAAC and standards are actually more mature in terms of both detail and issue status. Some of these standards are of particular relevance in the context of this review. Firstly, the ARINC 629 standard [AEEC94] describes a TDMA-based multi-access databus protocol. A detailed description of the protocol and an assessment for real-time applications can be found in [Grigg96]. The basic protocol configuration is aimed at supporting periodic, fixed length (or bounded variable length) messages - transmission 'bursts' are non-pre-emptive and so the addition of sporadics gives rise to variable cycle times, introduces significant periodic release jitter and so compromises schedulability. An extension to the basic protocol, referred to as the *combined protocol*, has been defined to enforce fixed cycle times so that sporadics and periodics can more readily co-exist in the same schedule. The ARINC 629 protocol (in its basic configuration) has been implemented and applied on the Boeing 777 where separate 629 buses are employed for flight controls and general avionics functions [Berger95].

A second ARINC standard that is particularly relevant in the context of this review is ARINC 653 - an API known as APEX [AEEC97]. APEX advocates a two-tier uniprocessor-level scheduling approach that is cyclic at the higher level and priority-driven at the lower. The cyclic scheduling provides application *partitions* which are temporally isolated from each other - the motivation being to support safety-critical and non-safety-critical applications on the same processor. Timing analysis for APEX-based applications has been developed by Audsley and Wellings [Audsley96], where the problem of release jitter is highlighted as a key factor in reducing schedulability. The APEX scheduling approach will be considered in detail as a target scheduling model later in the thesis (section 6.4).

2.7 Review Conclusions

The review has served to illustrate the complex nature of the scheduling and timing analysis problem for distributed real-time systems and the diversity of solutions proposed. Whilst extensive results are available, there is no generally agreed solution, nor even a widely-preferred class of solutions within which to work. The choice of most appropriate solution is clearly dependent upon the nature of the application requirements and the desired characteristics of the system engineering processes and target system architecture involved. Consequently, such details should be taken into account during the selection of a scheduling

and timing analysis solution for a given system. Recognition should be given to the desired characteristics of the system engineering process, such as the desire to incrementally evolve the system design and implementation, as much as the characteristics of the target system architecture.

It has been suggested several times throughout the course of real-time systems research that the basic methods proposed initially for uniprocessor systems do not scale well for distributed systems and that new approaches need to be developed for such systems. Mok and Dertuozos made the suggestion in 1978 [Mok78], Chang and Livny reiterated the point in 1985 [Chang85] and still the point is being made in modern texts such as by Burns and Wellings [Burns01]. This reflects the manner in which the development of techniques for distributed systems is typically approached, by taking a particular scheduling or timing analysis approach as a starting point and attempting to extend or adapt that approach to cater for distributed system applications. Instead, questions should be asked as to what are the most appropriate abstractions for representing and developing the timing and resource characteristics of a distributed system. Scalability and composability should be key drivers if the resulting techniques are to be applicable in practice for large-scale real-time applications. The ability to partition the overall system timing analysis model - to reason about the temporal behaviour of the system in part as well as in whole - is fundamental to the development of a scaleable solution.

The most promising results to date in this context are those timing analysis and associated scheduling schemes that support bandwidth-based (or, analogously, rate-based) expression of timing and resource requirements. These schemes offer a degree of abstraction from the target platform in the way that requirements are specified and reflect a trend towards more consistent analysis of communications and processing resources. The schemes described in the review, however, are invariably aimed at dynamic applications and employ EDF scheduling at some level - even those that use the idealised WRR cyclic model at the abstract level. Consequently, the amount of bandwidth reserved or the execution rate delivered at run-time tends to be dependent on the actual degree of competition for the resource at run-time, *eg.* as the total demand on the resource increases, the bandwidth reserved for a given task will decrease in absolute terms. Such solutions are more accurately referred to as *proportional share* methods than *bandwidth reservation* methods.

In conclusion, the general notion of *bandwidth-based* specification of resource requirements (and its *rate-based* analogue) is a potentially powerful abstraction around which to construct a scheduling and timing analysis solution for distributed real-time systems. This idea will be pursued in conjunction with an appropriate model of end-to-end timing behaviour in the remainder of the thesis to meet the research objectives described in chapter 1.

CHAPTER 3. Rationale and Basic Method of Solution

This chapter presents the rationale for the development of a timing analysis model to achieve the objectives defined in chapter 1 :

- the ability to model the timing behaviour of parts of the system independently from other parts and, correspondingly, limit the scope of re-analysis required in the event of localised changes;
- the ability to assess the timing behaviour of the system throughout the development process, based on the latest information available at each stage.

The rationale, which also serves as an introduction to the basic method of solution, begins with a discussion about the need for a requirements-driven derivation of the timing analysis model. The remainder of the chapter then addresses each of the above objectives in turn.

3.1 Requirements-Driven Derivation of the Timing Analysis Model

It can be seen from the survey in chapter 2 that the formulation of a timing analysis model for a distributed real-time system can be instigated but not normally completed until the implementation and integration details of the final system are known. In particular, the details of the scheduling solution adopted on each of the processing nodes in the system will dominate the form of the timing model. For example, the choice of cyclic or fixed priority scheduling schemes will normally lead to radically different models. This is similarly the case for details of the communication protocols employed. For example, the choice of ARINC 629 [AEEC94] or ATM [Boisseau95] would typically lead to completely different models for determining communication delays. Models can differ fundamentally in terms of their basic parameters and in terms of the degree of abstraction involved. This approach to the development of a timing model for a real-time system can be said to be ‘bottom-up’ - when the implementation and integration details of the system are finalised, the analytical model can be finalised and evaluated.

Even if sufficient details are available to construct a timing model at a relatively early stage in the implementation process, subsequent changes will inevitably be required to some parts of the system in practice and these will impact directly on the form and/or evaluation of the model. Changes can easily be enforced due to obsolescence problems, technical implementation problems or requirements upgrades, especially in light of the long development times associated with many large-scale industrial real-time systems, such as aircraft avionic systems.

In practice, the derivation of the timing model is further complicated by the fact that a range of timing models with different characteristics, *eg.* in terms of complexity, pessimism and sensitivity, can be derived for a given system on the basis of the same implementation details. This can be seen even in the relatively simple example of a uniprocessor system

operating under fixed-priority scheduling where a range of timing models can be defined to model the same system. For example, this is demonstrated by the proposition of multiple forms of 'sufficient-but-not-necessary' analysis with different degrees of pessimism and complexity for a uniprocessor static priority-based scheduling solution - see [Audsley93b]. Clearly, given the full set of the implementation details that describe a particular system, a wide range of alternative timing models could potentially be derived. There is no guarantee, however, that an acceptable model will be available post-implementation, *ie.* that the desired set of complexity, pessimism and sensitivity characteristics will be obtained.

In engineering terms, this means that careful consideration should be given to the requirements of the analytical model itself so that it can most appropriately support the development of the system. The analytical model should then be developed to meet these requirements with acceptable constraints placed on the system implementation, rather than the implementation dictating the forms of analysis available. It is acknowledged that the ability to perform timing analysis does, in any case, place a set of basic restrictions on the final system implementation, predictability being the most fundamental requirement. It is proposed that this 'top-down' philosophy should be applied more extensively to give assurance that an appropriate form of analysis is always available on completion of the implementation stage of system development. A further benefit of this approach is that the timing model is less likely to become obsolete during the system development process, since the implementation is constrained by the analytical model, rather than the other way round. Clearly, there is an acceptable balance sought in practice between achieving an appropriate timing analysis model and constraining the implementation of the system.

3.2 Partitionable Timing Analysis for a Distributed System

This section discusses the first requirement of the timing analysis model - the ability to model the timing behaviour of parts of the system independently from other parts and, correspondingly, limit the scope of re-analysis required in the event of localised changes. This is an issue of how to partition the timing analysis model.

Note also that, at the end of chapter 2, it was proposed that the ability to partition the overall system timing analysis model is fundamental to the development of a scaleable solution for large-scale practical applications. This is due to the fact that a partitioned model allows parts of the system, such as independent application software components, to have their timing properties verified in the context of their own developed, in isolation from other parts of the system. A partitioned model is also more amenable to supporting incremental growth than a holistic model - a partitioned model can potentially allow increments to the system to be introduced with minimal impact on existing timing analysis results.

In chapter 1, it was recognised that the combined effects of functional and physical integration in a distributed real-time system result in a holistic timing analysis model. Hence, in order to devise a suitable means of partitioning the timing analysis model, *ie.*

remove its holistic property, it is first necessary to understand the precise effects of functional and physical integration on the timing model. The model can be then partitioned by placing appropriate restrictions on the manner in which functional or physical integration of the system is performed. The following partitioning options arise :

- partitioning in the functional domain - breaking the dependencies in the model that arise due to functional integration (ordering/precedence relationships);
- partitioning in the physical domain - breaking the dependencies in the model that arise due to physical integration (resource sharing relationships);
- partitioning in both functional and physical domains.

Each of these options is discussed below with reference back to the example given in chapter 1 (figure 1-2).

3.2.1 Partitioning in the Functional Domain

Functional dependencies within the system, *ie.* ordering/precedence relationships between tasks, are manifest in the timing analysis model as *jitter inheritance* effects. In the example, the completion of task A1 is required before A2 can begin its execution. Consequently, in the timing analysis model, all variation in the time of completion of task A1 will be inherited as variation in the time at which A2 becomes ready for execution. This can be stated in timing analysis terminology as, "the response jitter of A1 will be inherited as arrival jitter in A2". This is illustrated in figure 3-1 where J_{A1}^{out} denotes the response jitter of A1 and J_{A2}^{in} denotes the arrival jitter of A2. A similar dependency arises due to the precedence relationship between B1 and B2, with similar impact on the timing model.

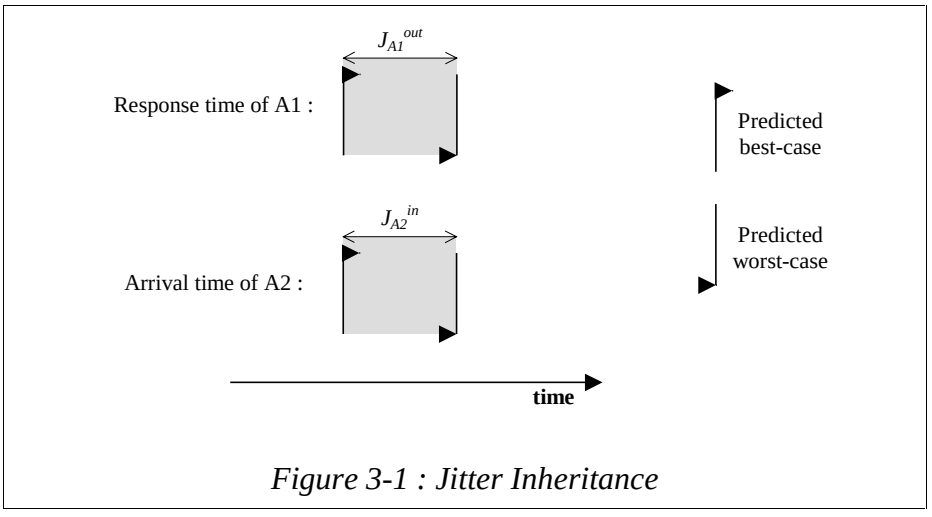


Figure 3-1 : Jitter Inheritance

To functionally partition the model is to enforce ordering/precedence constraints between tasks in a static manner. Temporal isolation is enforced between functionally-dependent components and all jitter inheritance effects are removed from the model. This leaves only the effects of physical integration to be resolved by analysis.

Static enforcement of precedence relationships can be implemented by pre-defining the arrival times of tasks that are subject to precedence constraints. To illustrate this, refer to figure 3-2 which depicts a functionally partitioned solution for the example originated in chapter 1.

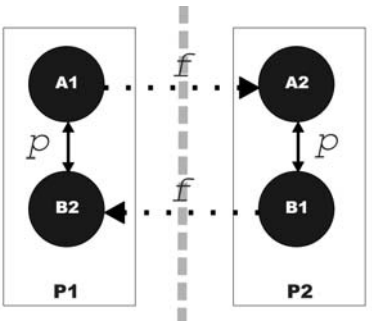


Figure 3-2 : Functionally Partitioned Model

Given the precedence constraints in the example, a functionally partitioned timing model can be obtained by statically enforcing the arrival times of tasks A2 and B2 such that these times are always later than the worst-case response times of their respective predecessors, A1 and B1. This could be achieved in one of two ways :

- by determining the worst-case response times of the tasks A1 and B1, then selecting the static arrival time values accordingly - but then the only way to calculate these response times is by performing a holistic analysis of the system;
- by fixing the arrival times of tasks A2 and B2 based purely on knowledge of the end-to-end timing requirements of the system and the task topology - the arrival times of A2 and B2 then represent hard deadlines that must be met by A1 and B1, respectively, in order for the precedence condition to be guaranteed under worst-case conditions.

The latter approach is clearly preferable since it avoids the need to perform holistic analysis at any stage. Examples of timing analysis methods that adopt this form of partitioning approach are those developed by Bate [Bate99] and Sha [Sha93], both of which refer to static priority-based solutions (as reviewed in section 2.4.3.2).

The net result of the functionally partitioned approach is that the timing behaviour of groups of tasks allocated to individual processors can be analysed independently - in the example, the group (A1, B2) can be analysed independently from (A2, B1). The same approach can be applied in the communication domain to give independent analysis of individual communication media.

Functional partitioning results in a ‘federated’ timing analysis model where resource boundaries in the physical architecture dictate the partitions in the timing model. The resulting model can only be applied after the physical integration stage of system development has been performed, *ie.* the allocation of tasks to processors (and other

resources) has been defined and the details of the scheduling solution and communication protocols have been finalised.

Figure 3-3 shows the impact of functional partitioning on the 'timeline' diagram for A1→A2, with the arrival jitter completely removed for task A2. The best- and worst-case arrival times for A2 are now equal, ignoring the fact that the worst-case value could increase later as a result of an imperfect implementation - the important property being that any arrival jitter associated with A2 is independent from the timing behaviour of its predecessor A1.

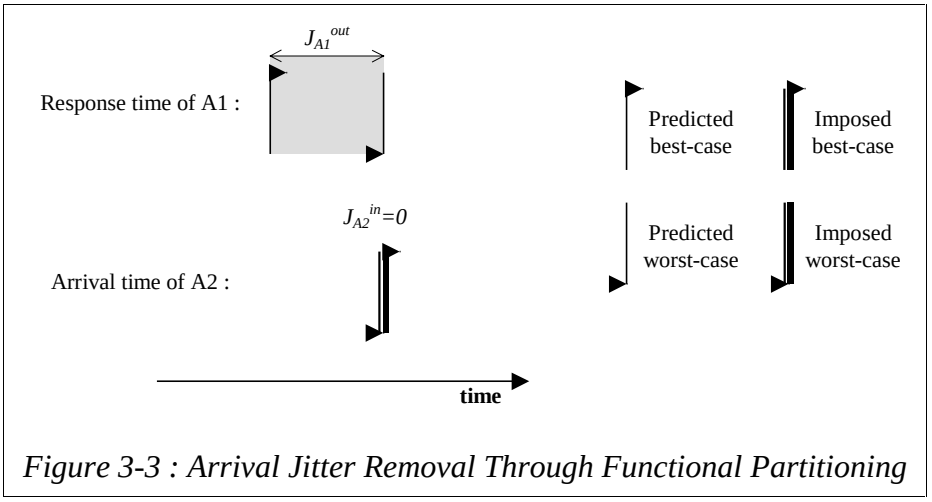


Figure 3-3 : Arrival Jitter Removal Through Functional Partitioning

3.2.2 Partitioning in the Physical Domain

Physical dependencies within the system, *ie.* resource sharing relationships between tasks, are manifest in the timing analysis model as *interference* effects. In the example from chapter 1, whilst task A1 is executing, the execution of B2 (or, more generally, all other tasks that are eligible for execution on processor P1) will be temporarily delayed by the amount of time that A1 is allowed to execute. This introduces a dependency within the timing analysis model that links the analyses of tasks A1 and B2 and, similarly, A2 and B1. The precise nature of interference effects is dependent upon the scheduling policy that governs access to each shared resource. The example illustrates the case of shared processing resources but analogous effects occur in the communication domain when communication media are shared.

To physically partition the timing model is to remove the variability in the model that is attributable to interference effects. Temporal isolation is thus provided between functionally-independent components. This leaves only the effects of functional integration to be resolved by analysis. A physically partitioned model can be produced by placing appropriate restrictions on the choice of scheduling solution for the system. Figure 3-4 illustrates a physically partitioned solution for the example system.

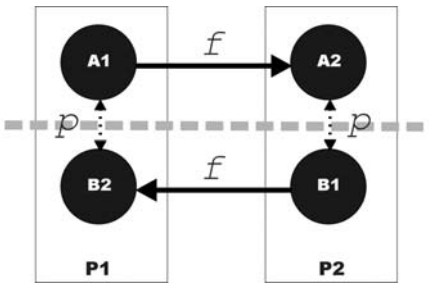


Figure 3-4 : Physically Partitioned Model

By appropriate choice of scheduling solution on processor P1, the best- and worst-case delays experienced by task A1 due to interference from B2 are calculable independently from the timing behaviour of B2. Similarly, the analysis of B2 can be performed independently from the behaviour of A1. The same applies in the case of tasks A2 and B1 on processor P2. Hence, the end-to-end behaviour of A1→A2 and B1→B2 can be analysed independently.

As can be seen from the example, the net result of physical partitioning is that the timing behaviour of functionally-dependent groups of tasks can be analysed independently. This offers some key advantages over functionally partitioned solutions. Firstly, for systems that embody a degree of safety-critical functionality, this means that there is inherent temporal isolation between safety-critical and non-safety-critical software components that share processing resources and/or communication media. This is beneficial in terms of the ability to argue, *eg.* as part of an overall system safety argument for systems subject to standards such as Defence Standard 00-55 [DS 00-55], that the sharing of resources between safety-critical and non-safety-critical applications does not compromise the integrity of the former (at least in the temporal domain). Secondly, and more generally from the perspective of the overall system engineering process, there is potential for supporting independent development and verification of groups of functionally-related components for a target computing environment that is physically integrated. Moreover, the analysis can potentially be applied before the physical integration stage has been performed, *ie.* much earlier in the development life of the system than for the functionally partitioned solution (as discussed in section 3.3). For these reasons, it is proposed that the physically partitioned solution is more in line with the requirements of real-time applications and the practical needs of the industrial system engineering process.

It can be seen from the survey in chapter 2 that a number of timing analysis models have been developed that support the physical partitioning approach to varying degrees - see section 2.3.3 for cyclic solutions, section 2.4.3.3 for static priority-based solutions and section 2.5.3.3 for dynamic priority-based solutions. Existing techniques, however, fall some way short of what is potentially achievable for analysing distributed real-time systems. In particular, the approach can be generalised to provide a consistent timing analysis model

across all shared processing and communication media in a distributed real-time system. This would not only give a consistent framework for end-to-end timing analysis but also provides the basis for a timing analysis approach that can be applied throughout the system life-cycle (as discussed in section 3.3).

3.2.3 Partitioning in both Functional and Physical Domains

A third partitioning option is now discussed. The approach is not fundamental in itself, involving the combined use of functional and physical partitioning techniques, but does represent a widely used approach in industrially-developed distributed real-time systems, the foremost academic application being the MARS time-triggered architecture [Kopetz89, Kopetz92, Kopetz94].

Since the approach applies the techniques of functional and physical partitioning in combination, the result is the removal of all variability from the timing model to produce a completely deterministic solution. This solution can then be enforced statically at run-time. Three solution strategies are apparent :

- perform the functional partitioning step first, then apply physical partitioning to the result - this strategy seems flawed since a functionally partitioned model cannot be applied until after the implementation/integration stage of system development whereas a physical partitioned model can be applied much earlier;
- perform the physical partitioning step first, then apply functional partitioning to the result - this strategy is more practical for the reason given above;
- perform both steps simultaneously - this requires an holistic analysis of the system and so defeats the object of partitioning the timing model.

The most appropriate strategy would appear to be that of performing the physical partitioning first then the functional partitioning later. Solutions arising from the combined use of functional and physical partitioning are often criticised, however, for being over-constrained [Locke92]. Since functional partitioning does not directly reflect the timing requirements of the system, the use of this technique should be justified on the basis of some demonstrable benefit in terms of system integration or analysis.

3.3 Timing Analysis through the System Life-Cycle

This section discusses the second requirement of the timing analysis model (as identified in chapter 1) - the ability to assess the timing behaviour of the system throughout the development process, based on the latest information available at each stage of development. Note that no particular form of ‘life-cycle’ is assumed here beyond the notion that requirements, design, implementation/integration and testing/verification stages are involved, where these stages may be performed with a degree of concurrency and iteration. More specifically, it is assumed that the following activities are involved in the process :

- definition of system-level timing requirements;
- decomposition of these requirements during system design and implementation;
- acquisition and integration of software and hardware components;
- verification of systems timing properties against stated requirements.

In section 3.1, it was identified that timing analysis models are normally developed in a 'bottom-up' manner, *ie.* the model is not finalised until after the implementation and integration details of the system have been decided. Hence, it is not possible to assess the timing behaviour of the system until late in the development process. Furthermore, subsequent changes to the implementation or integration details of the system, particularly changes to the scheduling or communication protocols involved, are likely to impact on the details of the timing model. The results of analysis performed at this late stage of development are, of course, essential to support final verification of the system timing requirements. Any deficiencies discovered at this late stage, however, can give rise to significant re-work, involving possible re-consideration of the artefacts produced from the integration, implementation, design or even requirements phases of development, depending on the severity of the problem. The costs associated with re-work are a major factor in the overall development cost of industrial real-time systems. For example, it is reported in [Burns94] that, for a typical DO-178B-compliant development process, the overall cost of verification effort (including re-work) is around 50% of the total system development cost - the Boeing 777 aircraft on-board computing system is cited as an example of such a project. A similar figure is quoted in [Hutchesson98] in the context of modern aircraft engine control computers.

The amount of re-work associated with the development and verification of the timing properties of the system could be reduced by making the notion of timing analysis more integral to the systems engineering process as a whole, allowing it to be applied throughout the development of the system, starting much earlier in the process. This would allow an ongoing assessment of emerging system timing properties relative to specified timing requirements and also provide progressive guidance on the selection of future design/implementation details at successive stages of development. This approach complies with the evolvable system development goal for future IMA systems in the aerospace domain.

Two fundamental issues can be identified regarding the provision of timing analysis through the system life-cycle :

- how to perform timing analysis with a lack of system integration and implementation details during the earlier stages of development;
- how to deal with a continually evolving definition of the system, with different parts of the system evolving at different rates.

3.3.1 Timing Analysis without System Integration and Implementation Details

In the earlier stages of development, the timing-related information for all parts of the system will be scarce. Whilst the system level timing requirements may be reasonably well understood and decomposed to varying extent during design stages, the ability to perform timing analysis relies on the notion of *resources* - some media through which the functions of the system can be performed. Processing resource details are used as a basis for calculating worst-case execution times. Similarly, details of the communication media (and associated protocols) are used to characterise worst-case communication delays.

In order to perform timing analysis prior to the system implementation stage, it is therefore necessary to work with an implementation-independent, or *abstract*, model of system resources. The model should provide a means to perform timing analysis in the earlier stages of system development using estimations or assumptions about system integration and implementation details that will not become concrete until a later date. The abstract model must be defined to provide a sufficient basis for performing timing analysis but without over-constraining the final implementation of the system. When implementation details are eventually finalised, the results of timing analysis performed via the abstract model must then be verified. This gives rise to a two-stage approach to timing analysis :

- *abstract timing analysis* - this is performed during the definition and decomposition stages of development on the basis of the abstract resource model; the net result is a set of best- and worst-case *guarantees* regarding the timing behaviour of the system that are subject to a set of *obligations* being met by the final implementation;
- *target-specific timing analysis* - this is performed during the system implementation and integration stages of system development, the aim being to demonstrate that the set of obligations imposed during the abstract timing analysis phase have actually been met.

By appropriate construction of the timing model, the guarantees generated during the abstract timing analysis phase can be inferred automatically at the target-specific analysis stage, *ie.* there should be no need to reconstruct the results of the system-wide abstract analysis after the target-specific details have been finalised and verified. This is not the manner in which traditional forms of timing analysis proceed, which is invariably to reflect the target system implementation/integration details directly in the timing model.

The ability to perform timing analysis in this way offers the practical benefit of leaving hardware procurement to a relatively late stage in development. This can help to mitigate the problem of hardware obsolescence during system development - a significant problem in the industrial development of large-scale real-time systems [DS 00-71]. The same approach supports hardware upgrades in service without re-analysis of system-wide timing behaviour. New (functionally equivalent) hardware components can be introduced to replace existing components that may be faulty or obsolete, by merely inheriting the same set of timing obligations from the abstract timing analysis model.

Significantly, the opportunity can be taken during definition of the abstract timing model to define a consistent (unified) model for system-wide analysis, *ie.* for end-to-end analysis across all system resources. This improves composability of the timing analysis model by supporting consistent modelling of different resource types (processing and communication media) in the system. As suggested earlier, it is normally the case that different types of system resources and resource access policies (scheduling and communication protocols) are modelled in different ways.

3.3.2 Timing Analysis for an Evolving System Definition

The development of a timing analysis model for application through the system life-cycle must inherently address the problem of dealing with an evolving definition of the system. Throughout the development process, different parts of the system will mature at different rates, leading to an inconsistent level of detail regarding the timing properties of the system. The timing model must not only be able to represent the various parts of the system at different levels of detail but also still be evaluated to determine system timing behaviour based on such information.

This can be achieved by structuring the timing model as a hierarchy, where successive levels of the hierarchy capture the results of successive stages of system evolution. At any given stage of development, the hierarchy should capture all known ordering/precedence relationships between the entities that describe the system at that stage and all known resource requirements. Ideally, the nodes of the hierarchy should be of a form that is consistent throughout the development of the system in order to avoid any problems associated with transforming the timing properties of the system between different representations associated with different stages of development. The precise form of these nodes should reflect other objectives of the timing analysis model. In particular, the proposal to partition the timing model in the physical domain, as described in section 3.2, suggests that the nodes of the hierarchy should be characterised by indivisible resource usage requirements (and the corresponding response times) associated with computational and communication elements within the system.

In this way, functionally unrelated parts of the system can potentially be modelled, analysed and modified independently and therefore allowed to evolve at different rates throughout the development process. Similarly, in the later stages of system development, the analysis should be applicable to partially integrated subsets of the system in order that verification of system timing properties can occur in step with the integration process itself, rather than waiting for the complete system to be finalised before the timing behaviour of any one part of it can be verified.

The two-stage approach described in section 3.3.1 for the purposes of performing timing analysis without implementation details can be generalised to support evolvable timing analysis through the life-cycle of the system. At successive stages of development, the

guarantees generated from performing abstract timing analysis represent timing obligations on all subsequent stages of development. This concept is illustrated in figure 3-5 in the context of a V-model development process.

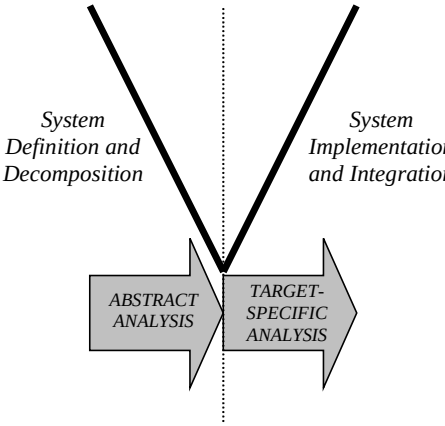


Figure 3-5 : Timing Analysis through the System Life-Cycle

By such means, a consistent approach to analysis can be provided throughout the life-cycle of the system. This is preferable to adopting a series of separate method/tool-specific techniques at different stages of system development where there is no guarantee that the resulting combination of techniques will deliver the required results.

3.3.3 Related Work

Given the general approach proposed above for performing timing analysis through the life-cycle, a brief mention of some related work is warranted. This work is described here rather than in the main literature review since no direct means for performing timing analysis is provided. The work is interesting, however, from the perspective of supporting a target-independent representation of a real-time system.

Stoyenko *et al* have developed a 'resource-oriented' system description language, *CaRT-Spec*, that supports an evolving definition of a distributed real-time system [Stoyenko96]. The system is represented as sets of concurrent tasks with associated timing requirements, with each task described by a single entry, single exit acyclic graph of *actions*. Actions capture the resource consumption characteristics of each task and can be structured hierarchically to reflect ongoing refinement during the development of the system. The system specification can be executed symbolically, accounting for recognised resource usage restrictions, such as constraints on pre-emptability and synchronous usage. Unlike the approach outlined in this chapter (and developed in the rest of the thesis) CART-Spec requires the application developer to model resource usage explicitly, *ie.* effectively defining an allocation of tasks to resources, before any assessment can be performed. Furthermore, this assessment does not constitute a form of feasibility analysis. Also, unlike the approach proposed here, there is no form of temporal partitioning between tasks.

Burns and Lister have described the TARDIS architectural framework for top-down development of timely and reliable distributed systems [Burns91a]. TARDIS distinguishes between the logical and physical descriptions of the system architecture with implementation-independent timing obligations being imposed from the former to the latter. The logical architecture definition embodies both functional and non-functional requirements of the system within an object-oriented framework. Timing requirements are obligated to *active* objects (threads) and worst-case delays assigned to *passive* objects (such as shared data). The physical architecture embodies all system hardware and software components and the associated constraints on their operation. Timing obligations are mapped from the logical architecture into the domain of the physical architecture, within which scheduler objects can be included to implement specific scheduling policies. The final choice of scheduling policy is left to the application developer - no specific scheduling or timing analysis approach is suggested for assessment of feasibility. The main similarity between TARDIS and the approach proposed here is in the clear distinction between the logical and physical levels of the architecture with implementation-independent timing obligations being imposed from the former to the latter. A key difference is identifiable, however, in that TARDIS does not aim to provide any form of timing analysis, leaving this to the application developer.

3.4 Summary

For distributed real-time systems, the development of the timing analysis model should be driven by the technical and engineering requirements of the system rather than by any preconceived target hardware implementation or particular form of run-time scheduling or communication policy. To this end, two key requirements for such a model have been identified :

- the ability to model the timing behaviour of parts of the system independently from other parts and, correspondingly, limit the scope of re-analysis required in the event of localised changes - *ie.* the model should be partitionable;
- the ability to assess the timing behaviour of the system throughout the development process, based on the latest information available at each stage of development.

Two fundamental techniques have been identified for partitioning the timing model, *ie.* avoiding holistic analysis - functional and physical partitioning. It has been proposed that physical partitioning provides the most appropriate support for practical real-time systems applications; in particular :

- the structure of the timing model reflects the application structure rather than that of the hardware architecture and so there is the possibility of performing timing analysis much earlier in the development process;
- timing analysis can be performed independently for independent applications.

The need to perform timing analysis throughout the system life-cycle, rather than merely at the end of the development phase, has been identified as a means of providing practical engineering and cost benefits. Timeliness is an intrinsic, non-functional property of real-time systems and cannot be added at the end of the development phase. The penalty for late consideration of system timing properties can be significant in terms of re-work and its associated costs. The timing behaviour of the system should be modeled at each stage of development and verified against the timing requirements of the system within the limitations of the information available at each particular stage. The same basic timing model should be applicable throughout the system life-cycle, providing a consistent approach to analysis rather than adopting a series of separate method/tool-specific techniques, since there is no guarantee that the resulting combination of techniques will deliver the required results.

CHAPTER 4. Reservation-Based Timing Analysis

This chapter describes the proposed timing analysis approach in detail. The approach is referred to as reservation-based timing analysis (RBA) for reasons that will become clearer during the description. The approach is described in the following stages :

- representation and refinement of system timing properties;
- end-to-end (system level) timing analysis;
- localised (component level) timing analysis.

The first two stages of the description cover target-independent aspects of the model, *ie.* the abstract timing analysis phase referred to in chapter 3. The third stage supports both target-independent and target-dependent analysis equally - this provides the basis for a smooth transition between the abstract timing analysis model and the target-dependent model at the appropriate time during system development.

4.1 Representation and Refinement of System Timing Properties

The end-to-end timing characteristics of the system are captured, developed and analysed in terms of *transactions*. A transaction captures the temporal relationship between :

- a set of *input events* from the external environment of the system or some other transaction(s);
- a set of *output events* to the external environment of the system or some other transaction(s).

In the ideal sense, transaction input and output events correspond to the arrival or dispatch of control signals and/or information from or to the external interfaces of the system. In order to better support the engineering of large-scale systems, however, such events can relate to other transactions within the system as well as the external environment of the system. This allows particular end-to-end requirements of the system to be modelled as multiple transactions where practical engineering constraints dictate, such as the need to allocate responsibilities across industrial partnerships and sub-contractors. This practical concern is supported, however, without compromising the system timing model due to the uniform nature adopted for expressing transaction structural and timing properties, as explained in sections 4.1.1 and 4.1.2, respectively.

4.1.1 Transaction Model Structure

The structure of the transaction model is hierarchical, reflecting the need to support an evolving definition of the system during development (as proposed in the rationale in chapter 3). The body of a transaction at any stage in its development is expressed in the form of an acyclic, directed, nested graph whose leaf nodes capture the concurrent processing and communication elements of the transaction, termed *activities*. Non-leaf

nodes are referred to as *nested transactions*. The edges (or arcs) of the graph capture the precedence and nesting relationships within the transaction.

Precedence relationships describe the required order of execution of activities and nested transactions. Nesting relationships capture specification-implementation relationships that arise during the evolution of the transaction model. In other words, the structural and timing properties attributable to each nested transaction form a specification that is implemented by the combined behaviour of all underlying nested transactions and activities.

Consider the transaction λ depicted in figure 4-1. This illustrates the evolving definition of a transaction from an initial, system level description (or level 0 model) through two stages of refinement (level 1 and 2 models). As a result of the first stage of refinement, the level 0 nested transaction λ_1 is implemented via three level 1 nested transactions. As a result of further refinement, each of the level 1 transactions is implemented as a set of level 2 activities.

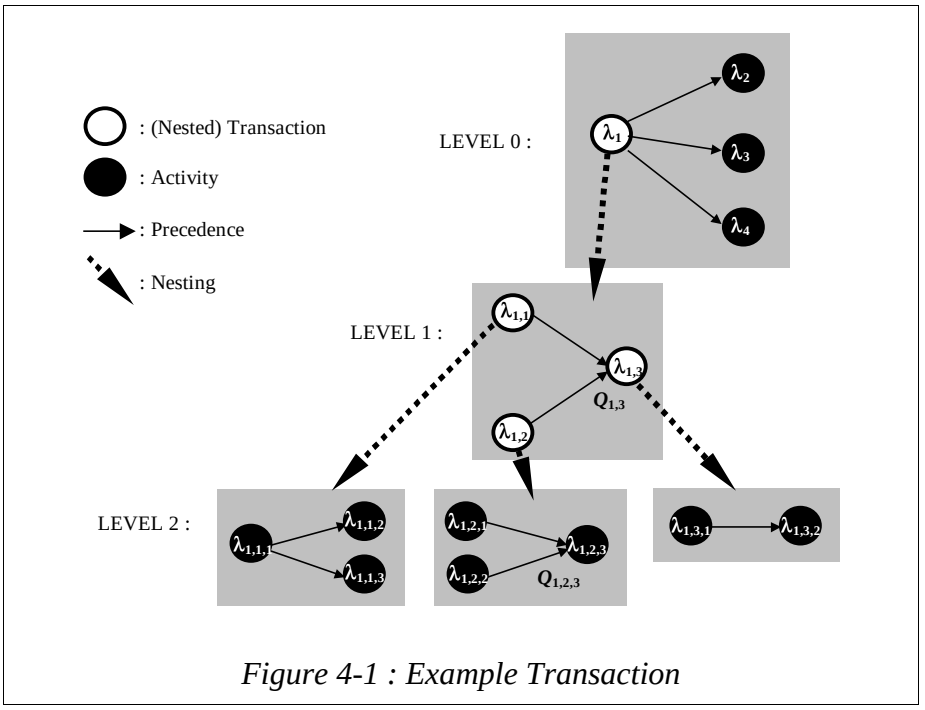


Figure 4-1 : Example Transaction

More generally, let $\lambda_{i,..,k}$ denote any arbitrarily nested transaction or activity within transaction λ. In order to capture the precise details of nesting and precedence relationships within λ, a *guard* function $Q_{i,..,k}$ is associated with each $\lambda_{i,..,k}$ describing the conditions that govern its release for execution. In many cases, $Q_{i,..,k}$ is implicit given the form of relationship involved. For example, the release of $\lambda_{i,..,k}$ dependent on completion of a single predecessor does not require $Q_{i,..,k}$ to be defined explicitly since the constraint is wholly described by topology of the directed graph. In other cases, however, $Q_{i,..,k}$ is not sufficiently defined from the graph topology alone and must be expressed explicitly. This is illustrated in the above example by association of the function $Q_{1,3}$ with the nested transaction $\lambda_{1,3}$. $Q_{1,3}$ must be defined explicitly since $\lambda_{1,3}$ could be released for execution upon completion of the

two predecessors, $\lambda_{1,1}$ and $\lambda_{1,2}$, on completion of either one of those predecessors, or even dependent on the value(s) of some data item(s) produced by one or both predecessors.

In practice, the precise form of $Q_{i,..,k}$ will be dependent on application design and implementation details and consequently its value will fluctuate at run-time dependent on actual variations in timing and/or data. That said, a basic categorisation of precedence and nesting relationships, from which more complex transaction topologies can be constructed, can be given as follows. Consider first the case of precedence relationships :

- *one-to-many* - where the completion of a single activity or nested transaction triggers the arrival of one or more successors (as illustrated in figure 4-1 within the level 0 model);
- *many-to-one* - where the completion of one or more concurrent activities or nested transactions is required to trigger the arrival of a single successor (as illustrated in figure 4-1 within the level 1 model).

The simple case of *one-to-one* precedence, where the completion of a single activity or nested transaction triggers the arrival of a single successor, is a special case of each of these classes and actually represents the intersection between the two.

An analogous categorisation can be defined for nesting relationships as follows, observing that these relationships involve successive (referred to as parent and child) levels of the transaction hierarchy :

- *one-to-many* - where the arrival of a single nested transaction at the parent level triggers the arrival of one or more concurrent activities at the child level (as illustrated in figure 4-1 in the transition from the level 1 to the 2 model at $\lambda_{1,2}$) or, traversing the hierarchy in the opposite direction, where the completion of a single activity at the child level triggers the completion of one or more concurrent nested transactions at the parent level;
- *many-to-one* - where the arrival of one or more concurrent nested transaction at the parent level triggers the arrival of a single activity at the child level or, traversing the hierarchy in the opposite direction, where the completion of one or more concurrent activities at the child level triggers the completion of a single nested transaction at the parent level (as illustrated in figure 4-1 in the reverse transition from the level 2 to the 1 model at $\lambda_{1,1}$).

For ease of future reference, nesting relationships that are directed from parent to child will be referred to as *descending* and those that are directed from child to parent will be referred to as *ascending*.

The purpose of the nesting relationships is to reflect the ongoing evolution of the system during development and later in service. For convenience, transactions can also be visualised at any stage of refinement in an equivalent ‘flat’ form, *ie.* with nesting relationships transformed into precedence relationships. This is achieved by performing a depth-first traversal of the nested transaction graph and successively replacing each nested

transaction with its lower level sub-graph, the sub-graph inheriting all higher level nesting and precedence relationships from the nested transaction that it replaces, as follows :

- all (descending) nesting and precedence relationships that represent immediate preconditions for the execution of the nested transaction being resolved are inherited by every input activity in the sub-graph;
- all (ascending) nesting and precedence relationships that represent immediate postconditions following the execution of the nested transaction being resolved are inherited by every output activity in the sub-graph.

To briefly illustrate this, consider the example transaction depicted in figure 4-1. Figure 4-2a illustrates the same transaction with the nesting relationship that stems from $\lambda_{1,1}$ resolved.

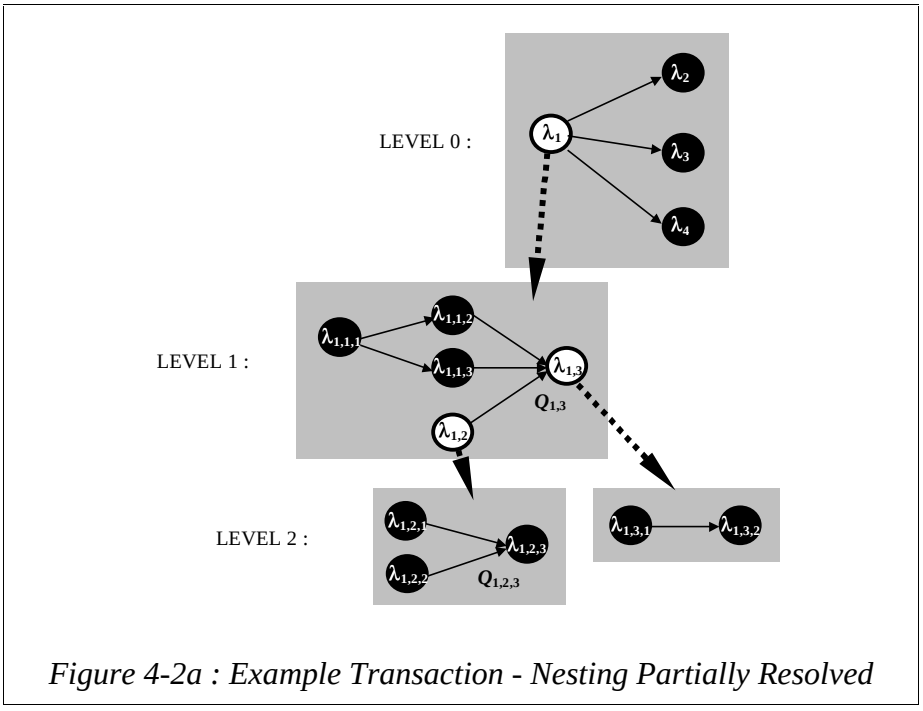


Figure 4-2a : Example Transaction - Nesting Partially Resolved

Applying this process to resolve all nesting relationships for the example transaction gives the resultant flat topology illustrated in figure 4-2b.

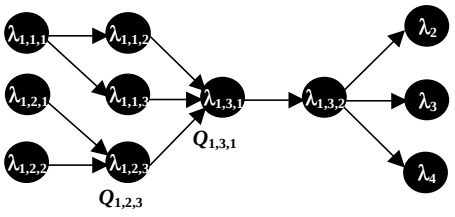


Figure 4-2b : Example Transaction - Nesting Fully Resolved

Note the impact of resolving nesting relationships on any guard functions involved. For example, suppose that $Q_{1,3}$ in the original nested transaction topology shown in figure 4-1 is defined such that $\lambda_{1,3}$ is released only upon completion of both predecessors, $\lambda_{1,1}$ and $\lambda_{1,2}$.

When the nesting relationship that stems from $\lambda_{1,1}$ is resolved, $Q_{1,3}$ must be restated to account for the fact that the sub-graph that implements $\lambda_{1,1}$ terminates with two concurrent activities. Hence, $Q_{1,3}$ is redefined such that $\lambda_{1,3}$ is released only upon completion of all predecessors, $\lambda_{1,1,2}$, $\lambda_{1,1,3}$ and $\lambda_{1,2}$. When the nesting relationship that stems from $\lambda_{1,3}$ itself is resolved, $Q_{1,3}$ must be relabelled $Q_{1,3,1}$ to comply with the activity labelling associated with the sub-graph. This is easy to see from the example notation given below for expressing guard functions.

Let $e_{i,..,k}$ denote the completion of activity $\lambda_{i,..,k}$. The original statement of $Q_{1,3}$ would then be expressed as follows :

$$Q_{1,3} = e_{1,1} \wedge e_{1,2}$$

Resolving $\lambda_{1,1}$ gives :

$$Q_{1,3} = (e_{1,1,2} \wedge e_{1,1,3}) \wedge e_{1,2}$$

Resolving $\lambda_{1,2}$ gives :

$$Q_{1,3} = (e_{1,1,2} \wedge e_{1,1,3}) \wedge e_{1,2,3}$$

Resolving $\lambda_{1,3}$ gives :

$$Q_{1,3,1} = (e_{1,1,2} \wedge e_{1,1,3}) \wedge e_{1,2,3}$$

This corresponds to the final form of the guard function after all nesting has been removed, as illustrated in figure 4-2b.

4.1.2 Transaction Timing Properties

In order for the timing model to be evolvable, the actual timing parameters that describe the two main objects of the transaction model, *ie.* the nested transaction and the activity, are defined such that these objects are interchangeable. In other words, any nested transaction can be implemented in terms of activities (and further nested transactions) and, *vice versa*, an activity can be replaced by a nested transaction and become the subject of further evolution. In this way, the same timing analysis approach can be applied to predict the behaviour of a transaction throughout its evolution.

The only distinction between activities and nested transactions is that activities, since these represent leaf nodes of the transaction graph, must define their resource requirements directly, whereas nested transactions inherit their resource characteristics from the activities they embody. Otherwise, the parameters via which timing behaviour is represented and observed are the same for a single activity, a group of activities, a nested transaction and a level 0 transaction.

For any arbitrarily nested transaction or activity, $\lambda_{i,..,k}$, with level 0 ‘parent’ λ_i, the timing properties are represented as follows :

- *input jitter*, $J^{in}_{i,..,k}$ - the maximum width of the time window that spans the arrival of all input events associated with $\lambda_{i,..,k}$;
- *output jitter*, $J^{out}_{i,..,k}$ - the maximum width of the time window that spans the deliverance of all output events associated with $\lambda_{i,..,k}$;
- *minimum I/O separation*, $d_{i,..,k}$ - the minimum separation in time (delay) between the input and output event windows of $\lambda_{i,..,k}$;
- *minimum inter-arrival time*, $a_{i,..,k}$ - the minimum separation in time between the input event windows associated with successive instances of $\lambda_{i,..,k}$.

The relationships between these parameters are depicted in figure 4-3.

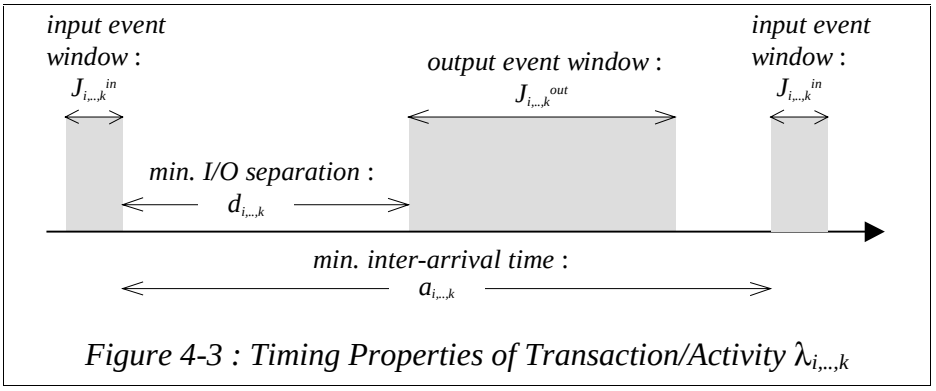


Figure 4-3 : Timing Properties of Transaction/Activity $\lambda_{i,..,k}$

In figure 4-3 the inter-arrival time of $\lambda_{i,..,k}$ is shown to be greater than the latest completion time of any $\lambda_{i,..,k}$ output event, corresponding to the case where any one instance of $\lambda_{i,..,k}$ will always complete before the next instance arrives. This is merely to keep the illustration simple and is not actually a constraint on the model. For example, a multimedia application involving video data streaming may require a minimum frame update time that is significantly less than its worst-case latency - multiple frames may be at various stages of processing at any one time, so long as they are processed in order at any given stage in the computational 'pipeline'. The RBA end-to-end timing analysis model supports this type of behaviour, though there are other constraints imposed on model parameters as follows, for reasons given :

- $d_{i,..,k} > -J^{in}_{i,..,k}$ - a necessary condition arising from the basic restriction that, for any given instance of $\lambda_{i,..,k}$, the output event window cannot begin before the input event window begins;
- $a_{i,..,k} > 0$ - a constraint imposed to ensure that the input event windows associated with successive instances of $\lambda_{i,..,k}$ are totally ordered.

The same set of timing parameters is sufficient to model periodic, sporadic and aperiodic behaviour as follows :

- if $a_{i,..,k}$ is unbounded then $\lambda_{i,..,k}$ is aperiodic;
- if $a_{i,..,k}$ is bounded but $J^{in}_{i,..,k}$ is unbounded then $\lambda_{i,..,k}$ is sporadic;
- if $a_{i,..,k}$ is bounded and $J^{in}_{i,..,k}$ is bounded then $\lambda_{i,..,k}$ is periodic with period equal to $J^{in}_{i,..,k} + a_{i,..,k}$.

Figure 4-4 illustrates this classification.

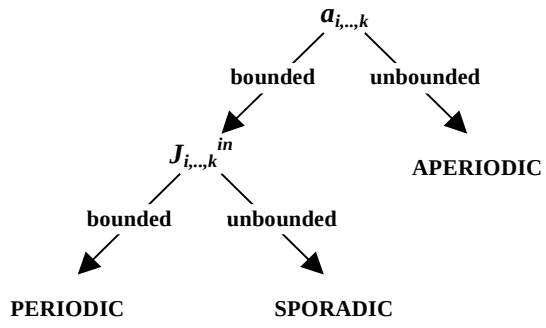


Figure 4-4 : Periodics, Aperiodics and Sporadics

In the periodic case, the input event window for every instance of $\lambda_{i,..,k}$ can be fixed *a priori* relative to that of the first instance of $\lambda_{i,..,k}$. In the sporadic case, however, since the input jitter is unbounded, the input event window for any given instance of $\lambda_{i,..,k}$ is completely undefined until the actual run-time arrival of the previous instance and, even then, only the start time of the window can be determined. Hence, offline timing analysis for sporadics is only achievable by defining an artificial upper bound on the input jitter parameter, otherwise analysis must be performed online when the actual arrival time is known - the input jitter being the difference between the specified minimum inter-arrival time and the actual arrival time.

4.2 End-to-End Timing Analysis

The overall timing behaviour of the system at a given stage of evolution can be assessed by analysing the current set of transaction graphs. This section describes an approach for modelling the end-to-end timing behaviour of each transaction independently from other transactions in the system and from the eventual target hardware implementation. Following the description of the analytical model, an example is given to illustrate its use.

4.2.1 Analytical Framework

The starting point for determining the end-to-end timing behaviour of a transaction is to recognise the relationship between the timing parameters associated with each nested transaction (as stated above) and the overall delays accrued at that level of nesting. Let $r_{i,..,k}$ and $R_{i,..,k}$ denote the minimum and maximum accrued delays (the best- and worst-case

response times) associated with any nested transaction or activity $\lambda_{i,..,k}$. The following relationships are observed :

$$d_{i,..,k} = r_{i,..,k} - J^{in}_{i,..,k} \qquad (4\text{-}1)$$

$$J^{out}_{i,..,k} = J^{in}_{i,..,k} + (R_{i,..,k} - r_{i,..,k}) \qquad (4\text{-}2)$$

The above relationships are clarified by the illustration in figure 4-5.

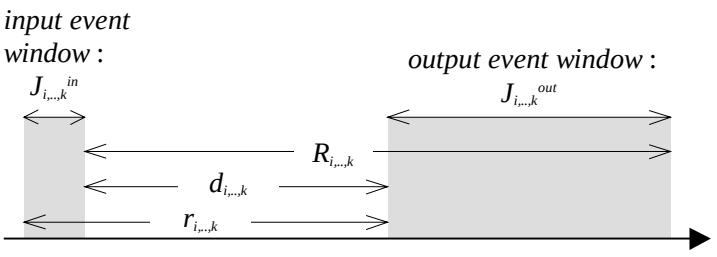


Figure 4-5 : Delay Relationships for Transaction/Activity $\lambda_{i,..,k}$

When $\lambda_{i,..,k}$ represents a single activity at the deepest level of nesting in the transaction graph, the parameters $r_{i,..,k}$ and $R_{i,..,k}$ represent localised delay values and must be assigned or evaluated directly (this is the subject of sections 4.2.3 and 4.3). For nested transactions, however, it is not possible to evaluate $r_{i,..,k}$ and $R_{i,..,k}$ directly and, instead, it is necessary to analyse the sub-graph that implements $\lambda_{i,..,k}$. Since any given sub-graph can contain further levels of nesting, the analysis must recursively descend the transaction graph, accounting at each stage for the impact of nesting relationships, precedence relationships and localised delays on the overall accrued delay values. The same approach can be taken to determining accrued delay variation, *ie.* jitter.

Ultimately, all accrued delay and jitter values relate back to the input event window for the level 0 transaction, λ. That said, the end-to-end timing model is constructed such that the delays and jitter accrued across any nested transaction or activity can be calculated relative to those inherited at its time of arrival, *ie.* the relative impact of $\lambda_{i,..,k}$ can be observed. In the transaction depicted in figure 4-1, for example, the activities $\lambda_{1,2,1}$ and $\lambda_{1,2,2}$ will each inherit accrued delay and jitter values on their arrival via a nesting relationship from their parent transaction $\lambda_{1,2}$. In the general case, let $d^{in}_{i,..,k}$ denote the accrued delay inherited by $\lambda_{i,..,k}$ upon its arrival and, in turn, let $d^{out}_{i,..,k}$ denote the accrued delay that $\lambda_{i,..,k}$ exports upon its completion. These values are related as follows :

$$d^{out}_{i,..,k} = d^{in}_{i,..,k} + r_{i,..,k} \qquad (4\text{-}3)$$

This relationship is illustrated in figure 4-6. Notice in the diagram that the term $d^{in}_{i,..,k}$ actually represents the separation in time (delay) between two input windows, rather than one input and one output window. This is consistent with the use of *d* to denote minimum I/O separation since the input event window of activity $\lambda_{i,..,k}$ (*ie.* any activity with

predecessors) is equivalent to the output window that describes the combined output jitter of it predecessors.

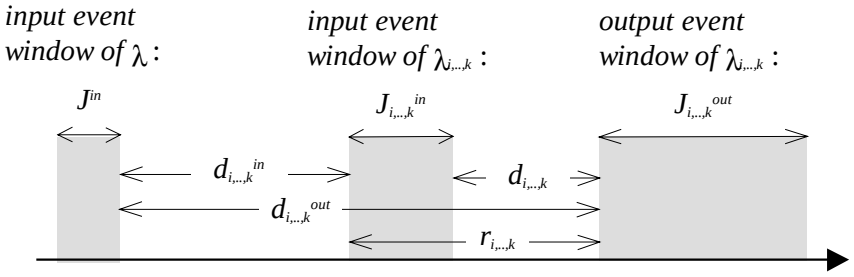


Figure 4-6 : Accrued Delay Relationships for Transaction/Activity $\lambda_{i,..,k}$

Equation *(4-3)* provides a means by which minimum localised delay values calculated at the activity level can be consolidated into the end-to-end delay calculation. The consolidation of maximum local delay values is adequately taken care of by the output jitter calculation given in equation *(4-2)*.

The manner in which accrued delays and jitter are inherited (and exported) for any particular $\lambda_{i,..,k}$ depends on the form of precedence or nesting relationships involved. In the previous section, a number of fundamental forms of such relationships were identified and these are now considered in turn.

Delay and jitter accrual across one-to-many relationships :

When nesting and precedence relationships are *one-to-many*, accrued delays and jitter are directly inherited (unchanged) across the relationship in the direction of the control flow :

- *one-to-many precedence relationship* - when $\lambda_{i,..,k}$ is one of many successors to $\lambda_{i,..,j}$:

$$d^{in}_{i,..,k} = d^{out}_{i,..,j} \qquad (4\text{-}4)$$

$$J^{in}_{i,..,k} = J^{out}_{i,..,j} \qquad (4\text{-}5)$$

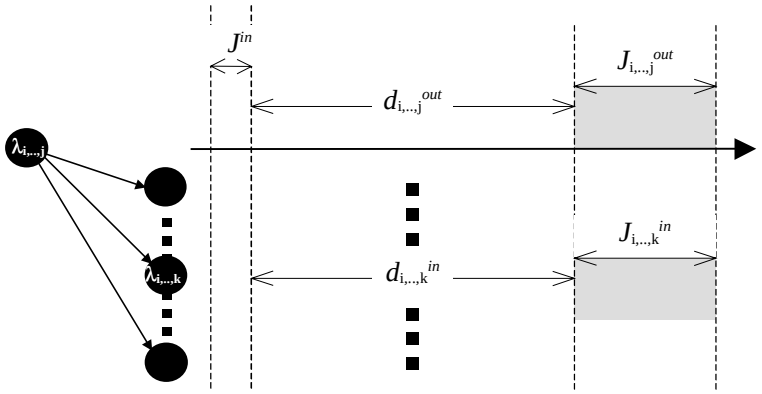


Figure 4-7 : Delay Inheritance - One-to-Many Precedence Relationship

- *one-to-many nesting relationship (descending)* - when $\lambda_{i,..,k}$ is one of many child activities whose arrival is triggered by that of the parent transaction $\lambda_{i,..,j}$:

$$d_{i,..,k}^{in} = d_{i,..,j}^{in} \qquad (4\text{-}6)$$

$$J_{i,..,k}^{in} = J_{i,..,j}^{in} \qquad (4\text{-}7)$$

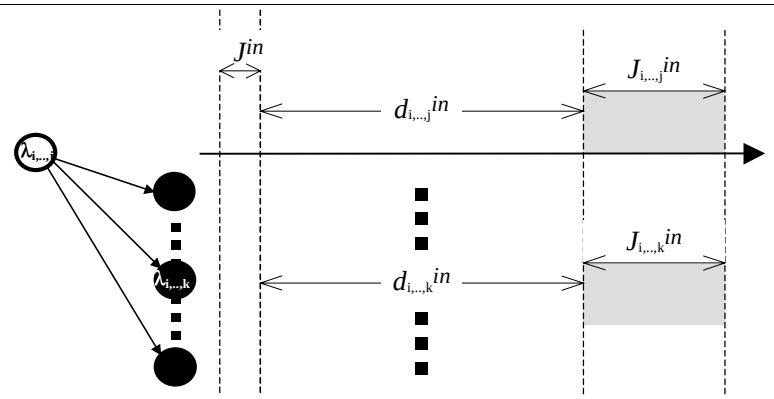


Figure 4-8 : Delay Inheritance - One-to-Many Nesting Relationship (Descending)

- *one-to-many nesting relationship (ascending)* - when $\lambda_{i,..,k}$ is one of many parent transactions whose completion is triggered by that of the child activity $\lambda_{i,..,j}$:

$$d_{i,..,k}^{out} = d_{i,..,j}^{out} \qquad (4\text{-}8)$$

$$J_{i,..,k}^{out} = J_{i,..,j}^{out} \qquad (4\text{-}9)$$

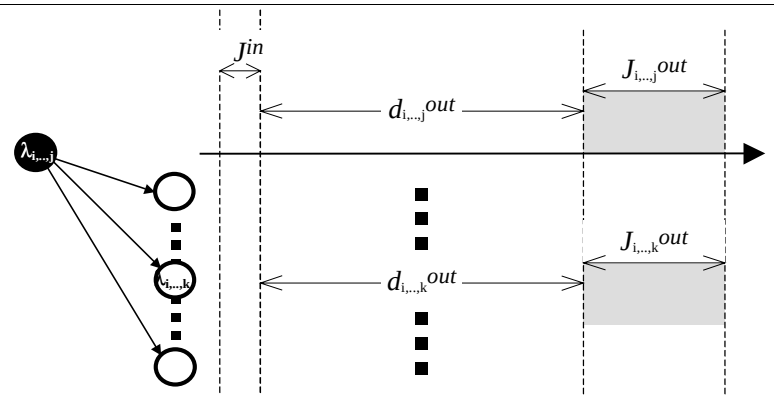


Figure 4-9 : Delay Inheritance - One-to-Many Nesting Relationship (Ascending)

Delay and jitter accrual across many-to-one relationships :

When nesting and precedence relationships are *many-to-one*, delay and jitter inheritance is less straight forward, depending ultimately on the form of guard function $Q_{i,..,k}$ that resides over the release of $\lambda_{i,..,k}$. Without knowledge of $Q_{i,..,k}$, the exact values for the inherited delay and jitter parameters cannot be determined, but the smallest 'safe' range of values can be stated as follows :

- *many-to-one precedence relationship* - when $\lambda_{i,..,j}$ is one of many predecessors to $\lambda_{i,..,k}$:

$$\min_j(d^{out}_{i,..,j}) \le d^{in}_{i,..,k} \le \max_j(d^{out}_{i,..,j}) \qquad (4\text{-}10)$$

$$\min_j(d^{out}_{i,..,j} + J^{out}_{i,..,j}) - d^{in}_{i,..,k} \le J^{in}_{i,..,k} \le \max_j(d^{out}_{i,..,j} + J^{out}_{i,..,j}) - d^{in}_{i,..,k} \qquad (4\text{-}11)$$

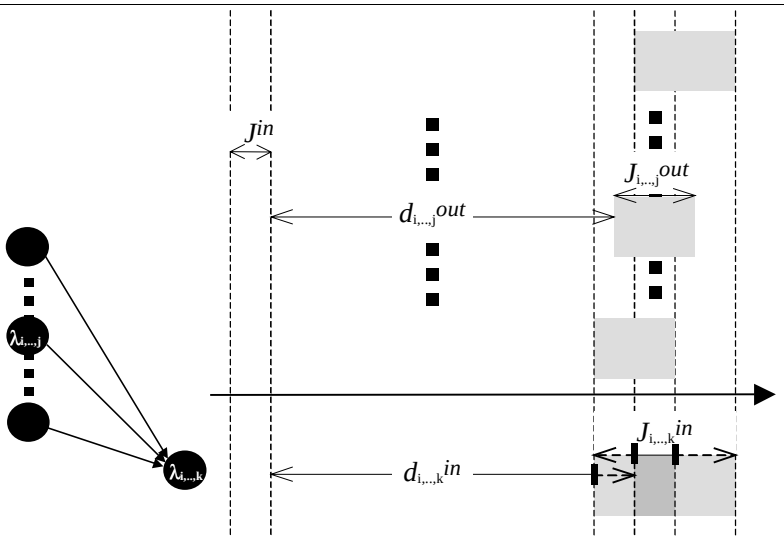


Figure 4-10 : Delay Inheritance - Many-to-One Precedence Relationship

- *many-to-one nesting relationship (descending)* - when $\lambda_{i,..,j}$ is one of many parent transactions whose arrival is required to trigger that of the child activity $\lambda_{i,..,k}$:

$$\min_j(d^{in}_{i,..,j}) \le d^{in}_{i,..,k} \le \max_j(d^{in}_{i,..,j}) \qquad (4\text{-}12)$$

$$\min_j(d^{in}_{i,..,j} + J^{in}_{i,..,j}) - d^{in}_{i,..,k} \le J^{in}_{i,..,k} \le \max_j(d^{in}_{i,..,j} + J^{in}_{i,..,j}) - d^{in}_{i,..,k} \qquad (4\text{-}13)$$

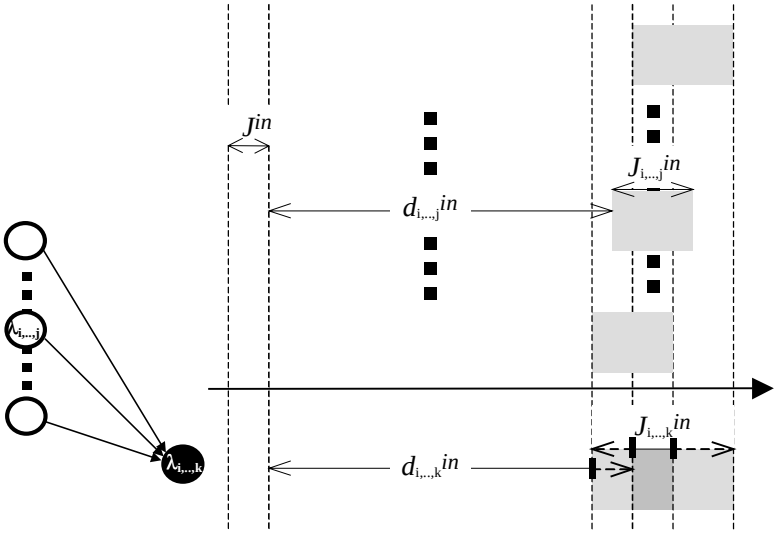


Figure 4-11 : Delay Inheritance - Many-to-One Nesting Relationship (Descending)

- *many-to-one nesting relationship (ascending)* - when $\lambda_{i,..,j}$ is one of many child activities whose completion is required to trigger that of the parent transaction $\lambda_{i,..,k}$:

$$\min_j(d^{out}_{i,..,j}) \le d^{out}_{i,..,k} \le \max_j(d^{out}_{i,..,j}) \qquad (4\text{-}14)$$

$$\min_j(d^{out}_{i,..,j} + J^{out}_{i,..,j}) - d^{out}_{i,..,k} \le J^{out}_{i,..,k} \le \max_j(d^{out}_{i,..,j} + J^{out}_{i,..,j}) - d^{out}_{i,..,k} \qquad (4\text{-}15)$$

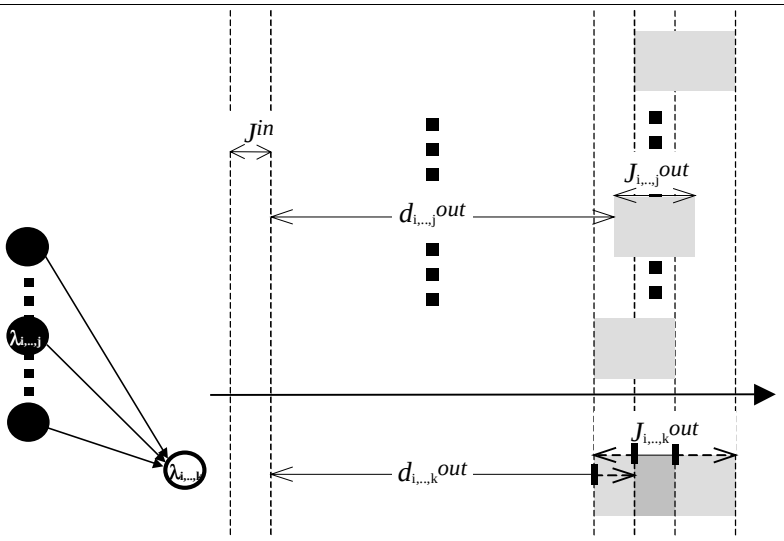


Figure 4-12 : Delay Inheritance - Many-to-One Nesting Relationship (Ascending)

Further explanation of these bounds can be given as follows. For the case of many-to-one precedence relationships (analogous statements can be made for each of the two cases of many-to-one nesting relationships) :

- given that the release of $\lambda_{i,..,k}$ will always require the completion of at least one predecessor activity $\lambda_{i,..,j}$, the stated lower bounds correspond to the case where $Q_{i,..,k}$ is defined such that $\lambda_{i,..,k}$ is released upon completion of *any one* $\lambda_{i,..,j}$;
- given that the release of $\lambda_{i,..,k}$ can potentially require the completion of all predecessor activities $\lambda_{i,..,j}$, the stated upper bounds correspond to the case where $Q_{i,..,k}$ is defined such that $\lambda_{i,..,k}$ is released upon completion of *all* $\lambda_{i,..,j}$.

In practice, the form of $Q_{i,..,k}$ will be defined (and refined) in line with the development of the associated transaction and this information can be used to reduce the pessimism of the accrued delay and jitter bounds compared to those determined by equations *(4-10)* to *(4-15)*. This can be illustrated by considering the two extreme cases that are used as the basis for deriving those equations. Note that the associated diagrams are presented in Appendix A1.

In the first case, where $Q_{i,..,k}$ is defined such that $\lambda_{i,..,k}$ is released upon completion of *any one* $\lambda_{i,..,j}$, equations *(4-10)* to *(4-15)* can be reduced as follows :

- *many-to-one precedence relationship* - when $\lambda_{i,..,j}$ is one of many predecessors to $\lambda_{i,..,k}$:

$$d^{in}_{i,..,k} = \min_j(d^{out}_{i,..,j}) \qquad (4\text{-}10a)$$

$$J^{in}_{i,..,k} = \min_j(d^{out}_{i,..,j} + J^{out}_{i,..,j}) - d^{in}_{i,..,k} \qquad (4\text{-}11a)$$

- *many-to-one nesting relationship (descending)* - when $\lambda_{i,..,j}$ is one of many parent transactions whose arrival is required to trigger that of the child activity $\lambda_{i,..,k}$:

$$d^{in}_{i,..,k} = \min_j(d^{in}_{i,..,j}) \qquad (4\text{-}12a)$$

$$J^{in}_{i,..,k} = \min_j(d^{in}_{i,..,j} + J^{in}_{i,..,j}) - d^{in}_{i,..,k} \qquad (4\text{-}13a)$$

- *many-to-one nesting relationship (ascending)* - when $\lambda_{i,..,j}$ is one of many child activities whose completion is required to trigger that of the parent transaction $\lambda_{i,..,k}$:

$$d^{out}_{i,..,k} = \min_j(d^{out}_{i,..,j}) \qquad (4\text{-}14a)$$

$$J^{out}_{i,..,k} = \min_j(d^{out}_{i,..,j} + J^{out}_{i,..,j}) - d^{out}_{i,..,k} \qquad (4\text{-}15a)$$

In the second case, where $Q_{i,..,k}$ is defined such that $\lambda_{i,..,k}$ is released only upon completion of *all* $\lambda_{i,..,j}$, equations *(4-10)* to *(4-15)* can be reduced as follows :

- *many-to-one precedence relationship* - when $\lambda_{i,..,j}$ is one of many predecessors to $\lambda_{i,..,k}$:

$$d^{in}_{i,..,k} = \max_j(d^{out}_{i,..,j}) \qquad (4\text{-}10b)$$

$$J^{in}_{i,..,k} = \max_j(d^{out}_{i,..,j} + J^{out}_{i,..,j}) - d^{in}_{i,..,k} \qquad (4\text{-}11b)$$

- *many-to-one nesting relationship (descending)* - when $\lambda_{i,..,j}$ is one of many parent transactions whose arrival is required to trigger that of the child activity $\lambda_{i,..,k}$:

$$d^{in}_{i,..,k} = \max_j(d^{in}_{i,..,j}) \qquad (4\text{-}12b)$$

$$J^{in}_{i,..,k} = \max_j(d^{in}_{i,..,j} + J^{in}_{i,..,j}) - d^{in}_{i,..,k} \qquad (4\text{-}13b)$$

- *many-to-one nesting relationship (ascending)* - when $\lambda_{i,..,j}$ is one of many child activities whose completion is required to trigger that of the parent transaction $\lambda_{i,..,k}$:

$$d^{out}_{i,..,k} = \max_j(d^{out}_{i,..,j}) \qquad (4\text{-}14b)$$

$$J^{out}_{i,..,k} = \max_j(d^{out}_{i,..,j} + J^{out}_{i,..,j}) - d^{out}_{i,..,k} \qquad (4\text{-}15b)$$

4.2.2 Initialisation and Finalisation of the End-to-End Timing Model

Initialisation of the end-to-end timing analysis model is straight forward, involving the assignment of input jitter and minimum I/O separation parameters for all nested transactions and activities that directly service the input events of the transaction. For example, for the transaction depicted in figure 4-1, this refers to the nested transaction λ_1. In the general case, the following assignments are made for each level 0 λ_i whose arrival is triggered directly by some transaction input event :

$$d_i^{in} = -J^{in} \quad (4\text{-}16)$$

$$J_i^{in} = J^{in} \quad (4\text{-}17)$$

where J^{in} is the transaction level input jitter, *ie.* the maximim variation in arrival time over all transaction input events.

Finalisation of the end-to-end analysis involves the assignment of transaction level minimum I/O separation and output jitter values. Transaction level values are determined by consolidating the values of the same parameters for all nested transactions and activities that directly service the output events of the transaction. For example, for the transaction depicted in figure 4-1, this refers to the activities λ_2, λ_3 and λ_4. In the general case, the following expressions are evaluated over all level 0 λ_i that relate directly to transaction output events :

$$\min_i(d_i^{out}) \le d \le \max_i(d_i^{out}) \quad (4\text{-}18)$$

$$\min_i(d_i^{out} + J_i^{out}) - d \le J^{out} \le \max_i(d_i^{out} + J_i^{out}) - d \quad (4\text{-}19)$$

where d and J^{out} are the transaction level minimum I/O separation and output jitter, respectively.

4.2.3 Evaluation of the End-to-End Timing Model

In order to evaluate the end-to-end timing model, values must be assigned to the localised (activity level) delay parameters $r_{i,..,k}$ and $R_{i,..,k}$ for each activity $\lambda_{i,..,k}$. In general terms, this can be done in one of two ways :

- by assigning $r_{i,..,k}$ and $R_{i,..,k}$ *budgets* for each $\lambda_{i,..,k}$ - this approach is illustrated by example in section 4.2.3.1;
- by calculating $r_{i,..,k}$ and $R_{i,..,k}$ values based on some localised timing analysis model - this approach is described (and illustrated by example) in section 4.3.2.

4.2.3.1 Example

This section illustrates the use of the end-to-end timing analysis model to support ‘top-down’ transaction definition and refinement. The example transaction topology used for illustration purposes in section 4.1 is also used here. To start, consider the assignment of end-to-end timing properties illustrated in figure 4-13.

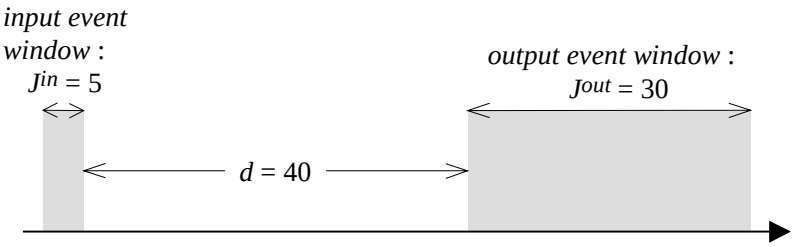


Figure 4-13 : Example Transaction - Timing Requirements

In practice, such details could be extracted from an overall statement of system level end-to-end timing requirements. The initial level 0 model of the transaction is depicted in figure 4-14. In practice, the level 0 topology could be extracted from (or form part of) the top level system design.

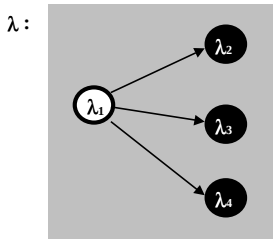


Figure 4-14 : Example Transaction - Level 0 Model

From the set of boundary conditions given in equations *(4-16)* to *(4-19)*, it is straight forward to assign a set of values to the corresponding level 0 timing attributes. Firstly, from the transaction input conditions given in equations *(4-16)* and *(4-17)*, values are directly inferred for the input jitter and initial accrued delay of λ_1 :

$$d_1^{in} = -5 \qquad J_1^{in} = 5$$

From the transaction output conditions given in equations *(4-18)* and *(4-19)*, suitable values can be found for the output jitter and final accrued delay of λ_2, λ_3 and λ_4 :

$$d_2^{out} = 40 \qquad J_2^{out} = 25$$
$$d_3^{out} = 45 \qquad J_3^{out} = 17$$
$$d_4^{out} = 55 \qquad J_4^{out} = 15$$

In practice, such parameter assignment could be based upon application-specific details. In this example, the parameters have been assigned such that the output event window of the transaction is exactly spanned by the set of activity level output windows. This means that the corresponding transaction level timing requirements have been met exactly, rather than leaving an element of redundancy in the transaction level requirements relative to the level 0 model timing properties. Beyond that, and the satisfaction of the boundary conditions given in equations *(4-16)* to *(4-19)*, the actual values assigned are somewhat arbitrary and chosen

purely for the purposes of illustration. Figure 4-15 illustrates this (partial) assignment of level 0 timing attributes.

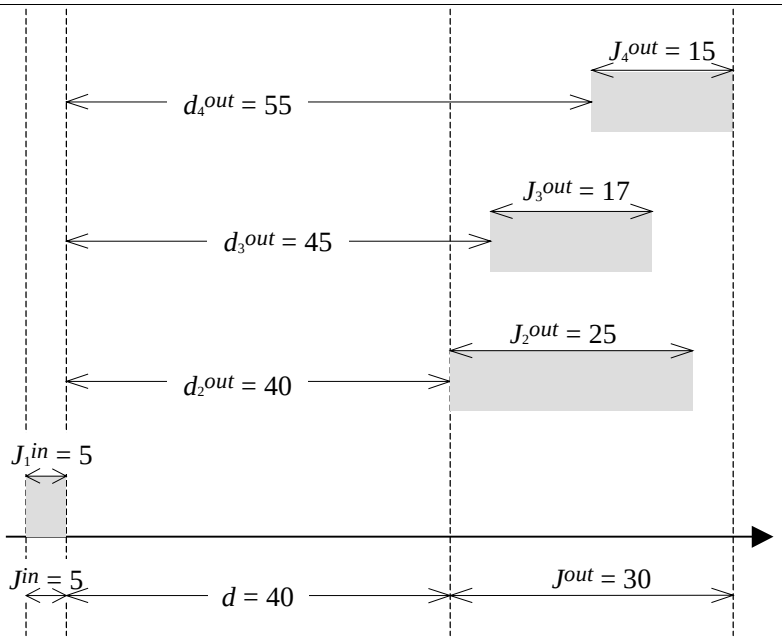


Figure 4-15 : Example Transaction - Partial Assignment of Level 0 Timing Attributes

The rest of the level 0 timing attributes can be assigned on the basis of the level 0 topology details and the appropriate means of accounting for precedence relationships as defined in equations *(4-4)* to *(4-9)*. Again in practice, additional application-specific information could be taken into account. In this example, values have been assigned as described below.

Given the one-to-many precedence relationship between λ_1 and its successor activities λ_2, λ_3 and λ_4, equation *(4-5)* implies that the parameters J_1^{out}, J_2^{in}, J_3^{in} and J_4^{in} should all be assigned the same value. Given that jitter tends to increase in the direction of control flow along the transaction (unless specific jitter control mechanisms are introduced - see section 5.1), this value should be less than any of the output jitter values already assigned for activities λ_2, λ_3 and λ_4. For the purposes of this example, the following assignment has been made :

$$J_1^{out} = J_2^{in} = J_3^{in} = J_4^{in} = 12$$

From equation *(4-4)*, the positions of the time windows whose widths are defined by the above jitter values are all fixed by the size of d_1^{out}. Hence, the parameters d_1^{out}, d_2^{in}, d_3^{in} and d_4^{in} should be assigned the same value. Given the topology of the transaction, a reasonable assignment for illustration purposes would be :

$$d_1^{out} = d_2^{in} = d_3^{in} = d_4^{in} = 35$$

The level 0 timing attributes are now sufficiently defined to fix the position of all input and output windows in the level 0 topology. Figure 4-16 illustrates this final assignment of level 0 timing attributes.

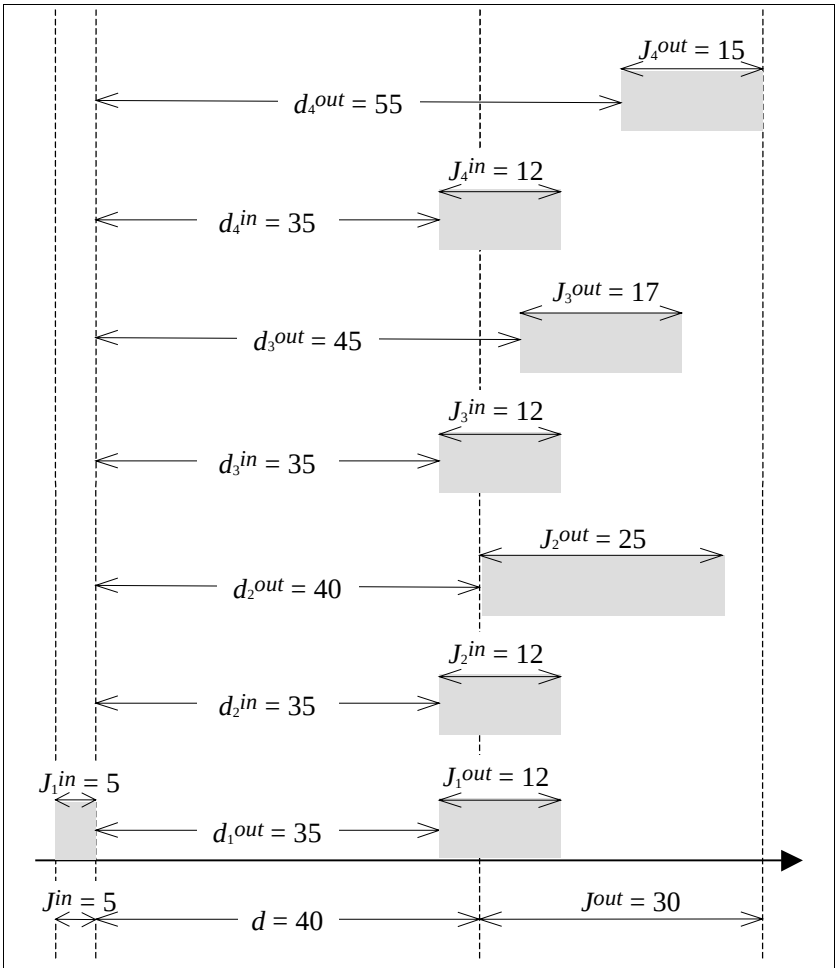


Figure 4-16 : Example Transaction - Final Assignment of Level 0 Timing Attributes

For the purposes of further decomposition (or direct implementation), a set of minimum and maximum response times and minimum I/O separation values can be determined (uniquely) for all level 0 activities from the application of equations *(4-3)*, *(4-2)* and *(4-1)*, respectively :

$r_1 = 40$ $R_1 = 47$ $d_1 = 35$

$r_2 = 5$ $R_2 = 18$ $d_2 = -7$

$r_3 = 10$ $R_3 = 15$ $d_3 = -2$

$r_4 = 20$ $R_4 = 23$ $d_4 = 8$

To illustrate how the model supports the ongoing refinement of timing properties, a second stage of decomposition is now illustrated. The level 1 model for the nested transaction λ_1 is depicted in figure 4-17.

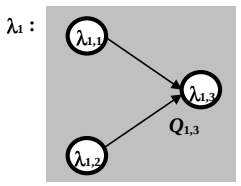


Figure 4-17 : Example Transaction - Level 1 Model for λ_1

From the statement of λ_1 timing attributes above and the set of equations *(4-4)* to *(4-15)*, it is straight forward to assign a set of values to the corresponding level 1 timing attributes. Firstly, given the one-to-many nesting relationship (descending) between λ_1 and its child input activities $\lambda_{1,1}$ and $\lambda_{1,2}$, equations *(4-6)* and *(4-7)* give :

$$d_{1,1}^{in} = -5 \qquad J_{1,1}^{in} = 5$$

$$d_{1,2}^{in} = -5 \qquad J_{1,2}^{in} = 5$$

Given the one-to-one nesting relationship (ascending) between λ_1 and its child output activity $\lambda_{1,3}$, equations *(4-8)* and *(4-9)* give :

$$d_{1,3}^{out} = 35 \qquad J_{1,3}^{out} = 12$$

Figure 4-18 illustrates this (partial) assignment of level 1 timing attributes for λ_1.

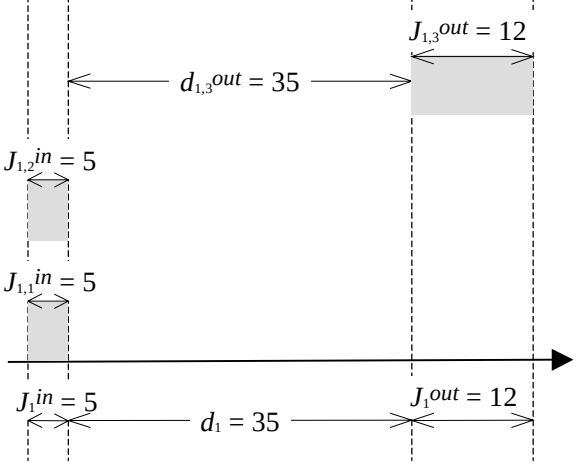


Figure 4-18 : Example Transaction - Partial Assignment of Level 1 Timing Attributes for λ_1

The rest of the level 1 timing attributes for λ_1 can be assigned on the basis of the level 1 topology details and the appropriate means of accounting for precedence relationships as defined in equations *(4-4)* to *(4-9)*. Note that the release condition $Q_{1,3}$ has already been specified such that $\lambda_{1,3}$ will be released only upon completion of both predecessors $\lambda_{1,1}$ and $\lambda_{1,2}$. Hence, the appropriate reduced forms of equations *(4-10)* and *(4-11)* apply, *ie.* equations *(4-10b)* and *(4-11b)*. Parameter values have been assigned as described below.

Given that jitter tends to increase in the direction of control flow along the transaction (unless specific jitter control mechanisms are introduced - see section 5.1), $J_{1,3}^{in}$ is assigned an appropriate intermediate value between the specified input and output jitter values for λ_1. $J_{1,1}^{out}$ and $J_{1,2}^{out}$ are then assigned with knowledge of $Q_{1,3}$ and the corresponding relationship with $J_{1,3}^{in}$ as expressed in equation *(4-11b)*. This expression implies that larger jitter values than $J_{1,3}^{in}$ can be assigned to some (though not all) predecessors in a many-to-one precedence relationship so long as the output window terminates no later than the required successor input window. On this basis, $J_{1,1}^{out}$ has been assigned a larger value than $J_{1,3}^{in}$, which means that this must be taken into account in the assignment of $d_{1,1}^{out}$ (see below). This leads to the following assignments :

$$J_{1,3}^{in} = J_{1,2}^{out} = 9 \qquad J_{1,1}^{out} = 11$$

An appropriate intermediate value can be assigned for $d_{1,3}^{in}$ to fix the position of the input window for $\lambda_{1,3}$. The accrued minimum delay requirements can then be specified for $\lambda_{1,1}$ and $\lambda_{1,2}$ such that the latest completion time for $\lambda_{1,1}$ is less than that of $\lambda_{1,2}$.

$$d_{1,3}^{in} = d_{1,2}^{out} = 20 \qquad d_{1,1}^{out} = 15$$

Figure 4-19 illustrates the final assignment of level 1 timing attributes for λ_1.

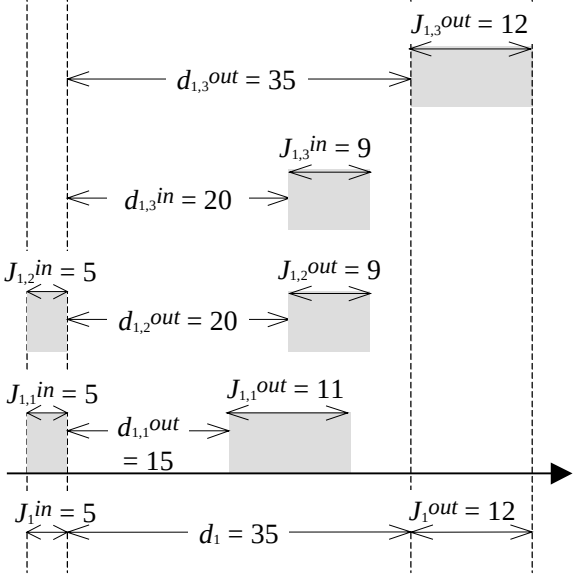


Figure 4-19 : Example Transaction - Final Assignment of Level 1 Timing Attributes for λ_1

For the purposes of further decomposition (or direct implementation), a set of minimum and maximum response times and minimum I/O separation values can be determined (uniquely)

for all level 1 activities from the application of equations *(4-3)*, *(4-2)* and *(4-1)*, respectively :

$r_{1,1} = 20$ $R_{1,1} = 26$ $d_{1,1} = 15$

$r_{1,2} = 25$ $R_{1,2} = 29$ $d_{1,2} = 20$

$r_{1,3} = 15$ $R_{1,3} = 18$ $d_{1,3} = 6$

A final stage of decomposition for the example transaction gives the set of level 2 timing attributes for the nested transaction λ_1 as depicted in figure 4-20. In practice, refinement of the transaction and its timing attributes could continue until the required level of detail is obtained. Clearly, the topological details generated at each stage of refinement and the number of refinement stages performed are application-specific.

Once again, a set of minimum and maximum response times and minimum I/O separation values can be determined (uniquely) via equations *(4-3)*, *(4-2)* and *(4-1)*, respectively :

	$r_{i,..,k}$	$d_{i,..,k}$	$R_{i,..,k}$
$\lambda_{1,1,1}$	15	10	18
$\lambda_{1,1,2}$	5	-3	8
$\lambda_{1,1,3}$	5	-3	12
$\lambda_{1,2,1}$	10	5	20
$\lambda_{1,2,2}$	20	15	22
$\lambda_{1,2,3}$	5	-2	7
$\lambda_{1,3,1}$	13	4	15
$\lambda_{1,3,2}$	2	-9	3

Table 4-1 : Example Transaction - Localised Timing Attributes for λ_1

Assuming no further refinement of the example transaction prior to its implementation, the final set of timing attributes represent constraints on the implementation of the system. During system implementation stages, however, the timing model could be evolved further in the same manner as above as a means of supporting progressive implementation (and integration) of the final system. It should be observed during any such evolution, however, that the stage at which the timing model become target-specific or integration-specific, *ie.* appropriate to a particular scheduling or communication regime, is the stage at which it ceases to support changes to the target system without the need to re-analyse the system (in part or whole) or possibly restate the model.

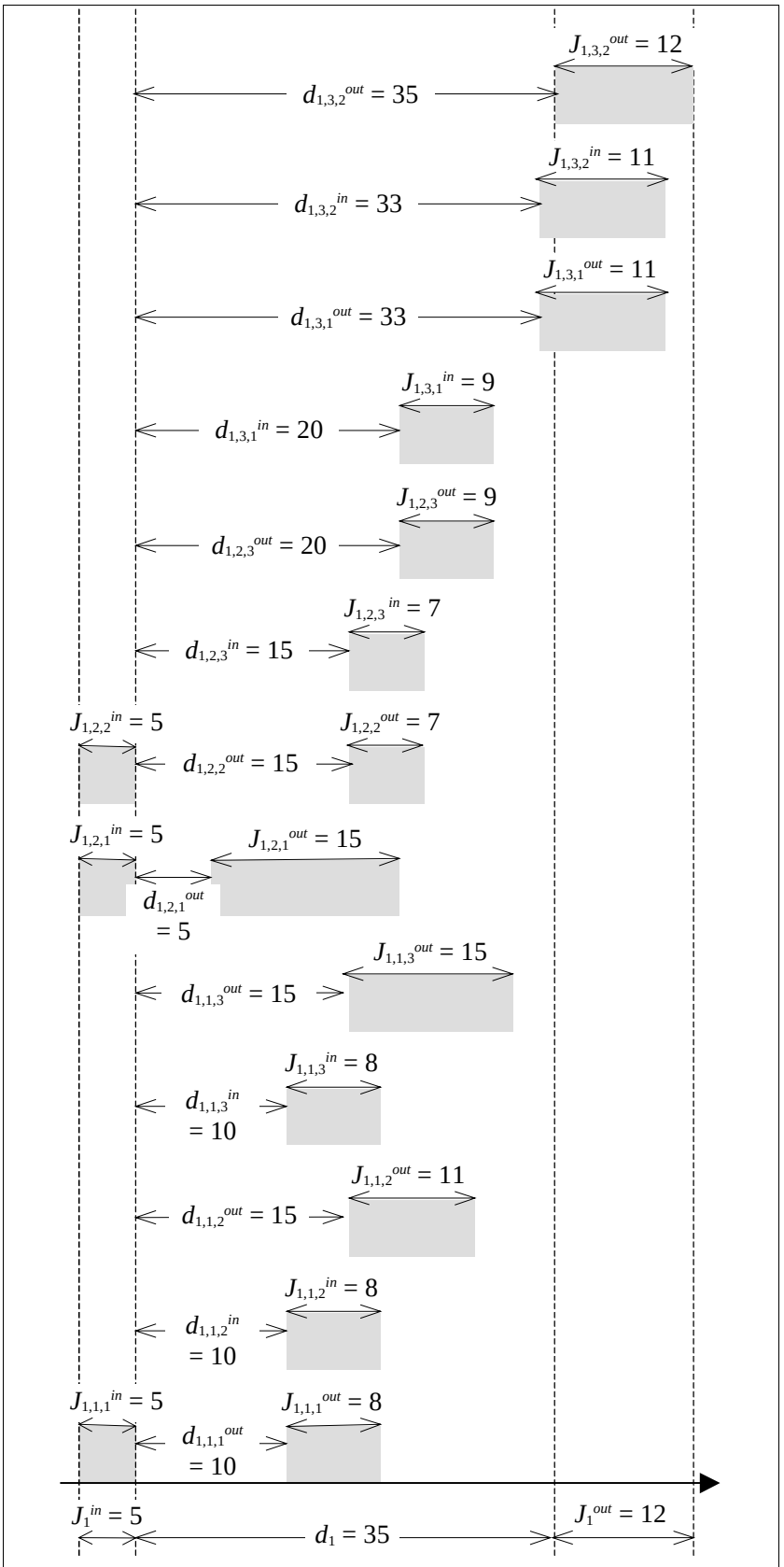


Figure 4-20 : Example Transaction - Final Assignment of Level 2 Timing Attributes for λ_1

When the final implementation and integration details of the system are known, the timing model must be verified, *ie.* the response time bounds defined in the abstract timing model must be shown to be safe. This requires some form of localised timing analysis model, *ie.* a model to determine activity level delay and jitter characteristics.

4.3 Localised Timing Analysis

Localised timing analysis refers to the analysis of activity level delay and jitter characteristics with respect to resource usage requirements. This involves the determination of the set of $r_{i,..,k}$ and $R_{i,..,k}$ values required to populate the end-to-end timing analysis framework described in section 4.2 based on some localised timing analysis model.

This level of analysis is, in general, dependent upon the implementation and integration details of the system, including :

- resource allocation and scheduling policies - these policies (including the scheduling of communication resources) dictate the form of schedulability analysis that is applicable;
- target hardware details - the implementation details of the hardware dictate the form of worst-case execution time analysis and analogous communication resource analysis models that are applicable.

Whilst any form of localised timing analysis that can deliver the response time and jitter values necessary to 'plug into' the end-to-end timing model could theoretically be applied, it is possible to construct an approach that preserves the key characteristics of the RBA end-to-end timing analysis model, *ie.* analysis of individual transactions independent from :

- other transactions;
- target hardware implementation details.

The approach taken, as initially suggested in section 3.3, is to work with an abstract model of system resources and associated scheduling policies. The use of any implementation-specific or system integration details is then deferred until a final verification exercise is performed for the system. This allows a seamless transition to be made from the abstract analysis stage of RBA through to the target-specific analysis of the final system, provided that the final scheduling solution conforms to certain assumptions prescribed by the localised RBA model, as described below.

4.3.1 Rate-Based Execution Model

The starting point for the rate-based execution model is a nested transaction graph that is defined to the extent that each $\lambda_{i,..,k}$ at the deepest level of nesting represents a single activity that can be assigned static requirements for one or more shared system resources. In other words, after the final stage of decomposition, activities should represent the stages within a transaction between which its resource requirements change. The information used to calculate the delay and jitter bounds for each activity is then restricted as follows :

- no details relating directly to any other activities can be used in the calculation, including those that may eventually be (or already have been) assigned to resources shared with the activity in question;
- no implementation-specific details of the underlying hardware can be used, including target-specific resource requirements.

The first constraint restricts the timing analysis for a given activity to the use of timing and resource information that refers only to that activity. This raises the problem of how to calculate delays incurred by one activity as a result of *interference* from others that share the same system resource(s). The solution, as initially proposed in section 3.2, is to physically partition the timing model. This is achieved by imposing a rate-based execution model on the run-time execution of activities whereby each activity can be guaranteed to make progress at some minimum specified rate irrespective of the behaviour of other activities allocated to the same resource(s). A rate parameter $v_{i,..,k}$ is associated with each activity $\lambda_{i,..,k}$, where $v_{i,..,k}$ can take any real value in the range [0, 1]. For example, a value of 0.5 means that the activity will execute for at least half of any observed time interval, irrespective of other activities that share resource access. At the limits of the valid range of values for $v_{i,..,k}$, a rate of 0 implies that no execution of the activity will ever occur and a rate of 1 implies that execution of the activity is continuous. More generally, the time $\chi_{i,..,k}$ for which each activity $\lambda_{i,..,k}$, will be executed in any interval of elapsed time $\rho_{i,..,k}$ is guaranteed to be no less than some minimum bound as follows :

$$\chi_{i,..,k} \geq v_{i,..,k} \rho_{i,..,k} \qquad (4\text{-}20)$$

Clearly, this impacts on the implementation of the final run-time scheduling solution for the system. In practice, since the theoretical model is continuous (the value of $\rho_{i,..,k}$ can be arbitrarily small) whereas the actual execution of any given activity will occur in discrete intervals of time, actual run-time rates of execution will vary. It is sufficient for the purposes of the localised delay analysis model, however, to ensure that the actual run-time rate of progress for each activity remains within specified bounds, the lower bound being $v_{i,..,k}$ (an upper bound $V_{i,..,k}$, required to ensure best-case requirements are met, will be discussed shortly).

The second constraint on the localised delay calculation raises the problem of how to express activity resource requirements in a target-independent form. To this end, associated with each activity is a pair of resource requirement bounds [$c_{i,..,k}$, $C_{i,..,k}$] by which the final implementation of that activity, *ie.* the combination of final software implementation and choice of target hardware, is ultimately constrained. Subsequently, any given implementation will be acceptable provided that the actual best- and worst-case resource requirements are bounded by [$c_{i,..,k}$, $C_{i,..,k}$].

Given a worst-case delay requirement $R_{i,..,k}$ then, from *(4-20)*, the time $\chi_{i,..,k}$ for which $\lambda_{i,..,k}$ will be executed is guaranteed to be no less than some minimum bound as follows :

$$\chi_{i,...,k} \geq v_{i,...,k} R_{i,...,k} \qquad (4\text{-}21)$$

Provided that the worst-case resource requirement $C_{i,..,k}$ is no greater than the minimum bound for $\chi_{i,..,k}$ then $\lambda_{i,..,k}$ will meet its worst-case delay requirement $R_{i,..,k}$. Consequently, the largest integer value that $C_{i,..,k}$ can be assigned such that $R_{i,..,k}$ is guaranteed to hold is as follows :

$$C_{i,...,k} = \left\lfloor v_{i,...,k} R_{i,...,k} \right\rfloor \qquad (4\text{-}22)$$

Conversely, given a worst-case resource requirement value $C_{i,..,k}$, the worst-case delay $R_{i,..,k}$ can be determined as follows :

$$R_{i,...,k} = \left\lceil \frac{C_{i,...,k}}{v_{i,...,k}} \right\rceil \qquad (4\text{-}23)$$

Alternatively, given a worst-case resource requirement $C_{i,..,k}$ and a worst-case delay requirement $R_{i,..,k}$, a suitable rate $v_{i,..,k}$ can be determined to as follows :

$$v_{i,...,k} = \frac{C_{i,...,k}}{R_{i,...,k}} \qquad (4\text{-}24)$$

The best-case behaviour of an activity must also be considered since this governs its minimum delay and, in turn, its maximum output jitter. There are two alternative ways to manage best-case requirements :

- analogous to the treatment of worst-case requirements, by specifying a *maximum* rate of execution $V_{i,..,k}$, with corresponding constraints on the run-time scheduling solution - this option is briefly discussed below;
- by enforcing best-case requirements directly at the end-to-end timing model level - this option is covered in section 5.1.

If a maximum rate $V_{i,..,k}$ is specified then the time $\chi_{i,..,k}$ for which each activity $\lambda_{i,..,k}$, will be executed in any interval of elapsed time $\rho_{i,..,k}$ can be bounded as follows :

$$\chi_{i,...,k} \leq V_{i,...,k} \rho_{i,...,k} \qquad (4\text{-}25)$$

Given a best-case delay requirement $r_{i,..,k}$ then, from *(4-25)*, the time $\chi_{i,..,k}$ for which $\lambda_{i,..,k}$ will be executed is bounded as follows :

$$\chi_{i,...,k} \leq V_{i,...,k} r_{i,...,k} \qquad (4\text{-}26)$$

Provided that the best-case resource requirement $c_{i,..,k}$ is no less than the maximum bound for $\chi_{i,..,k}$ then $\lambda_{i,..,k}$ will meet its best-case delay requirement $r_{i,..,k}$. Consequently, the smallest integer value that $c_{i,..,k}$ can be assigned such that $r_{i,..,k}$ is guaranteed to hold is as follows :

$$c_{i,..,k} = \lceil V_{i,..,k} r_{i,..,k} \rceil \qquad (4\text{-}27)$$

Conversely, given a best-case resource requirement value $c_{i,..,k}$, the best-case delay $r_{i,..,k}$ can be determined as follows :

$$r_{i,..,k} = \left\lfloor \frac{c_{i,..,k}}{V_{i,..,k}} \right\rfloor \qquad (4\text{-}28)$$

Alternatively, given a best-case resource requirement $c_{i,..,k}$ and a best-case delay requirement $r_{i,..,k}$, a suitable rate $V_{i,..,k}$ can be determined as follows :

$$V_{i,..,k} = \frac{c_{i,..,k}}{r_{i,..,k}} \qquad (4\text{-}29)$$

Equations *(4-27)* and *(4-28)* represent sufficient but not necessary conditions for $\lambda_{i,..,k}$ to meet its best-case delay requirement, since these are based on a maximum rate of execution which (since the actual rate will vary) will normally be greater than that achieved at run-time.

Having outlined analogous approaches for the treatment of best- and worst-case requirements, it is necessary to consider the combined use of these two approaches in order to ensure that the final assignment of timing attributes is valid, *ie.* $r_{i,..,k} \leq R_{i,..,k}$, $v_{i,..,k} \leq V_{i,..,k}$, and $c_{i,..,k} \leq C_{i,..,k}$. Figure 4-21 illustrates the general case of a valid attribute assignment, the shaded region representing the set of all permissible run-time behaviours for $\lambda_{i,..,k}$.

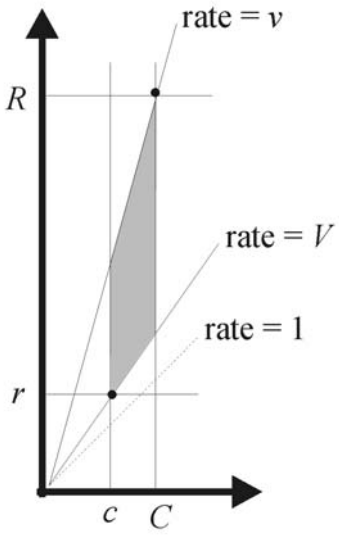


Figure 4-21 : Consistency of Best- and Worst-Case Timing Attributes

From a practical viewpoint, it is possible to identify a key trade-off in the assignment of best- and worst-case delay, rate and resource attributes using the techniques outlined above. As the best-case response time approaches the worst-case value, *ie.* $r_{i,..,k} \rightarrow R_{i,..,k}$, the best-case rate and resource parameters will also tend towards the values of their corresponding worst-case attributes, *ie.* $V_{i,..,k} \rightarrow v_{i,..,k}$ and $c_{i,..,k} \rightarrow C_{i,..,k}$. This scenario is illustrated in figure 4-22 which shows how, after assigning the worst-case attributes $R_{i,..,k}$, $v_{i,..,k}$ and $C_{i,..,k}$, the scope for assigning valid best-case values $V_{i,..,k}$ and $c_{i,..,k}$ is continually reduced as $r_{i,..,k} \rightarrow R_{i,..,k}$. The practical implication of this is a potential lack of flexibility in the implementation and run-time execution of any activity that is required to exhibit low output jitter, *ie.* low variability in response time. A better method for treating low output jitter requirements is to adopt transaction level jitter control mechanisms (in conjunction with the above method for assignment of worst-case attributes) as described in section 5.1.

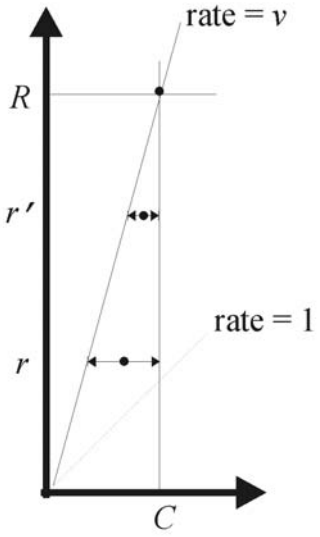


Figure 4-22 : Timing Attribute Assignment as $r_{i,..,k} \rightarrow R_{i,..,k}$

These expressions (and related assumptions) describe a simple abstract resource model that allows activity resource requirements to be specified before the actual target hardware is known. Eventually, when the target becomes known (or changed at a later date), target-specific values must be checked for sufficiency. Due to the manner in which the abstract timing model is constructed, however, verification of the final system timing properties can be performed on a per-activity basis (since activity response times are dependent only upon their own rate and resource requirements). Hence, the transition from target-independent to target-dependent timing analysis can be made at the most appropriate times for different applications or transactions within the system. Similarly, any timing guarantees generated for the system can be re-verified on a per-activity basis following any subsequent changes to the target hardware or software implementation details.

4.3.2 Example

The example transaction used to illustrate the end-to-end timing analysis model in section 4.2 is now extended to illustrate the localised timing analysis approach described above. Once again, the example illustrates the use of the timing model in top-down mode, where resource requirements are derived from given delay and jitter requirements, rather than the traditional bottom-up mode of applying timing analysis, where delay and jitter characteristics are predicted given a set of resource requirements.

The figures given in table 4-1 can be used in conjunction with the localised timing model described above to determine a set of execution rates and resource requirements such that the example transaction will be schedulable under any valid implementation of the RBA rate-based execution model. A set of worst-case attributes are given in table 4-2.

	$R_{i,..,k}$	$v_{i,..,k}$	$C_{i,..,k}$
$\lambda_{1,1,1}$	18	0.4	7
$\lambda_{1,1,2}$	8	0.3	2
$\lambda_{1,1,3}$	12	0.5	6
$\lambda_{1,2,1}$	20	0.2	4
$\lambda_{1,2,2}$	22	0.4	8
$\lambda_{1,2,3}$	7	0.3	2
$\lambda_{1,3,1}$	15	0.2	3
$\lambda_{1,3,2}$	3	0.4	1
λ_{2}	18	0.3	5
λ_{3}	15	0.3	4
λ_{4}	23	0.2	4

Table 4-2 : Example Transaction - Assignment of Worst-Case Execution Rates and Resource Requirements

The above set of $C_{i,..,k}$ values represent constraints on the final implementation of the transaction. Similarly, the set of $v_{i,..,k}$ values represent constraints on the run-time scheduling solution (assuming compliance with the RBA rate-based execution model). Provided these constraints are observed in the final system, then the set of localised worst-case response time requirements specified for the transaction will be met. In turn, provided that the set of precedence relationships implied by the transaction topology are enforced, the transaction will meet its worst-case end-to-end response time requirements.

4.4 Summary

This chapter has described the proposed reservation-based timing analysis (RBA) approach. The approach can be summarised as follows :

- *Representation and refinement of system timing properties* :
  The end-to-end timing characteristics of the system are captured, developed and analysed in terms of *transactions*. The transaction model is hierarchical, in the form of an acyclic, directed, nested graph, reflecting the need to support an evolving definition of the system during development. The leaf nodes of the graph represent the concurrent processing and communication elements within the transaction, termed *activities*; non-leaf nodes are referred to as *nested transactions*. The edges (or arcs) of the graph capture the precedence and nesting relationships within the transaction. The parameters via which timing behaviour is represented and observed are the same for a single activity, a group of related activities, a nested transaction and a level 0 (system level) transaction, thus providing a highly composable and scaleable timing model. The transaction model, in terms of both structure and timing properties, is target-independent.

- *End-to-end (system level) timing analysis* :
  The overall timing behaviour of the system at a given stage of evolution can be assessed by analysing the current set of transaction graphs. The end-to-end timing behaviour of each transaction can be assessed independently from other transactions in the system (including those which may eventually share processing and/or communication resources) and from the eventual target hardware implementation details. End-to-end delays and jitter are determined as a function of the localised (activity level) delays, depending on the forms of precedence relationships and nesting relationships involved. Localised delay values can be substituted directly in the form of *budgets* or acquired indirectly via some form of localised timing analysis.

- *Localised (activity level) timing analysis* :
  Activity level delays and jitter can be determined in a target-independent manner based on an abstract, rate-based model of execution. This model allows the worst-case response time of each activity to be expressed as a function of its own worst-case resource requirements and its assigned minimum execution rate, independently from other activities allocated to the same shared resource(s). When the target platform (hardware and software) implementation details and the system integration (allocation and scheduling) details are finalised, activities can be then verified independently. This provides the basis for a smooth transition between abstract timing analysis and target-dependent analysis at the appropriate time for each part of the system under development. The rate-based execution model also provides a degree of independence from the final integration details of the system, including the precise form of run-time scheduling solution.

CHAPTER 5. Extensions to the RBA End-to-End Timing Model

This chapter describes the following extensions to the RBA end-to-end timing analysis model as presented in the previous chapter :

- how to control jitter accumulation at the transaction level;
- how to derive upper and lower bounds for the separation in time between any two (or more) concurrent events within a transaction.

5.1 Transaction Level Jitter Control Mechanisms

The transaction model described in chapter 4 is fundamentally event-triggered. Such models are prone to producing a high degree of jitter in the end-to-end response times predicted. This may be perfectly acceptable in certain applications but, in others, some means of limiting such jitter may be required. For example, an output to a sensor or a control loop feedback may need to occur in a stable periodic manner. More generally, there may not be such strict limitations required but still a tolerance on the maximum degree of jitter that is acceptable.

Note that to allow such limitations to be placed on jitter is not the same as functionally partitioning the timing model as discussed in section 3.2. The motivation is solely to allow application-specific jitter constraints to be implemented and is not linked to the avoidance of an holistic analysis of the system which would, in any case, encourage a pure time-triggered transaction model.

RBA allows the maximum input jitter associated with any node $\lambda_{i,..,k}$ of a transaction graph λ to be specified independently from the value it would otherwise inherit as a result of the timing behaviour of its predecessors. The inherited value corresponds to that which is calculated from the pure event-triggered model as described in section 4.2 (via equations *(4-4)* to *(4-15)* depending on the type of precedence or nesting relationship involved). RBA transactions can therefore be more generally referred to as *jitter-controlled* - with time-triggered and event-triggered transactions representing the two extremes of behaviour.

Consider the imposition of a fixed jitter tolerance value $J_{i,..,k}^{in^*}$ at node $\lambda_{i,..,k}$. Let $J_{i,..,k}^{in\leftarrow}$ denote the input jitter value that would otherwise be inherited by $\lambda_{i,..,k}$, with corresponding minimum accrued delay $d_{i,..,k}^{in\leftarrow}$. Similarly, let $J_{i,..,k}^{in\rightarrow}$ denote the input jitter value that is actually inherited by $\lambda_{i,..,k}$, after taking the specified jitter tolerance into account, with corresponding minimum accrued delay $d_{i,..,k}^{in\rightarrow}$. Three cases arise (as illustrated in figure 5-1) :

- $J_{i,..,k}^{in^*} < J_{i,..,k}^{in\leftarrow}$ - a (positive) maximum jitter tolerance value is specified that is less than the predicted inherited value;

- $J_{i,..,k}^{in*} \geq J_{i,..,k}^{in\leftarrow}$ - a (positive) maximum jitter tolerance value is specified that is greater than (or equal to) the predicted inherited value;
- $J_{i,..,k}^{in*} \leq 0$ - a negative (or zero) value is specified.

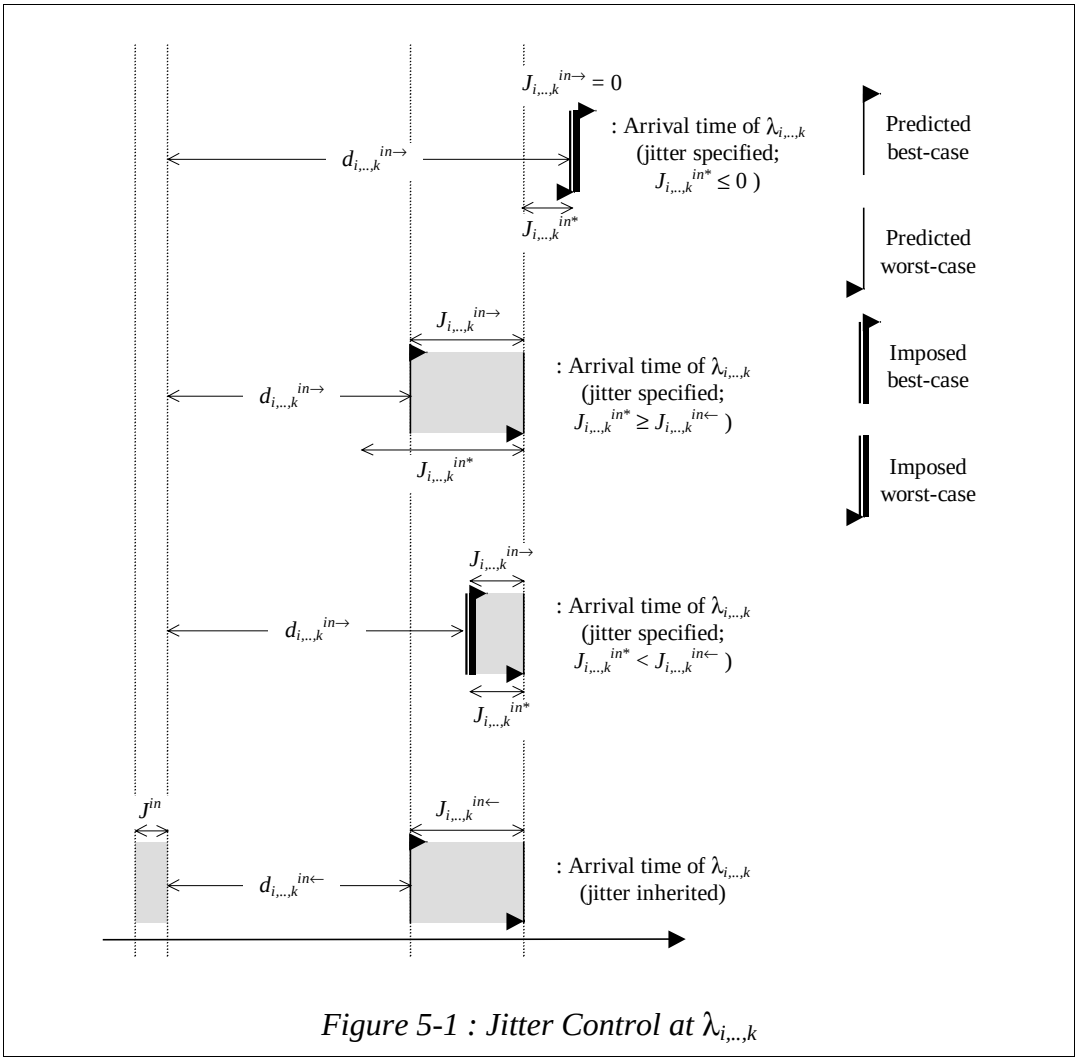


Figure 5-1 : Jitter Control at $\lambda_{i,..,k}$

The impact on the timing attributes of the transaction at $\lambda_{i,..,k}$ can be stated as follows :

- when $J_{i,..,k}^{in*} < J_{i,..,k}^{in\leftarrow}$, the net effect is to reduce the input jitter of $\lambda_{i,..,k}$ compared to that which would otherwise be inherited and to give an equivalent increase in the minimum accrued delay :

$$J_{i,..,k}^{in\rightarrow} = J_{i,..,k}^{in*} \qquad (5\text{-}1)$$

$$d_{i,..,k}^{in\rightarrow} = d_{i,..,k}^{in\leftarrow} + J_{i,..,k}^{in\leftarrow} - J_{i,..,k}^{in*} \qquad (5\text{-}2)$$

- when $J_{i,..,k}^{in*} \geq J_{i,..,k}^{in\leftarrow}$, there is no impact :

$$J_{i,..,k}^{in\rightarrow} = J_{i,..,k}^{in\leftarrow} \qquad (5\text{-}3)$$

$$d_{i,..,k}^{in\rightarrow} = d_{i,..,k}^{in\leftarrow} \qquad (5\text{-}4)$$

- when $J_{i,..,k}^{in^*} \leq 0$, the net effect is to eliminate jitter completely (the arrival time of $\lambda_{i,..,k}$ is fixed to a point in time at or beyond its inherited worst-case, *ie.* $\lambda_{i,..,k}$ becomes time-triggered) and to increase the minimum accrued delay accordingly :

$$J_{i,..,k}^{in\rightarrow} = 0 \qquad (5\text{-}5)$$

$$d_{i,..,k}^{in\rightarrow} = d_{i,..,k}^{in\leftarrow} + J_{i,..,k}^{in\leftarrow} - J_{i,..,k}^{in^*} \qquad (5\text{-}6)$$

The separate cases identified above can be consolidated (without generalisation) as follows :

$$J_{i,..,k}^{in\rightarrow} = \max(0, \min(J_{i,..,k}^{in^*}, J_{i,..,k}^{in\leftarrow})) \qquad (5\text{-}7)$$

$$d_{i,..,k}^{in\rightarrow} = d_{i,..,k}^{in\leftarrow} + \max(0, (J_{i,..,k}^{in\leftarrow} - J_{i,..,k}^{in^*})) \qquad (5\text{-}8)$$

Note that it is both theoretically and practically reasonable to assign a negative value to $J_{i,..,k}^{in^*}$. The result is an unconditional delay in the release of $\lambda_{i,..,k}$ beyond its inherited worst-case arrival time. This is analogous to a release *offset* as prescribed for fixed-priority scheduling solutions with time-triggered transactions [Audsley93, Bate97].

5.1.1 Impact on Localised Timing Analysis

The limitation of input jitter at $\lambda_{i,..,k}$ will normally have an impact on the localised timing attributes of $\lambda_{i,..,k}$. The extent to which localised timing attributes are affected depends upon the relative value of the specified jitter tolerance $J_{i,..,k}^{in^*}$ compared to the jitter that would otherwise be inherited, as calculated from equation *(4-3)*. Considering the same three cases as above and using the superscripts $\leftarrow$ and $\rightarrow$ to denote parameter values before and after jitter controls have been imposed, the impact on localised timing attributes can be seen as follows :

- when $J_{i,..,k}^{in^*} < J_{i,..,k}^{in\leftarrow}$, the minimum response time requirement of $\lambda_{i,..,k}$ is reduced :

$$r_{i,..,k}^{\rightarrow} = r_{i,..,k}^{\leftarrow} - (J_{i,..,k}^{in\leftarrow} - J_{i,..,k}^{in^*}) \qquad (5\text{-}9)$$

- when $J_{i,..,k}^{in^*} \geq J_{i,..,k}^{in\leftarrow}$, there is no impact;

- when $J_{i,..,k}^{in^*} \leq 0$, the minimum response time requirement of $\lambda_{i,..,k}$ is reduced, as is the minimum I/O separation for $\lambda_{i,..,k}$:

$$r_{i,..,k}^{\rightarrow} = r_{i,..,k}^{\leftarrow} - (J_{i,..,k}^{in\leftarrow} - J_{i,..,k}^{in^*}) \qquad (5\text{-}10)$$

$$R_{i,..,k}^{\rightarrow} = R_{i,..,k}^{\leftarrow} + J_{i,..,k}^{in^*} \qquad (5\text{-}11)$$

$$d_{i,..,k}^{\rightarrow} = d_{i,..,k}^{\leftarrow} + J_{i,..,k}^{in^*} \qquad (5\text{-}12)$$

Once again, the separate cases identified above can be consolidated (without generalisation) as follows :

$$r_{i,..,k}^{in\rightarrow} = r_{i,..,k}^{in\leftarrow} - \max(0,(J_{i,..,k}^{in\leftarrow} - J_{i,..,k}^{in*})) \quad \textit{(5-13)}$$

$$R_{i,..,k}^{\rightarrow} = R_{i,..,k}^{\leftarrow} + \min(0, J_{i,..,k}^{in*}) \quad \textit{(5-14)}$$

$$d_{i,..,k}^{\rightarrow} = d_{i,..,k}^{\leftarrow} + \min(0, J_{i,..,k}^{in*}) \quad \textit{(5-15)}$$

In order to maintain the restriction that, for any given instance of $\lambda_{i,..,k}$, the output event window cannot begin before the input event widow begins, the condition $d_{i,..,k} > -J_{i,..,k}^{in}$ (taken from the original statement of transaction timing properties in section 4.1.2) must be preserved. This restricts the assignment of a maximum jitter tolerance value at $\lambda_{i,..,k}$ such that $J_{i,..,k}^{in*} > -d_{i,..,k}^{\leftarrow}$.

5.1.2 Example 1

The example transaction used to illustrate RBA in the previous chapter is now extended to illustrate transaction level jitter control. Suppose that the transaction output activity λ_4 cannot tolerate input jitter to the same extent as the other output activities λ_2 and λ_3. Hence, the input jitter inherited directly from λ_1 may be acceptable for λ_2 and λ_3 but not for λ_4. Consider the following two cases :

- $J_4^{in*} = 4$ - a maximum input jitter tolerance is defined for λ_4;
- $J_4^{in*} = -4$ - the arrival of λ_4 is offset from its otherwise inherited worst-case arrival time, *ie.* λ_4 is time-triggered.

Firstly, consider the impact of $J_4^{in*} = 4$ on end-to-end timing attributes. Applying equations *(5-1)* and *(5-2)*, values before and after transformation are as follows :

Before :	$J_4^{in\leftarrow} = 12$	$d_4^{in\leftarrow} = 35$
After :	$J_4^{in\rightarrow} = 4$	$d_4^{in\rightarrow} = 43$

Figure 5-2a illustrates these results.

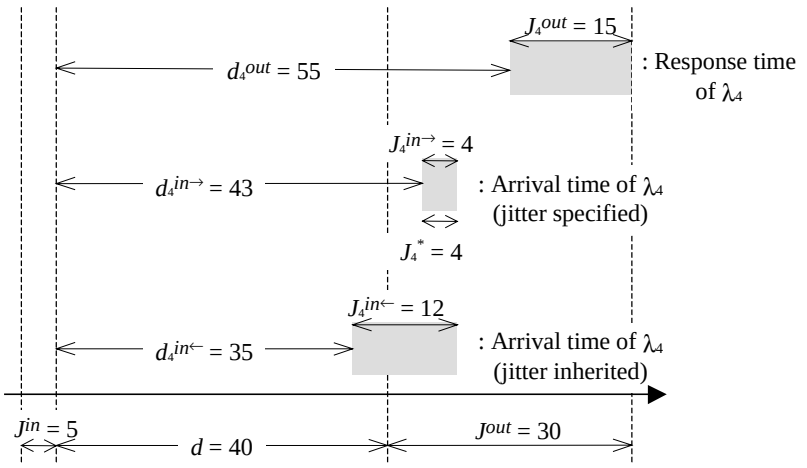


Figure 5-2a : Example Transaction - Jitter Control at λ_4 (Positive)

The corresponding impact on localised timing attributes can be determined simply from equation *(5-9)* - there is no impact on minimum I/O separation and maximum response time requirements :

Before :	$r_4^{\leftarrow} = 20$	$R_4^{\leftarrow} = 23$	$d_4^{\leftarrow} = 8$
After :	$r_4^{\rightarrow} = 12$	$R_4^{\rightarrow} = 23$	$d_4^{\rightarrow} = 8$

Now consider the case of a defined release offset $J_4^{in*} = -4$. The impact on end-to-end timing attributes is determined from equations *(5-5)* and *(5-6)*. Values before and after transformation are as follows :

Before :	$J_4^{in\leftarrow} = 12$	$d_4^{in\leftarrow} = 35$
After :	$J_4^{in\rightarrow} = 0$	$d_4^{in\rightarrow} = 51$

Figure 5-2b illustrates these results.

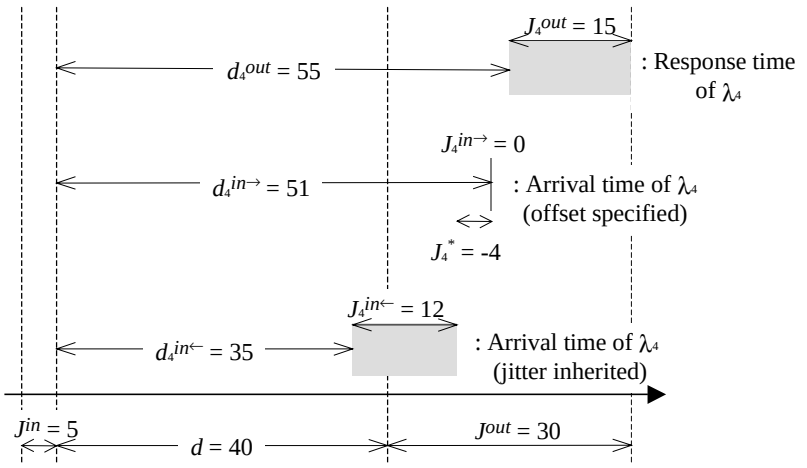


Figure 5-2b : Example Transaction - Jitter Control at λ_4 (Negative)

The corresponding impact on localised timing attributes can be determined simply from equations *(5-10)* , *(5-11)* and *(5-12)* :

Before :	$r_4^{\leftarrow} = 20$	$R_4^{\leftarrow} = 23$	$d_4^{\leftarrow} = 8$
After :	$r_4^{\rightarrow} = 4$	$R_4^{\rightarrow} = 19$	$d_4^{\rightarrow} = 4$

5.1.3 Example 2

This example illustrates the use of jitter control mechanisms to alleviate the need to implement best-case localised timing requirements explicitly at the activity level. Recall from section 4.3 that this is preferable for activities whose minimum response time requirement is close to that for the maximum response time (leaving limited scope for variability in the resource usage characteristics of the activity, *eg.* computation time, in its final implemented form). An alternative motivation could be the impracticality (or inability) to determine effective best-case resource usage bounds by static analysis or testing.

Consider the general case of a set of precedence-constrained activities, *ie.* transaction, denoted by λ. Suppose that the activity $\lambda_{i,..,k}$ is implemented such that its worst-case resource requirements are observed but any best-case requirements are ignored. In turn, suppose that $\lambda_{i,..,k}$ is executed according to the RBA rate-based execution model to meet its worst-case timing requirements but with no concern for its best-case response time. Under such circumstances, the best-case response time of the activity cannot be bounded any more optimistically than by the interval [0, $R_{i,..,k}$]. Since a response time anywhere in the interval [0, $r_{i,..,k}$) represents completion of the activity *too early*, there is potential to invalidate the localised and end-to-end timing analysis results, with a risk of damage to the system and/or some aspect of its operating environment. Note that required event *orderings* as captured in the transaction topology will still be preserved, but the *times* at which events occur subsequent to the execution of $\lambda_{i,..,k}$ will be jeopardised with respect to the end-to-end timing analysis.

To illustrate this by example, consider the example transaction and, in particular, the set of precedence-constrained activities $\lambda_{1,1,1}$, $\lambda_{1,1,2}$ and $\lambda_{1,1,3}$. Suppose that the activity $\lambda_{1,1,1}$ is implemented such that its worst-case resource requirement $C_{1,1,1}$ is observed but any best-case requirements are ignored. In turn, suppose that $\lambda_{1,1,1}$ is executed according to the RBA rate-based execution model to meet its worst-case timing requirement $R_{1,1,1}$ but with no concern for its best-case response time. Under such circumstances, the best-case response time of $\lambda_{1,1,1}$ is bounded only by the interval [0, $R_{1,1,1}$], with the potential to complete too early, some time in the interval [0, $r_{1,1,1}$). A potential consequence of this is the arrival of activities $\lambda_{1,1,2}$ and $\lambda_{1,1,3}$ being triggered earlier than specified. Early triggering of $\lambda_{1,1,2}$ and $\lambda_{1,1,3}$ can be prevented without modifying the behaviour of $\lambda_{1,1,1}$, however, by enforcing

maximum jitter tolerances $J_{1,1,2}^{in^*}$ and $J_{1,1,3}^{in^*}$ equal to the original input jitter values specified for $\lambda_{1,1,2}$ and $\lambda_{1,1,3}$ under the (now invalid) assumption that $\lambda_{1,1,1}$ will complete no earlier than its specified minimum response time $r_{1,1,1}$.

Consider the case of $\lambda_{1,1,2}$ (the case of $\lambda_{1,1,3}$ is analogous). Since the minimum response time of $\lambda_{1,1,1}$ is potentially as low as zero, $\lambda_{1,1,2}$ could inherit an input jitter value $J_{1,1,2}^{in\leftarrow} = 23$ (applying equation *(4-2)* to determine that $J_{1,1,1}^{out} = 23$ and then *(4-5)* to assign $J_{1,1,2}^{in\leftarrow}$).Hence, set a jitter tolerance value $J_{1,1,2}^{in^*} = 8$ (the original input jitter value specified for $\lambda_{1,1,2}$ on the basis of $\lambda_{1,1,1}$ meeting its output jitter specification) to counteract the potential for $\lambda_{1,1,2}$ to arrive early due to early completion of $\lambda_{1,1,1}$. Since $J_{1,1,2}^{in^*} < J_{1,1,2}^{in\leftarrow}$, the impact on the arrival time attributes of $\lambda_{1,1,2}$ is captured in equations *(5-1)* and *(5-2)* - values before and after transformation are as follows :

Before :	$J_{1,1,3}^{in\leftarrow} = 23$	$d_{1,1,3}^{in\leftarrow} = -5$
After :	$J_{1,1,3}^{in\rightarrow} = 8$	$d_{1,1,3}^{in\rightarrow} = 10$

Hence, the arrival time behaviour of $\lambda_{1,1,2}$ will meet its specification irrespective of the response time behaviour of $\lambda_{1,1,1}$. Figure 5-3 illustrates these results.

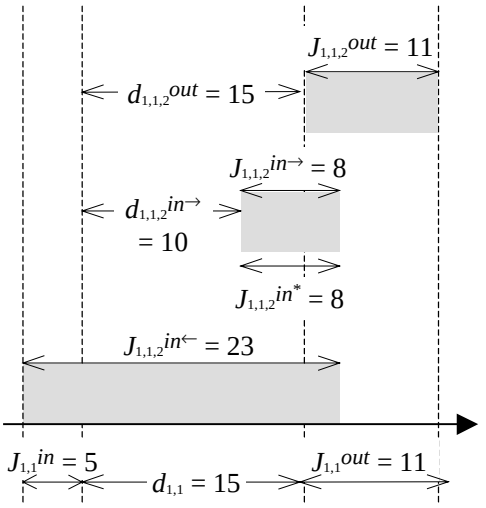


Figure 5-3 : Example Transaction - Final Assignment of Level 2 Timing Attributes for $\lambda_{1,1}$

5.2 Determining Event Phase Relationships

During the development of a real-time system in practice, it may be necessary to determine the maximum variation in the time of occurrence of *multiple* events within the system - termed the *phase* relationship between the given set of events and denoted by Ω. The phase relationship is simply the difference between the latest and earliest possible times that any event in the observed set can occur. The term ‘event’ refers to an arrival event or completion

event of any activity or nested transaction (given their equivalence). RBA allows this calculation to be performed for sets of events belonging to the same transaction and for sets of events that span multiple transactions (with a certain proviso).

5.2.1 Phase Relationships Within a Single Transaction

Firstly, consider the case of a set of events belonging to the same transaction. Let $e_{i,..,j}$ denote any one member of a set of observed events belonging to some transaction λ. The phase relationship between these events is equal to the difference between the latest time by which all events must have occurred and the earliest time that any one event could occur, expressed as follows :

$$\Omega = \max_{j}(d^{e}_{i,..,j} + J^{e}_{i,..,j}) - \min_{j}(d^{e}_{i,..,j}) \qquad \textit{(5-16)}$$

where $d^{e}_{i,..,j}$ and $J^{e}_{i,..,j}$ respectively denote the minimum accrued delay and maximum jitter associated with each event $e_{i,..,j}$. Note that the values of these parameters are available directly from the end-to-end timing analysis results (if the model is being applied bottom-up) or parameter assignment (if the model is being applied top-down) for the transaction.

The phase calculation can be illustrated by example by returning to the parameter assignment for the previous example transaction, specifically that for the nested transaction λ_1 as depicted in figure 4-14. Suppose that it is necessary to determine the phase relationship between the arrival events of the three activities $\lambda_{1,1,2}$, $\lambda_{1,1,3}$ and $\lambda_{1,2,3}$, between which there is no direct causal relationship. Denote the arrival events of these activities by $e^{in}_{1,1,2}$, $e^{in}_{1,1,3}$ and $e^{in}_{1,2,3}$, respectively. Applying equation *(5-16)* gives the following result (which can also be seen from the illustration in figure 5-4) :

$\Omega\,(e^{in}_{1,1,2},\ e^{in}_{1,1,3},\ e^{in}_{1,2,3})$ = max (18, 18, 22) - min (10, 10, 15) = 12

Note that, in certain circumstances, the evaluation of equation *(5-16)* can be dominated by the attributes of a single event. This is the case when the same event $e_{i,..,j}$ determines both $\max_{j}(d^{e}_{i,..,j} + J^{e}_{i,..,j})$ and $\min_{j}(d^{e}_{i,..,j})$, beyond which the details of all other observed events are masked. For example, consider the pair of events $e^{out}_{1,1,3}$ and $e^{in}_{1,3,1}$, whose behaviour can be seen in figure 5-4. Applying equation *(5-16)* gives :

$\Omega\,(e^{out}_{1,1,3},\ e^{in}_{1,3,1})$ = max (30, 29) - min (15, 20) = 15

It can be seen that the calculation is dominated by the attributes of event $e^{out}_{1,1,3}$ and that the attributes of $e^{in}_{1,3,1}$ do not directly influence the result.

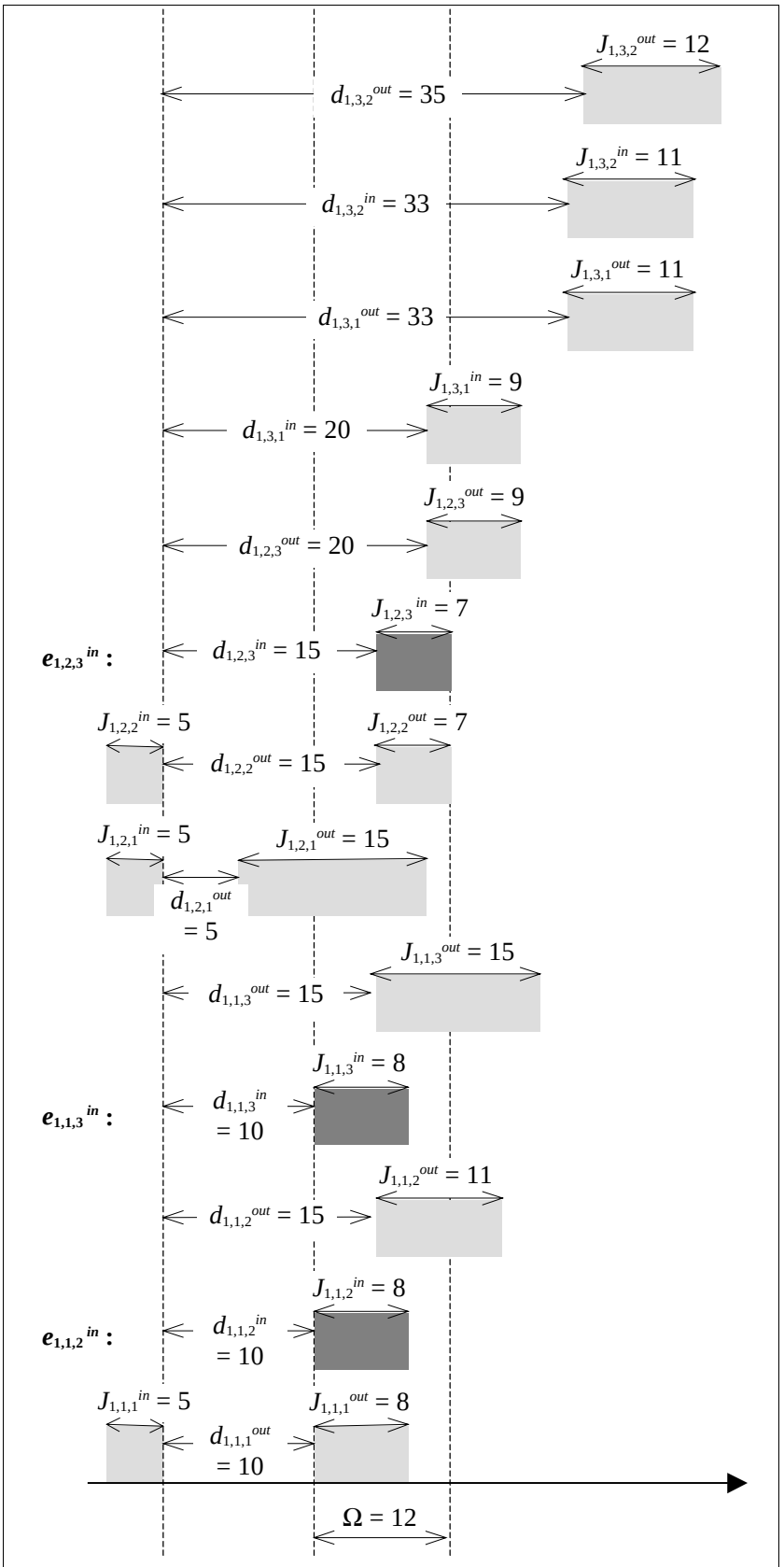


Figure 5-4 : Example Transaction - Phase Calculation for Events $e_{1,1,2}^{in}$, $e_{1,1,3}^{in}$ and $e_{1,2,3}^{in}$ in λ_1

5.2.2 Phase Relationships Across Multiple Transactions

Consider now the case of a set of events that spans multiple transactions. The phase calculation can only be performed if the transaction arrival phase relationship is either specified (fixed during a top-down parameter assignment) or determinable (from the results of a bottom-up analysis of the system). The transaction arrival phase relationship is the maximum time variation over all transaction arrival events for the set of transactions under observation - denote this value by ω.

Let $e_{\ldots,j}$[†] denote any one member of a set of observed events spanning multiple transactions with arrival phase ω. The phase relationship between this set of events is, once again, equal to the difference between the latest time by which all events must have occurred and the earliest time that any one event could occur. Unlike the single transaction case, however, where all observed events are relative to the same transaction input event window, the set of observed events relate to different transactions between which there is a degree of uncertainty in the relative positions of their input event windows, *ie.* ω. This gives rise to an amended expression as follows (with further explanation below) :

$$\Omega = \max_{j}(d^{e}_{\ldots,j} + J^{e}_{\ldots,j}) - \min_{j}(d^{e}_{\ldots,j} - \omega_{\ldots,j}) \qquad (5\text{-}17)$$

where $d^{e}_{\ldots,j}$ and $J^{e}_{\ldots,j}$ respectively denote the minimum accrued delay and maximum jitter associated with each event $e_{\ldots,j}$ and $\omega_{\ldots,j}$ is the difference between ω and the input jitter for the transaction to which event $e_{\ldots,j}$ belongs. Hence, for a specific event belonging to transaction λ with transaction level input jitter J^{in}, the term $\omega_{\ldots,j}$ would be calculated as follows :

$$\omega_{\ldots,j} = \omega - J^{in} \qquad (5\text{-}18)$$

Note that equation *(5-17)* is pessimistic due to uncertainty in the position of the input event window of each individual transaction within the overall multi-transaction arrival phase window. Given this uncertainty, assumptions must be made about the best-case and worst-case times of occurrence of each event $e_{\ldots,j}$ (relative to the transaction level arrival phase window) that are mutually incompatible. This situation is depicted in figure 5-5 which shows the best-case and worst-case scenarios for the same event $e_{\ldots,j}$.

[†] the lead index *i* has been dropped in the subscript to indicate that the event is tied to no specific transaction.

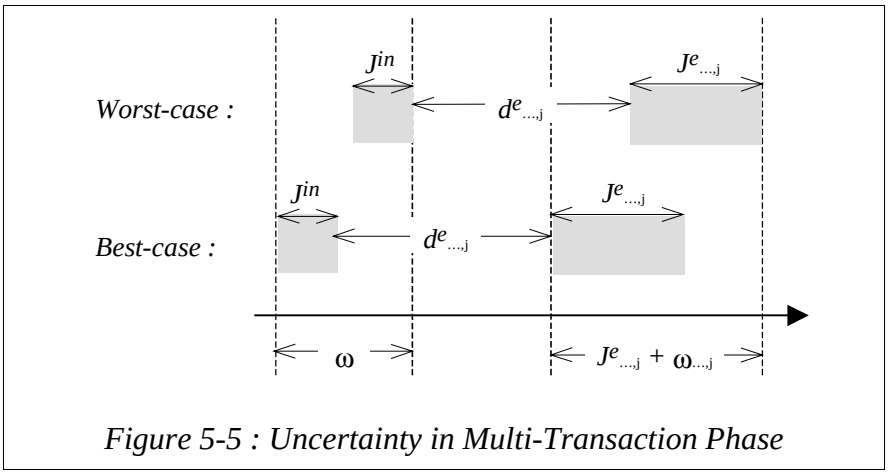


Figure 5-5 : Uncertainty in Multi-Transaction Phase

To illustrate the phase calculation across multiple transactions, consider the following set of observed events :

- $e^{in}_{1,1,2}$, $e^{in}_{1,1,3}$ and $e^{in}_{1,2,3}$ from the example transaction above;
- $e^{in}_{7,2}$ and $e^{in}_{9,1}$ from two other transactions with attributes ($d^{in}_{7,2} = 12$, $J^{in}_{7,2} = 13$ and $J^{in}_{7} = 3$) and ($d^{in}_{9,1} = 8$, $J^{in}_{9,1} = 6$ and $J^{in}_{9} = 6$), respectively.

Suppose that the phase relationship between the three transactions is specified (or derived) as $\omega = 10$. The phase relationship between the set of observed events can then be determined as follows :

Ω ($e^{in}_{1,1,2}$, $e^{in}_{1,1,3}$, $e^{in}_{1,2,3}$, $e^{in}_{7,2}$, $e^{in}_{9,1}$) = max (18, 18, 22, 25, 14) - min (5, 5, 10, 5, 4) = 21

Interpreting this result, the five observed events will all occur within a maximum time interval of 21 time units. Figure 5-6 illustrates this result and its derivation.

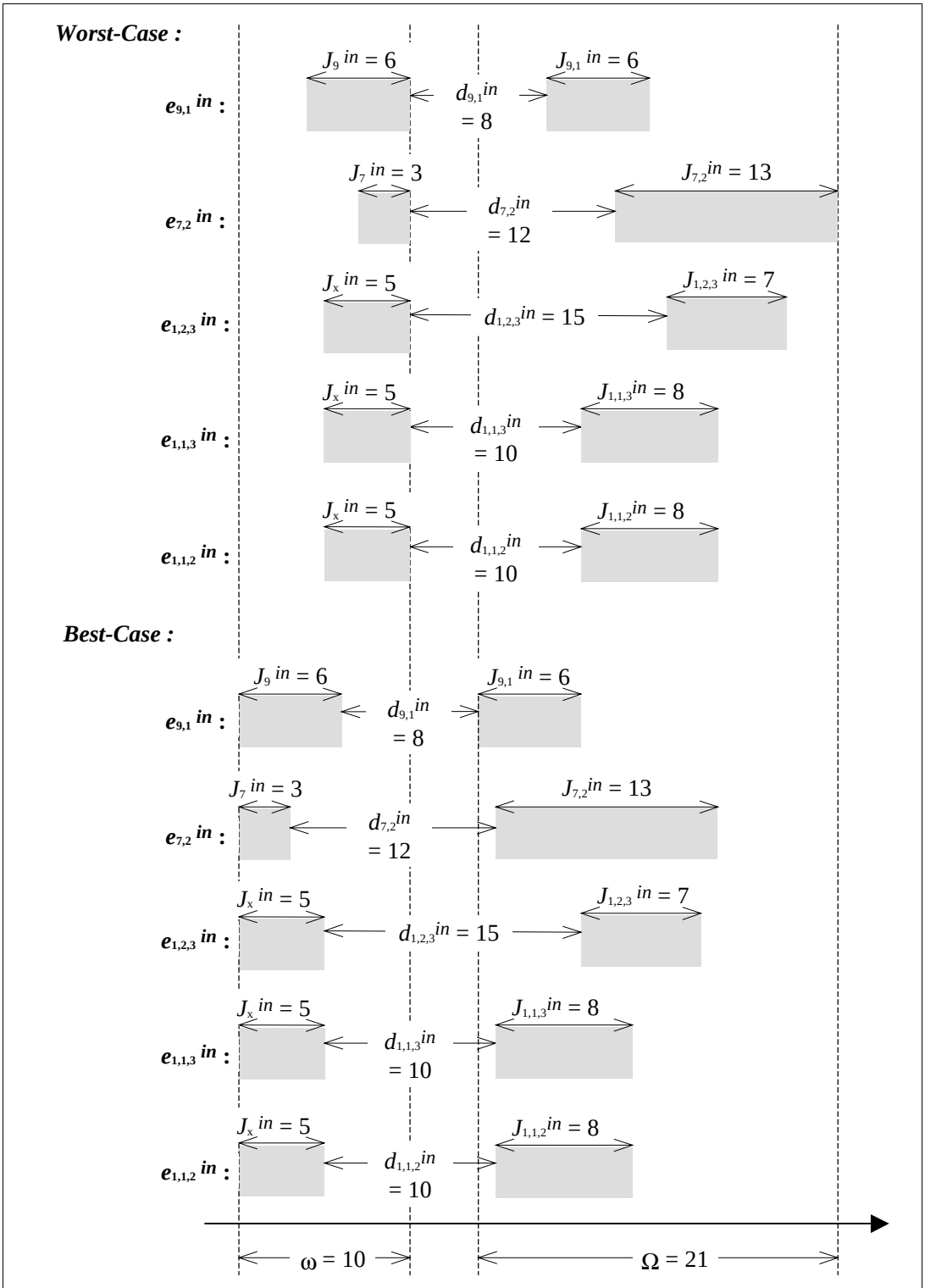


Figure 5-6 : Example Transaction - Phase Calculation for Events $e^{in}_{1,1,2}$, $e^{in}_{1,1,3}$, $e^{in}_{1,2,3}$, $e^{in}_{7,2}$ *and* $e^{in}_{9,1}$

5.2.3 Phase - A Generalisation of Jitter

Notice that phase is actually a generalisation of jitter, being the variability in time of occurrence over multiple events rather than a specific individual event. This can be seen by observing the behaviour of equations *(5-16)* and *(5-17)* for certain limiting cases.

Firstly, consider the behaviour of equation *(5-16)*, the phase calculation across multiple events within a single transaction, in the limiting case of a *single* observed event $e_{i,..,j}$. The expression can easily be seen to evaluate the worst-case jitter associated with the event as follows :

$$\Omega = (d^e_{i,..,j} + J^e_{i,..,j}) - d^e_{i,..,j} = J^e_{i,..,j}$$

Similarly, the multi-transaction phase calculation given in equation *(5-17)* can be seen to reduce to the uni-transaction phase calculation given in equation *(5-16)* when applied to a single transaction. In such a case, the transaction arrival phase ω is equivalent to the input jitter of the single transaction and so :

$$\omega_{...,j} = \omega - J^{in} = 0$$

Hence, equation *(5-17)* is reduced to equation *(5-16)* :

$$\Omega = \max_j(d^e_{...,j} + J^e_{...,j}) - \min_j(d^e_{...,j})$$

5.3 Summary

Two extensions have been described to the RBA end-to-end timing model as presented in chapter 4 :

- how to control jitter accumulation at the transaction level;
- how to determine the maximum separation in time between multiple concurrent events within the system (termed the phase relationship between the events).

Section 5.1 has extended the RBA end-to-end timing model to allow jitter accumulation to be controlled such that :

- a purely time-triggered transaction can be produced by specifying all activity input jitter values to be zero (or negative to give an offset) throughout the transaction;
- a purely event-triggered transaction can be produced by allowing all input jitter to be inherited,
- a hybrid time-and-event-triggered (referred to as *jitter-controlled*) transaction can be defined by specification of appropriate jitter tolerance levels.

In this way, the effects of jitter can be controlled throughout the system to a degree that is entirely driven by application requirements. The imposition of a time-triggered transaction model as a means of partitioning the timing model, irrespective of any application

requirements, has already been discounted by virtue of the physically partitioned RBA localised timing model.

Section 5.2 has extended the RBA end-to-end timing model to allow the phase relationship between multiple system events to be determined. The calculation can be performed within a single transaction or, provided that the transaction arrival phase is known, across multiple transactions. The notion of phase has been shown to generalise that of jitter, being the variability in time of occurrence over multiple events rather than a specific individual event.

CHAPTER 6. RBA and System Implementation

The timing analysis model described in chapters 4 and 5 is completely target-independent. The outcome is a set of *timing obligations* that represent constraints on the final implementation of the system. The implementation stage is now discussed as follows :

- the selection of the final scheduling solutions for the system (including communication schedules);
- the final verification of system timing properties, *ie.* final analytical 'proof' that the system correctly implements its timing requirements.

The selection of appropriate scheduling solutions for system CPU and communication resources is essential in order to readily support subsequent changes during system integration and thereafter during in-service upgrades. This is the key driver for the implementation scheme described below. The description is given in two stages :

- a basic cyclic scheduling scheme;
- a server-based cyclic scheduling scheme - this scheme extends the basic cyclic scheme for reasons that will be outlined.

A consistent example is used throughout this chapter - the *generic avionics platform* (GAP) case study, as described in [Locke91]. The GAP example is introduced in section 6.1 and discussed initially in the context of the RBA rate-based execution model. In section 6.4, the cyclic/server-based implementation scheme is mapped to the standard scheduling framework proposed for future Civil IMA systems as defined in the APEX interface standard [AEEC97].

6.1 Introduction to GAP Case Study

Table 6-1 presents a set of timing attributes taken from the GAP case study [Locke91]. There is no specific notion of transactions in this example and so it is appropriate to model each GAP 'task' as a single RBA 'activity' - there is no benefit in further decomposition of individual GAP tasks into multiple RBA activities. Consequently, the terms 'task' and 'activity' may be used interchangeably in the context of this example. Note that this does not limit the applicability of the GAP example at this stage. In the RBA approach, end-to-end transactions are captured and verified by the end-to-end timing analysis model, whereas the objective at this stage is to define and assess 'localised' scheduling and timing analysis methods that can implement the timing obligations inherited from that model. These obligations are expressed in the form of worst-case responsiveness and jitter requirements on individual RBA activities and, where the RBA rate-based execution model is adopted, a set of worst-case resource requirements and minimum execution rates.

In recognition of the fact that implementation schemes are 'resource-oriented', *eg.* scheduling a CPU resource, rather than 'transaction-oriented', a more convenient form of notation can be adopted at this stage (which also provides consistency with the GAP example notation). Let $\{\lambda_j;\ j = 1,..,n\}$ denote the set of activities allocated to a shared resource and denote the associated set of timing obligations by $\{(C_j, v_j, R_j);\ j = 1,..,n\}$.

j	*Function*	C_j	T_j	R_j	U_j
1	Radar Tracking Filter	2	25	25	0.08
2	RWR Contact Mgt.	5	25	25	0.2
3	Data Bus Poll Device	1	40	40	0.025
4	Weapon Aiming	3	50	50	0.06
5	Radar Target Update	5	50	50	0.1
6	Nav. Update	8	59	59	0.1355
7	Display Graphic	9	80	80	0.1125
8	Display Hook Update	2	80	80	0.025
9	Tracking Target Update	5	100	100	0.05
10	Weapon Protocol	1	(200)	200	0.005
11	Nav. Steering Cmds.	3	200	200	0.015
12	Display Sores Update	1	200	200	0.005
13	Display Keyset	1	200	200	0.005
14	Display Status Update	3	200	200	0.015
15	BET E Status Update	1	1000	1000	0.001
16	Nav. Status	1	1000	1000	0.001

Table 6-1 : GAP Example - Timing Requirements (CPU Resource)

All tasks are periodic, except for τ_{10} which is sporadic with minimum inter-arrival time a_{10}=200. Since no input jitter is specified for the periodic tasks, it is assumed that $a_j=T_j$ for these tasks. Conversely, assigning $a_{10}=T_{10}$ for the sporadic task (the value of 200 shown in brackets in the table) gives a total task set utilisation requirement of 83.5%.

The ability to implement a final scheduling solution for the GAP task set that is fully compliant with RBA, *ie.* supports independent activity level analysis, can be initially examined by applying the RBA rate-based execution model to the task set. The set of worst-case response times and CPU resource requirements are given in table 6-1 and so it is necessary to derive a set of minimum execution rates, using equation *(4-24)*, to complete the set of parameters for the rate-based model. Since $T_j=D_j$ for all activities, however, the minimum execution rate of each activity (which is also its minimum bandwidth allocation requirement) is exactly equal to its true utilisation requirement, *ie.* $\{v_j=U_j.;\ j = 1,..,16\}$. The total bandwidth reservation requirement is therefore equal to the total utilisation requirement

of the task set, *ie.* 83.5%, which is schedulable on a single processor by an exact implementation of the RBA rate-based model.

6.2 Basic Cyclic Scheduling Implementation of RBA

According to the RBA rate-based execution model, the timing obligations associated with the *worst-case* requirements of a given activity λ_j allocated to the shared resource, *eg.* a CPU, are expressed as follows :

- maximum execution time C_j;
- minimum execution rate v_j;
- maximum response time R_j.

It is assumed that best-case obligations are met by imposing jitter controls at the transaction level according to the method described in section 5.1. Also, it is assumed that resource allocation can be performed by some appropriate means, *eg.* automated techniques have been proposed in the form of simulated annealing [Tindell92] and genetic algorithms [Nicholson98]. In general, the allocation phase will be dependent to some extent on the final schedule construction method in that the acceptability of any given allocation will depend on the results of feasibility analysis. Consequently, this represents an area where further work could be performed to devise an allocation approach that can exploit the fact that the RBA model is partitioned, as is any compliant implementation scheme such as that proposed in this chapter.

The simplest form of RBA-compliant implementation scheme is a cyclic scheduling scheme characterised as follows :

- a fixed cycle time $\Delta \leq \min_j R_j$;
- a fixed time and duration of execution δ_j for each activity λ_j within each cycle;
- the restriction $R_j \leq a_j$.

Consequently, each activity will execute for exactly δ_j time units in *any* interval of time Δ, *ie.* not necessarily aligned to the cycle time of the schedule. The actual order of execution of activities within each cycle is arbitrary. Moreover, the execution time δ_j allocated to an activity within a cycle does not need to be contiguous.

It is necessary to assign an appropriate value for Δ and for each δ_j such that the timing obligations for each activity λ_j are met. The following scheme can be applied to achieved this but note that other valid assignments will normally exist for a given set of timing obligations. An example is developed alongside the description of the scheme by considering the activity λ_7 from the GAP task set :

j	*Function*	C_j	T_j	R_j	$v_j=U_j$
7	Display Graphic	9	80	80	0.1125

Firstly, 'normalise' the timing obligations according to the stated value of Δ. Define the normalised response time $R_j^{\Delta} \le R_j$ as follows :

$$R_j^{\Delta} = \left\lfloor \frac{R_j}{\Delta} \right\rfloor \Delta \qquad \textit{(6-1)}$$

For the example task, given a value of Δ = 25, the normalised response time $R_7^{\Delta} \le R_7$ is given as follows :

$$R_7^{\Delta} = \left\lfloor \frac{R_7}{\Delta} \right\rfloor \Delta = \left\lfloor \frac{80}{25} \right\rfloor 25 = 75$$

From equation *(4-24)*, define the corresponding normalised rate of execution v_j^{Δ} as follows :

$$v_j^{\Delta} = \frac{C_j}{R_j^{\Delta}} \qquad \textit{(6-2)}$$

For the example task, the normalised rate of execution v_7^{Δ} is given as follows :

$$v_7^{\Delta} = \frac{C_7}{R_7^{\Delta}} = \frac{9}{75} = 0.12$$

It can be seen from equation *(4-24)* that $v_j^{\Delta} \ge v_j$ since $R_j^{\Delta} \le R_j$. Finally, by applying equation *(4-20)*, assign δ_j the minimum value that will guarantee λ_j to meet its normalised worst-case response time requirement R_j^{Δ} (and therefore its true worst-case response time requirement R_j) :

$$\delta_j = \left\lceil v_j^{\Delta} \Delta \right\rceil \qquad \textit{(6-3)}$$

For the example task, assign δ_7 as follows :

$$\delta_7 = \left\lceil v_7^{\Delta} \Delta \right\rceil = \left\lceil \frac{9}{75} \cdot 25 \right\rceil = 3$$

Consequently, in the final cyclic schedule, each activity λ_j will be executed at a guaranteed minimum rate v_j^{δ} given as follows :

$$v_j^{\delta} = \frac{\delta_j}{\Delta} \qquad \textit{(6-4)}$$

where $v_j^{\delta} \ge v_j^{\Delta} \ge v_j$. Hence, each activity will be executed at a rate which is no less than that specified by its minimum rate requirement.

For the example task, assign v_7^δ as follows :

$$v_7^\delta = \frac{\delta_7}{\Delta} = \frac{3}{25} = 0.12$$

Hence, time is allocated in the cyclic schedule for the execution of the example task as follows :

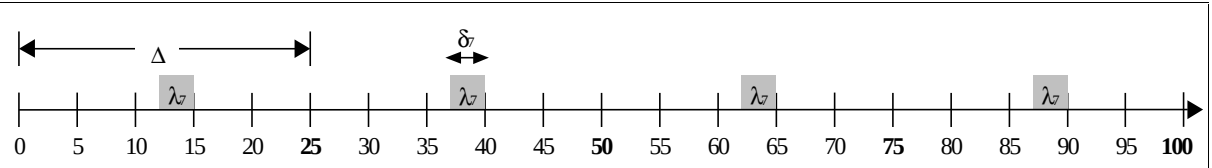


Figure 6-1 : Example Task in Cyclic Schedule

It can be seen from the diagram that the example task will always receive sufficient CPU time (3*25=75) to meet its worst-case response time requirement (80) irrespective of its own arrival time (subject to the original assumption $R_j \le a_j$) and also independently from other tasks in the task set.

More generally, the worst-case response time can be guaranteed for any target-specific resource requirement, *eg.* worst-case execution time, in the range [0, C_j]. This makes final verification for a specific target implementation very straight forward. Denoting the target-specific resource requirement of each activity λ_j by C_j^* gives the following target-specific feasibility test :

$$C_j^* \le C_j \qquad (6\text{-}5)$$

This test is independent from the choice of Δ and, for any given activity, the timing and resource requirements of other activities allocated to the shared resource. The test also allows simple re-verification of λ_j following any changes to the system that impact on the value of C_j^*, such as a change in software implementation or target hardware.

The final execution rate values for a complete set of activities allocated to a shared resource provide a basis for stating a target-independent feasibility test for the set of activities as a whole. This can be stated as follows :

$$\sum_{j=1}^{n} v_j^\delta \le 1 \qquad (6\text{-}6)$$

Since the value of v_j^δ is only dependent upon the timing obligations associated with activity λ_j, the test can be applied incrementally, *ie.* it is possible to accept or reject the addition of a 'new' activity to an existing set by comparing its final rate requirement with the remaining capacity available, independent from the actual rate requirements of activities that already

exist in the schedule (that are already guaranteed). Hence, denoting the new activity by λ_k, the following acceptance test can be applied :

$$v_k^\delta \leq 1 - \sum_{j=1}^{n} v_j^\delta \qquad (6\text{-}7)$$

Neither form of the test can be applied until the value of Δ is fixed, since this determines the value of each v_k^δ. When the value of Δ is fixed at design-time, this could be taken into account in the assignment of timing obligations at the final stage of decomposition of the end-to-end transactions. For example, the value of Δ impacts on the efficiency of the final bandwidth allocation as discussed below.

For any activity, the inefficiency of the scheme increases as the worst-case response time requirement decreases relative to the minimum inter-arrival time. This inefficiency is manifest in the final scheduling solution as an over-allocation of bandwidth compared to that which is actually required, as stipulated by the true utilisation requirement of the activity. This arises since the minimum inter-arrival time is not recognised in the construction of the cyclic schedule beyond the assumption that it is greater than the original response time requirement, *ie.* $R_j \leq a_j$. Consequently, sufficient capacity is reserved in the schedule to execute each activity once in any time interval of duration R_j (irrespective of the minimum inter-arrival time). Any over-allocated bandwidth, however, along with any that is genuinely required but unused at run-time due to variation in actual computation times, can potentially be 'reclaimed' at run-time, *eg.* to execute aperiodics or possibly even to sacrifice the ability to independently establish the timing guarantees for certain activities. Reclamation of unused bandwidth is discussed in chapter 8. Alternatively, the over-allocation of bandwidth can actually be exploited by restating the target-specific feasibility test for λ_j, as originally stated in equation *(6-5)*, to give a larger upper bound for the target-specific worst-case computation time as follows :

$$C_j^* \leq C_j^\delta \qquad (6\text{-}8)$$

where C_j^δ represents the actual guaranteed computation time allocated in the cyclic schedule over the time duration R_j, given as follows :

$$C_j^\delta = \frac{R_j^\Delta}{\Delta}\delta_j \qquad (6\text{-}9)$$

Note that the value of C_j^δ will automatically be integer since δ_j is integer and R_j^Δ is exactly divisible by Δ. This larger upper bound on the target-specific worst-case computation time can then be exploited to give a (specified) margin for error in either :

- the actual execution-time of λ_j at run-time compared to the specified value C_j such that transient over-run of the activity can be tolerated;

- the worst-case computation time of a software component procured from some third-party compared to the specified value C_j such that failure of the supplier to meet the original specification can be tolerated to a limited extent.

The final target-independent worst-case response time R_j^{δ} for λ_j, given the original worst-case computation time budget C_j and the final bandwidth allocation due to the cyclic scheduling solution, can be stated as follows :

$$R_j^{\delta} = \left\lceil \frac{C_j}{\delta_j} \right\rceil \Delta \qquad \textit{(6-10)}$$

Hence, the target-dependent worst-case response time R_j^* for λ_j, given an actual target-specific worst-case computation time value $C_j^* \leq C_j^{\delta}$ can be stated as follows :

$$R_j^* = \left\lceil \frac{C_j^*}{\delta_j} \right\rceil \Delta \qquad \textit{(6-11)}$$

Note that the scheme is exact, in the sense that the allocated bandwidth is both necessary and sufficient to meet true worst-case utilisation requirements, under certain conditions. This is the case when there are nil effects from rounding in equations *(6-1)* and *(6-3)*, the 'sufficient but not necessary' stages of the calculation. For this to occur, R_j must be exactly divisible by Δ and, in turn, C_j must be exactly divisible by that result. Clearly, in the more general case, the degree of inefficiency of the scheme is dependent upon the actual timing requirements of the activities. In exchange for a degree of inefficiency, however, the scheme provides a basis for incremental definition and modification of the final scheduling solution for the system. This is a valuable property when considered in the context of engineering large-scale, long life-time industrial real-time systems where evolvable development and upgrade methods are currently proposed as a means for containing life-cycle costs. To back up this proposition, it is explicitly recognised in the military IMA ASAAC programme that it is acceptable to sacrifice efficiency (of the overall system implementation) in order to achieve longer term system engineering benefits associated with a modular open architecture [Kemp00]. That said, an appropriate balance is clearly required in practice.

6.2.1 Example

In this section, the basic cyclic schedule construction method is applied to the GAP task set as presented in table 6-1. It has already been seen (in section 6.1) that it is possible to schedule the task set by an exact implementation of the RBA rate-based execution model.

The above approach can be applied to determine an RBA-compliant cyclic schedule as follows :

- select an appropriate value for the cycle time Δ;

- for each task λ_j :
  - determine the 'normalised' response time requirement R_j^{Δ} ;
  - determine the 'normalised' execution rate v_j^{Δ} ;
  - determine the time δ_j for which the task must be executed in each cycle Δ.

This defines the cyclic schedule. The set of final activity characteristics can then be derived as follows :

- for each task λ_j :
  - derive the guaranteed response time R_j^{δ} ;
  - derive the minimum run-time execution rate v_j^{δ} ;
  - derive the guaranteed computation time C_j^{δ} .

Observing the original constraint on the construction of the cyclic schedule, $\Delta \leq \min_j R_j$, assign Δ=25. This leads to the cyclic schedule attributes and final set of activity run-time properties given in tables 6-2, 6-3 and 6-4 and illustrated in figures 6-2, 6-3 and 6-4. Related parameters have been grouped together in the tables/figures for ease of comparison.

j	U_j	v_j	v_j^{Δ}	v_j^{δ}
1	0.0800	0.0800	0.0800	0.0800
2	0.2000	0.2000	0.2000	0.2000
3	0.0250	0.0250	0.0400	0.0400
4	0.0600	0.0600	0.0600	0.0800
5	0.1000	0.1000	0.1000	0.1200
6	0.1356	0.1356	0.1600	0.1600
7	0.1125	0.1125	0.1200	0.1200
8	0.0250	0.0250	0.0267	0.0400
9	0.0500	0.0500	0.0500	0.0800
10	0.0050	0.0050	0.0050	0.0400
11	0.0150	0.0150	0.0150	0.0400
12	0.0050	0.0050	0.0050	0.0400
13	0.0050	0.0050	0.0050	0.0400
14	0.0150	0.0150	0.0150	0.0400
15	0.0010	0.0010	0.0010	0.0400
16	0.0010	0.0010	0.0010	0.0400
Σ	0.8351	0.8351	0.8837	1.2000

Table 6-2 : GAP Example - Cyclic Schedule Implementation (Execution Rate Parameters)

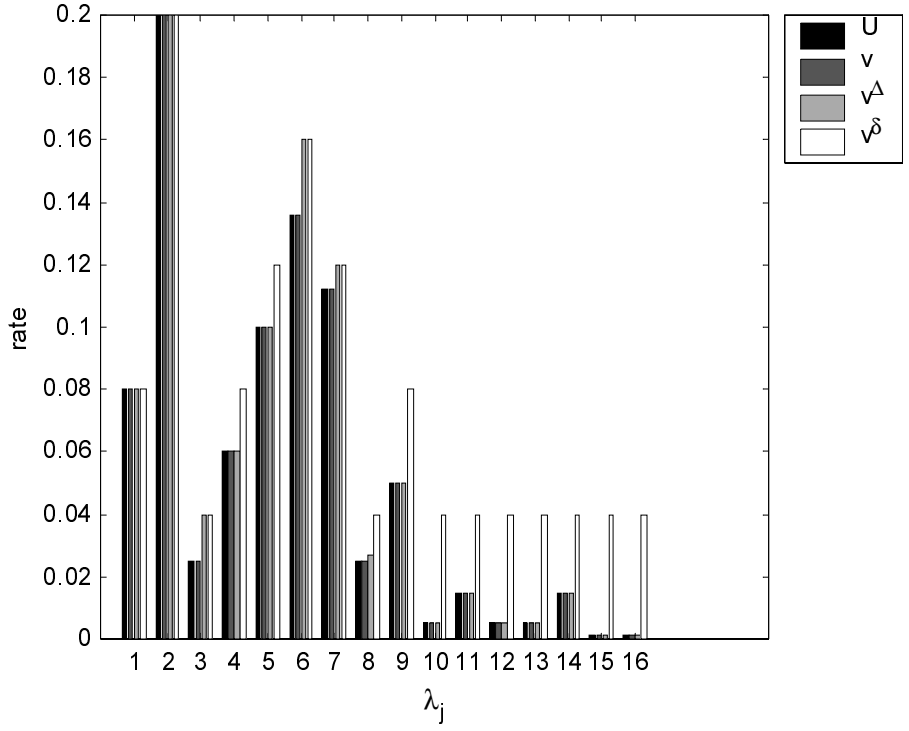


Figure 6-2 : GAP Example - Cyclic Schedule Implementation (Execution Rate Parameters)

j	R_j	R_j^{Δ}	R_j^{δ}
1	25	25	25
2	25	25	25
3	40	25	25
4	50	50	50
5	50	50	50
6	59	50	50
7	80	75	75
8	80	75	50
9	100	100	75
10	200	200	25
11	200	200	75
12	200	200	25
13	200	200	25
14	200	200	75
15	1000	1000	25
16	1000	1000	25

Table 6-3 : GAP Example - Cyclic Schedule Implementation (Response Time Parameters)

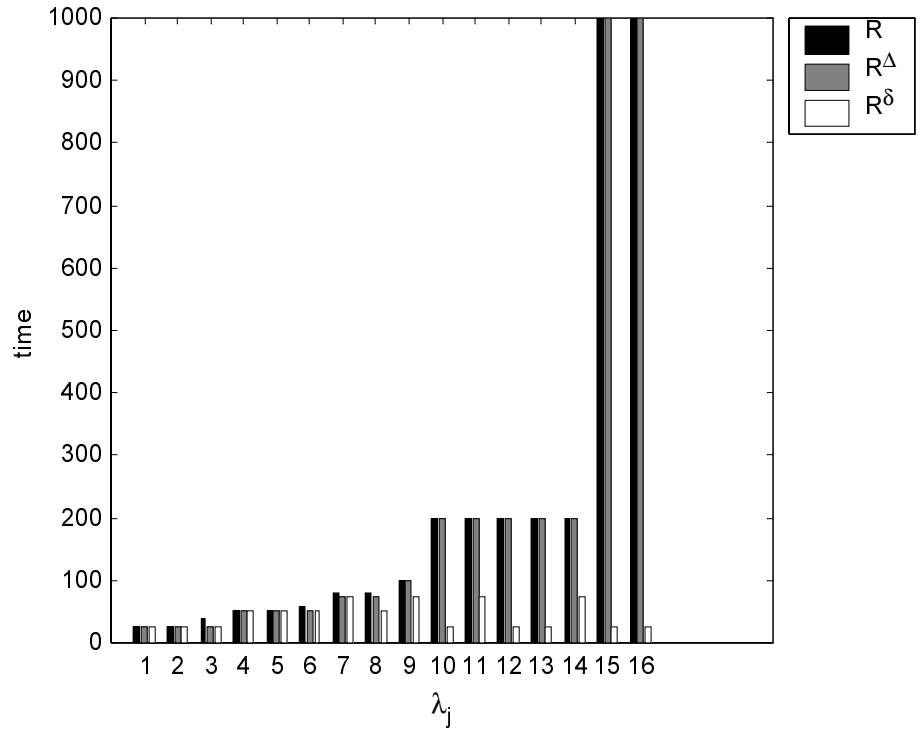


Figure 6-3 : GAP Example - Cyclic Schedule Implementation (Response Time Parameters)

j	C_j	C_j^{δ}	δ_j
1	2	2	2
2	5	5	5
3	1	1	1
4	3	4	2
5	5	6	3
6	8	8	4
7	9	9	3
8	2	3	1
9	5	8	2
10	1	8	8
11	3	8	1
12	1	8	1
13	1	8	1
14	3	8	1
15	1	40	1
16	1	40	1

Table 6-4 : GAP Example - Cyclic Schedule Implementation (Computation Time Parameters)

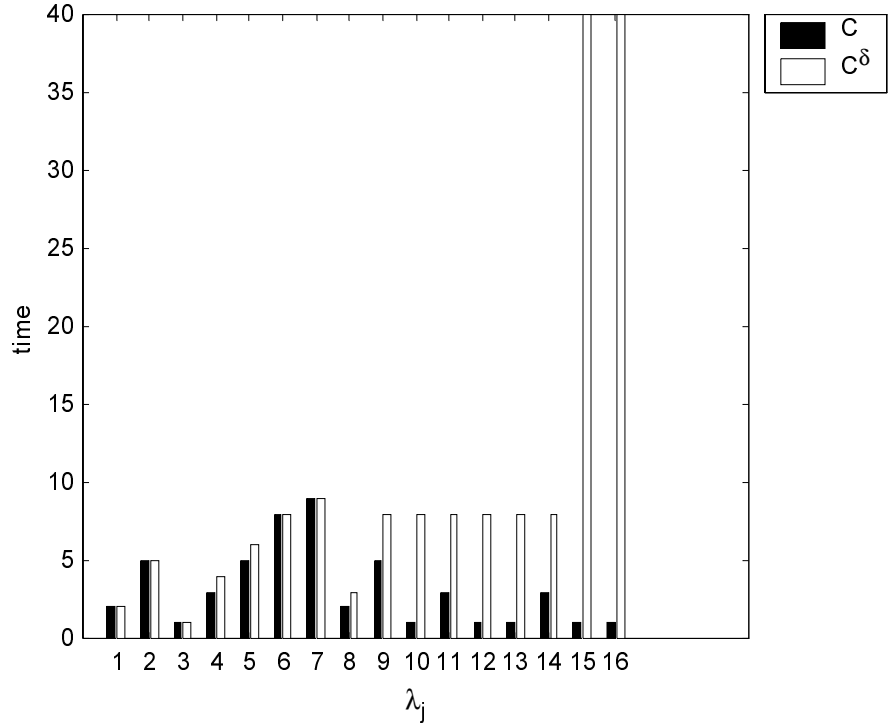


Figure 6-4 : GAP Example - Cyclic Schedule Implementation (Computation Time Parameters)

A number of observations can be made from these results. From table 6-2, the sum of the initial execution rate parameters (v_j) corresponds exactly to the total utilisation requirement of the task set (83.51%). This arises since the worst-case response time of every task is equal to its minimum inter-arrival time. After defining a cycle time of Δ=25, the sum of the rate parameters (v_j^{Δ}) corresponds to a total bandwidth allocation of 88.37%, a noticeable but reasonable increase compared to the true requirement. At the final stage of calculation, however, the need to provide integer values for δ_j prior to the calculation of the final rate parameters (v_j^{δ}) gives rise to a significant over-allocation of bandwidth due to the combination of rounding effects for the overall task set. The final bandwidth allocation is 120% and, hence, the extent of the over-allocation is sufficient to make the task set no longer schedulable on a single processor (by this scheme). The cyclic schedule has been constructed, however, to allow individual activities to be removed (or have their timing attributes changed) without affecting other activities in the schedule. Hence, it is straight forward to reduce the task set *without any further analysis* to one that is schedulable on a single processor by simply removing one or more activities (to be allocated to another processor) until the final bandwidth allocation is less than 100%. The ability to manipulate the schedule in this manner is a considerable benefit in the context of engineering larger-scale real-time systems.

A counter effect of bandwidth over-allocation is an equivalent reduction in worst-case response times (R_j^{δ}) compared to the stated response time requirements (R_j), as can be seen in table 6-3. For example, λ_{15} has a final bandwidth allocation of 4% (equivalent to its execution rate of 0.04) compared to its stated requirement of 0.1%. The corresponding reduction in its worst-case response time is apparent in the final value of 25 compared to an original requirement of 1000.

The over-allocation of bandwidth is due to the restriction that every task is executed (for a duration δ_j) in every cycle time Δ, which is reflected in the final computation time figures (C_j^{δ}) given in table 6-4. This restriction leads to a simpler (and more readily modifiable) scheduling solution but can be lifted to allow a more flexible scheme to be defined in favour of reducing the bandwidth over-allocation. Such a scheme is described and illustrated in the next section. Before that, however, two final observations can be made about the basic scheme as follows :

- the cyclic schedule does *not* compromise the true timing requirements of the task set - there is no imposition of false iteration rates for the purposes of constructing a schedule (a criticism often levelled at cyclic scheduling solutions);
- the cyclic schedule is incrementally modifiable - this is supported as follows :
    - one or more activities can be removed, *eg.* to allocate to a different processor, without affecting the remaining activities in the set - hence, it is possible to reduce

the task set *without any further analysis* to one that is schedulable on a single processor by simply removing activities until the final bandwidth allocation is less than 100%;

- new activities can be added to an existing set that has already been shown to be schedulable on a single processor, provided that the final bandwidth allocation remains less than 100% and as long as the existing value of Δ is suitable for the new activities;
- the timing attributes of existing activities can be changed independently, again provided that the final bandwidth allocation remains less than 100% and as long as the existing value of Δ is suitable for the new timing attributes.

6.3 Cyclic Server Implementation of RBA

As suggested above, it is possible to reduce the bandwidth over-allocation associated with the basic cyclic implementation scheme by relaxing the constraint that every activity must be offered the chance to execute in every scheduling cycle. This gives rise to the notion of the cyclic schedule as a number of cyclic bandwidth *servers* that can be allocated to make more effective use of the available bandwidth for a given set of activities.

The starting point is once again the selection of a schedule cycle time Δ subject to the constraint $\Delta \leq \min_j R_j$. Given that value, define the server activity $\lambda^S(\delta^S,N^S)$ as a notional activity that is allocated δ^S execution time units in every Δ cycle but does not actually consume that allocation itself. Instead, the server offers the resource to other activities so that these can execute with an effective cycle time of $N^S\Delta$. The total bandwidth of the server can then be used to execute a number of activities that individually have relatively low bandwidth requirements that would otherwise be allocated a disproportionate amount of bandwidth by the basic scheme (due to the restriction that each activity must be allocated at least one execution time unit in every Δ). Assuming that the server executes its allocated activities in a fixed order then an activity λ_j allocated to a server $\lambda^S(\delta^S,N^S)$ will execute for a duration δ_j spread over an interval of $N^S\Delta$.

The cyclic server exploits the fact that the basic scheme, and its analysis , does not require the execution time allocated to an activity within a scheduling cycle to be contiguous. The analysis associated with the cyclic server is, therefore, exactly analogous to that for the basic cyclic scheme but with Δ replaced by $N^S\Delta$. Hence, the derivation of δ_j for an activity λ_j executed via a cyclic server $\lambda^S(\delta^S,N^S)$ is given by equations *(6-1)* to *(6-3)* with Δ replaced by $N^S\Delta$. Similarly, equations *(6-4)*, *(6-9)* and *(6-10)* can be applied with Δ replaced by $N^S\Delta$ to determine the final rate, computation time and response time values, respectively. For this reason, the cyclic server method is actually a generalisation of the basic cyclic scheme described previously, where multiple cycle times are supported. These multiple cycle times need not be harmonic, only a multiple of the basic cycle time Δ.

For the general case of activity execution via a cyclic server $\lambda^S(\delta^S,N^S)$, the expression for determining the normalised response time for λ_j is adapted as follows :

$$R_j^\Delta = \left\lfloor \frac{R_j}{N^S\Delta} \right\rfloor N^S\Delta \qquad \textit{(6-12)}$$

The corresponding normalised execution rate v_j^Δ is then found as before by equation *(6-2)*. The minimum execution time δ_j per interval $N^S\Delta$ that will guarantee λ_j to meet its normalised worst-case response time requirement R_j^Δ (and therefore its true worst-case response time requirement R_j) is then given by :

$$\delta_j = \left\lceil v_j^\Delta N^S\Delta \right\rceil \qquad \textit{(6-13)}$$

Allocating δ_j execution time units per interval $N^S\Delta$ in the final cyclic schedule means that each activity λ_j will be executed at a guaranteed minimum rate v_j^δ given as follows :

$$v_j^\delta = \frac{\delta_j}{N^S\Delta} \qquad \textit{(6-14)}$$

The final total computation time C_j^δ allocated in the cyclic schedule over a time interval of duration R_j is given as follows :

$$C_j^\delta = \frac{R_j^\Delta}{N^S\Delta}\delta_j \qquad \textit{(6-15)}$$

The final target-independent worst-case response time R_j^δ for λ_j, given the original worst-case computation time budget C_j and the final bandwidth allocation due to the cyclic scheduling solution, is given as follows :

$$R_j^\delta = \left\lceil \frac{C_j}{\delta_j} \right\rceil N^S\Delta \qquad \textit{(6-16)}$$

To illustrate the cyclic server approach by example, consider activity λ_{15} from the GAP case study (which was shown in section 6.2.1 to suffer significant bandwidth over-allocation when the basic cyclic scheme is applied). Given the previous value of $\Delta = 25$, define a cyclic server $\lambda^S(1,8)$ to serve activity λ_{15} (and potentially other activities with similarly over-allocated bandwidth). The normalised response time $R_{15}^\Delta \le R_{15}$ is then determined from equation *(6-12)* as follows :

$$R_{15}^\Delta = \left\lfloor \frac{R_{15}}{8\Delta} \right\rfloor 8\Delta = \left\lfloor \frac{1000}{200} \right\rfloor 200 = 1000$$

The normalised rate of execution v_{15}^Δ is given by equation *(6-2)* as follows :

$$v_{15}^{\Delta} = \frac{C_{15}}{R_{15}^{\Delta}} = \frac{1}{1000} = 0.001$$

Equation *(6-13)* then gives δ_{15} as follows :

$$\delta_{15} = \left\lceil v_{15}^{\Delta} \cdot 8\Delta \right\rceil = \left\lceil \frac{1}{1000} \cdot 200 \right\rceil = 1$$

Consequently, in the final cyclic schedule, activity λ_{15} will be executed at a guaranteed minimum rate v_{15}^{δ} given by equation *(6-14)* as follows :

$$v_{15}^{\delta} = \frac{\delta_{15}}{8\Delta} = \frac{1}{200} = 0.005$$

Whilst this result still represents a significant over-allocation (5 times the true requirement), the result of the basic scheme was far worse (40 times the original requirement). In general, as is the case for the basic scheme, the server can provide exact bandwidth allocation for an activity if R_j is exactly divisible by $N^S\Delta$ and, in turn, C_j is exactly divisible by that result. This occurs, for example, when $N^S\Delta = R_j$ and so the use of a server $\lambda^S(1,40)$ would give an exact allocation for λ_{15}.

The utilisation-based feasibility test given in equation *(6-6)*, involving the sum of the final activity execution rates, is no longer exact but merely a necessary condition when one or more activities are executed via servers. A sufficient test can be produced by replacing the combined execution rates of the activities executed by servers by the total capacities of the corresponding servers, where the total capacity v^S of a server $\lambda^S(\delta^S,N^S)$ is given by adaptation of equation *(6-4)* as follows :

$$v^S = \frac{\delta^S}{\Delta} \qquad \textit{(6-17)}$$

A simple test for feasible allocation of server bandwidth is given as follows :

$$\sum_{k=1}^{m} v_k^{\delta} \le v^S \qquad \textit{(6-18)}$$

where v_k^{δ} denotes the final execution rate of each activity λ_k allocated to the server and m is the total number of such activities.

For a server $\lambda^S(\delta^S,N^S)$ with defined period N^S (in multiples of Δ) and set of allocated activities $\{\lambda_k;\ k = 1,..,n^S\}$, the value of δ^S can actually be derived from the set of activity execution times $\{\delta_k \text{ in } N^S\Delta;\ k = 1,..,n^S\}$ as follows :

$$\delta^S = \left\lceil \frac{\sum_{k=1}^{n^S} \delta_k}{N^S} \right\rceil \qquad \textit{(6-19)}$$

Observing that the total time required to execute non-server-based activities in the basic cyclic schedule is equivalent to that of a cyclic server $\lambda^S(\delta^S,1)$, referred to as the *base level* server, the final bandwidth requirement Ψ is given as follows :

$$\Psi = \frac{1}{\Delta}\sum_{i=1}^{nS}\delta_i^S \qquad (6\text{-}20)$$

given the set of servers $\{\lambda_i^S(\delta_i^S, N_i^S)\ ;\ i = 1,..,nS\}$ that includes the base level server. This expression is sufficient-but-not-necessary since, depending on the actual server periods and final utilisation figures, it is possible that the bandwidth requirements of a lower rate server could be completely absorbed within the spare capacity of a higher rate server. In such cases, the bandwidth requirements of a lower rate server can be effectively eliminated from the total bandwidth calculation (as illustrated later by examples in chapter 8). This possibility is not accounted for in equation *(6-20)* and leads to the potential for future work to generalise the cyclic server method to support the notion of 'nested' or 'hierarchical' servers within the RBA cyclic implementation scheme.

6.3.1 Example

This section illustrates the use of the cyclic server method to improve bandwidth allocation compared to the basic cyclic RBA implementation scheme given in section 6.2. Assuming the same basic cycle time Δ=25, define a server $\lambda^S(2,8)$ to execute the low utilisation activities $\{\lambda_{10}, \ldots, \lambda_{16}\}$. This gives the improved results (compared to the basic cyclic scheme) shown in tables 6-5, 6-6 and 6-7 and in figures 6-5, 6-6 and 6-7.

The total capacity of the server $\lambda^S(2,8)$ is given by equation *(6-17)* as follows :

$$v^S = \frac{\delta^S}{\Delta} = \frac{2}{25} = 0.08$$

This means that 92% of the total processor capacity is available for non-server-based activities $\{\lambda_1, \ldots, \lambda_9\}$ and 8% for server-based activities $\{\lambda_{10}, \ldots, \lambda_{16}\}$. So, whilst the total bandwidth allocation is more efficient than for the basic cyclic scheme - 97.5% compared to 120% - this is not sufficient to guarantee feasibility on a single processor. It is necessary to show separately that activities $\{\lambda_1, \ldots, \lambda_9\}$ can be executed within their 92% bandwidth allocation and that activities $\{\lambda_{10}, \ldots, \lambda_{16}\}$ can be executed within their 8% bandwidth allocation. From table 6-5, the combined bandwidth allocation for activities $\{\lambda_1, \ldots, \lambda_9\}$ turns out to be exactly 92% and the combined allocation for activities $\{\lambda_{10}, \ldots, \lambda_{16}\}$ is 5.5%. Hence, the complete set of activities *is* now schedulable on a single processor.

The improved efficiency of this scheduling solution is also reflected in the increased number of activities that have now been allocated the exact bandwidth to meet their stated requirements, compared to the previous example. 10 out of the 16 activities have now been allocated exact bandwidth compared to only 4 previously.

j	U_j	v_j	v_j^{Δ}	v_j^{δ}
1	0.0800	0.0800	0.0800	0.0800
2	0.2000	0.2000	0.2000	0.2000
3	0.0250	0.0250	0.0400	0.0400
4	0.0600	0.0600	0.0600	0.0800
5	0.1000	0.1000	0.1000	0.1200
6	0.1356	0.1356	0.1600	0.1600
7	0.1125	0.1125	0.1200	0.1200
8	0.0250	0.0250	0.0267	0.0400
9	0.0500	0.0500	0.0500	0.0800
10	0.0050	0.0050	0.0050	0.0050
11	0.0150	0.0150	0.0150	0.0150
12	0.0050	0.0050	0.0050	0.0050
13	0.0050	0.0050	0.0050	0.0050
14	0.0150	0.0150	0.0150	0.0150
15	0.0010	0.0010	0.0010	0.0050
16	0.0010	0.0010	0.0010	0.0050
Σ	0.8351	0.8351	0.8837	0.9750

Table 6-5 : GAP Example - Cyclic Server Implementation (Execution Rate Parameters)

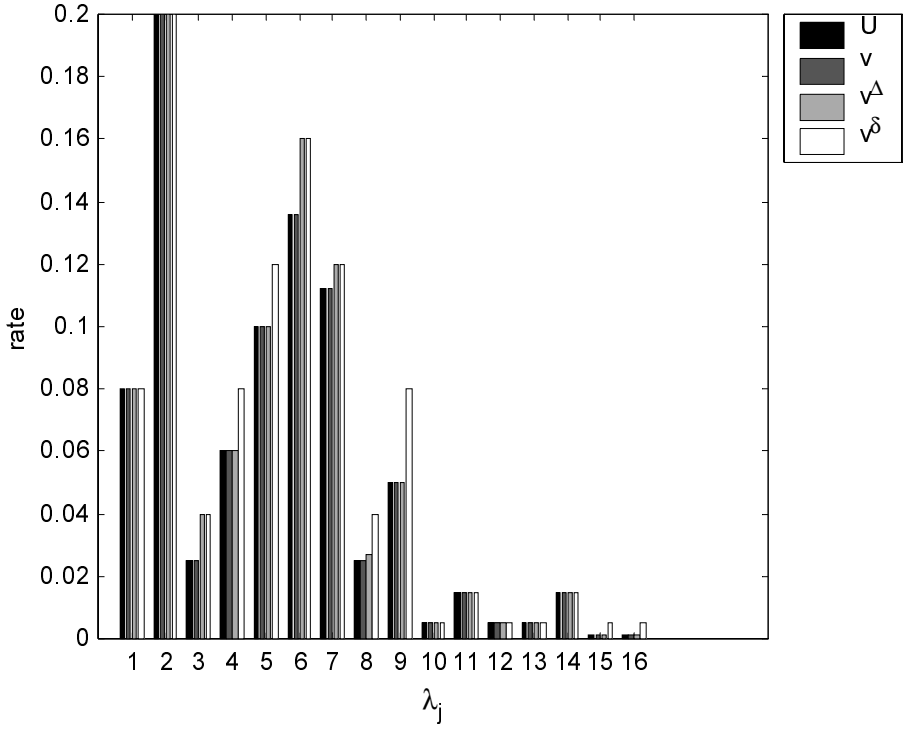


Figure 6-5 : GAP Example - Cyclic Server Implementation (Execution Rate Parameters)

j	R_j	R_j^{Δ}	R_j^{δ}
1	25	25	25
2	25	25	25
3	40	25	25
4	50	50	50
5	50	50	50
6	59	50	50
7	80	75	75
8	80	75	50
9	100	100	75
10	200	200	200
11	200	200	200
12	200	200	200
13	200	200	200
14	200	200	200
15	1000	1000	200
16	1000	1000	200

Table 6-6 : GAP Example - Cyclic Server Implementation (Response Time Parameters)

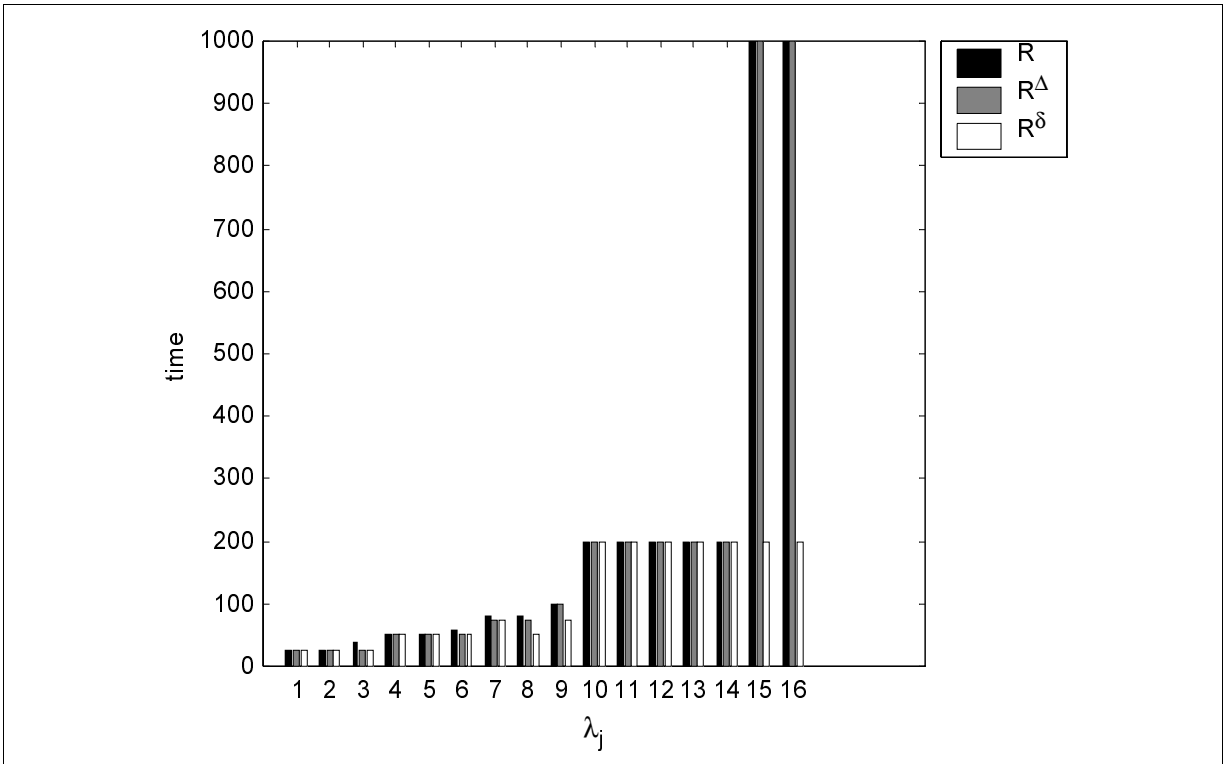


Figure 6-6 : GAP Example - Cyclic Server Implementation (Response Time Parameters)

j	C_j	C_j^{δ}	δ_j
1	2	2	2
2	5	5	5
3	1	1	1
4	3	4	2
5	5	6	3
6	8	8	4
7	9	9	3
8	2	3	1
9	5	8	2
10	1	1	1
11	3	3	3
12	1	1	1
13	1	1	1
14	3	3	3
15	1	5	1
16	1	5	1

Table 6-7 : GAP Example - Cyclic Server Implementation (Computation Time Parameters)

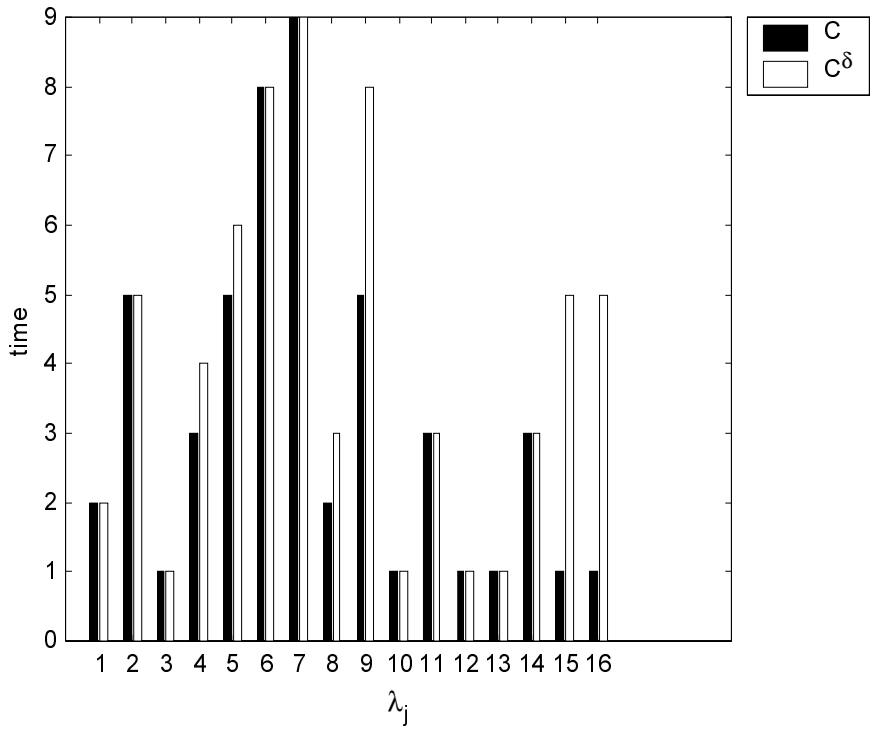


Figure 6-7 : GAP Example - Cyclic Server Implementation (Computation Time Parameters)

6.4 Implementation of RBA for Civil IMA APEX Applications

Throughout the thesis, *integrated modular avionics* (IMA) [AEEC91, Multido98] has been cited as a prime target application for the RBA approach. It is therefore appropriate to consider the construction of an RBA-compliant implementation scheme for IMA systems. The military variant of IMA [Multido98] does not prescribe any particular scheduling policy, only that it should support the overall objectives of IMA in terms of evolvable system development and upgrade. The civil variant of IMA [AEEC91], however, does prescribe a specific run-time CPU scheduling scheme as part of the application programming interface standard, known as the APEX interface [AEEC97]. In this section, the APEX scheduling model is summarised and a mapping from the RBA model to the APEX model is given.

APEX defines the following two-tier scheduling model :

- *partition scheduling* - CPU time on each processor is allocated according to a fixed cyclic schedule to create multiple temporal 'partitions'; a given partition may execute multiple times within the cycle time of the schedule); whilst executing, a partition is granted exclusive access to all resources shared with other partitions;
- *process scheduling* - within each partition of the cyclic schedule, multiple concurrent processes can compete for execution according to a static priority-based regime.

Communication between partitions is handled via operating system calls, giving a degree of independence from the underlying network topology and actual allocation of partitions to processors. Within a partition, additional 'logical' communication resources may be shared between concurrent processes. Where this is required, mutual exclusion can be enforced by the priority-based process level scheduler via some form of priority inheritance scheme (as reviewed in section 2.4.3.1).

The main motivation for prescribing a cyclic partition level scheduling model is to support temporal isolation/partitioning between safety-related applications and other less critical applications in an integrated computing environment. Historically, segregation between such applications has invariably been enforced by allocation to separate physical computing platforms - the so-called 'federated' approach. APEX leaves the assignment of partition (and process) level timing attributes to the system/application engineers, *ie.* places no restrictions, nor offers any advice. The cyclic RBA implementation scheme described in the sections above thus provides an appropriate mechanism for assigning APEX partition level scheduling parameters in a manner that supports independent response time analysis of APEX applications.

The first step is to assume temporal equivalence between an RBA activity and an APEX partition. Thus, the APEX partition level schedule is characterised as follows for a set of APEX partitions $\{\lambda_j; j = 1,..,n\}$:

- a fixed cycle time, Δ;
- a maximum duration of execution δ_j (within each cycle) for each λ_j.

where an appropriate value for the cycle time Δ and each δ_j can be derived by the RBA cyclic implementation scheme such that the timing requirements of each APEX partition λ_j are met (or shown to be infeasible on a single processor and consequently re-allocated). Final verification of the target-independent RBA timing analysis and the resulting APEX cyclic schedule can be performed trivially by ensuring that the target-specific computation time of each APEX partition is no greater than the specified target-independent bound C_j. There is no need to perform any further timing analysis to verify worst-case end-to-end delay and jitter bounds - these can be inferred directly from the RBA end-to-end timing analysis model.

By further analogy to the general cyclic RBA implementation scheme, the APEX schedule also allows simple and independent re-verification of changes to the IMA system that impact on partition level timing requirements or target-specific resource requirements. For example, changes to the software implementation or target hardware will invariably lead to different target-specific computation times - these can be checking independently for each affected partition against the corresponding target-independent bound. This supports the evolvable development and upgrade objective of IMA.

This mapping only supports APEX process level response time analysis when there is a single process executing in each APEX partition. For the case of multiple processes executing in a partition, a response time analysis has previously been documented in [Audsley96]. For consistency with the previous results described in this chapter, this analysis can be stated using RBA notation as follows :

$$R_{j,k} = C_{j,k} + B_{j,k} + M_{j,k} + \left\lceil \frac{\max(0, R_{j,k} - \delta_j)}{N^S \Delta} \right\rceil (N^S \Delta - \delta_j) + \sum_{i=1}^{k-1} \left\lceil \frac{R_{j,k} + J^{in}_{j,i}}{T_{j,i}} \right\rceil C_{j,i} \quad \textit{(6-21)}$$

where $R_{j,k}$ is the worst-case response time of the process $\tau_{j,k}$ within the APEX partition λ_j, $C_{j,k}$ its worst-case computation time, $B_{j,k}$ its worst-case blocking time (due to process level synchronous communication), $M_{j,k}$ its total worst-case delay due to operating system and other overheads (as detailed in the paper), ($T_{j,i}$, $J^{in}_{j,i}$, $C_{j,i}$) the period, worst-case input/release jitter and computation time of each higher priority process $\tau_{j,i}$ in the same partition as $\tau_{j,k}$ and ($N^S\Delta$, δ_j) the period and execution time of the partition λ_j.

It is not obvious how to derive the partition level scheduling attributes from a given set of process level timing attributes when there is more than one process per partition. Also, final verification and subsequent changes may or may not be performed independently at the process level, depending on :

- interference between processes in the same partition - changes affecting a higher priority process may affect the feasibility of all lower priority processes in that partition, *eg.* an increase in the worst-case computation time of a higher priority process will lead to an increase in the response time of all lower priority processes (whereas in the single process per partition case, such changes merely lead to a reduction in slack time in the partition level cyclic schedule);
- precedence relationships - changes affecting the behaviour of one process may impact on the feasibility of others in the same partition, where precedence relationships exist, via worst-case input jitter; similarly, changes may impact on the feasibility of processes in other partitions where precedence relationships exist between these.

A concern is identified in [Audsley96] in that the period of the partition to which a given process is assigned could make an intolerably large contribution to the worst-case response time of that process through input jitter. It is further suggested that the problem can be somewhat reduced by adjusting process periods to be in line with partition periods and/or allocating processes with similar periods to the same partition. This does not, however, tackle the root of the problem which is that the input jitter will dominate the analysis for any process with a short deadline compared to the period of its allocated partition. This problem does not occur when the APEX partition level attributes are assigned via the RBA cyclic implementation scheme since this will give rise to partition periods that are short compared to the deadlines of the processes they serve, with only a single process per partition. The inter-partition communication mechanism is then responsible for ensuring that communication delays are within required bounds. Note that the manner in which inter-partition communication occurs is not explicitly specified in the APEX standard and is dependent upon the implementation of the underlying operating system.

In conclusion, a fully-compliant RBA implementation scheme can be constructed by reducing the APEX scheduling model to one that supports a single process per partition. For this implementation scheme, final verification can be performed independently per APEX partition/process by a simple comparison between the target-independent and target-dependent computation times.

The ARINC 629 communication protocol that is specified for Civil IMA systems can also be configured to implement a static cyclic schedule [Grigg96]. Consequently, the RBA cyclic implementation scheme can be directly mapped to this protocol to support derivation of the 629 databus schedule directly from the RBA abstract model.

6.5 Summary

The cyclic scheduling scheme described in this chapter provides a sufficient implementation of the RBA rate-based execution model proposed in section 4.3. The scheme allows :

- the run-time scheduling solution to be derived directly from an RBA target-independent model without compromising the true timing requirements;
- the response time analysis to remain partitioned at the target-dependent stage;
- final verification (and re-analysis after changes) to be performed on a per-activity basis by simple comparison of target-dependent and target-independent resource requirements.
- target-specific end-to-end response times to be verified automatically by inference from the RBA end-to-end timing model, without the need to perform target-specific end-to-end timing analysis.

The proposed RBA cyclic implementation scheme has been shown to map readily to a reduced form of the APEX scheduling model [AEEC96] that been proposed for future Civil IMA systems. Since the ARINC 629 communication protocol can be configured to implement a cyclic schedule, this also supports a straight-forward mapping from the RBA cyclic implementation scheme.

A number of possible future work areas have been identified as follows :

- consider the development of a bandwidth-based scheme for performing the allocation stage of the process (the mapping of concurrent RBA activities to physical resources) that exploits the partitioned nature of the RBA end-to-end analysis and the target-independent nature of the rate-based execution model;
- extend the RBA cyclic implementation scheme to support ‘nested’ or ‘hierarchical’ bandwidth servers as a means of making more efficient use of the available bandwidth (reducing the extent of bandwidth over-allocation);
- further investigate the mapping of the RBA cyclic implementation scheme to the ARINC 629 protocol.

CHAPTER 7. Comparative Assessment

This chapter considers the effectiveness of the RBA cyclic implementation scheme (as described in chapter 6) in comparison to a pure static priority-based implementation subject to different forms of timing analysis (as described in section 7.1). The comparative assessment has been performed by formulation of the various timing analysis models in MATLAB and subjecting these to a series of randomly generated test-sets. Each test-set is representative of a distributed real-time system - a set of distributed transactions that are decomposed into component activities and allocated to shared resources using a simple utilisation-based method. Each of the response time analysis models is then applied to determine the set of localised delays on each shared resource. From these results, the worst-case end-to-end transaction delays are determined.

The following characteristics of the implementation schemes and associated timing analysis models are the focus of the assessment :

- responsiveness - the relative duration of the worst-case response times predicted for each implementation scheme by each associated timing model;
- bandwidth utilisation (for the RBA implementation scheme only) - the effectiveness of the final bandwidth allocation relative to the true utilisation requirements;
- change sensitivity - the extent to which changes in timing requirements and target implementation details impact on responsiveness and bandwidth allocation.

The assessment is performed in three stages as follows :

- stage 1 - responsiveness and bandwidth utilisation for time-triggered transactions;
- stage 2 - responsiveness and bandwidth utilisation for event-triggered (jitter-controlled) transactions;
- stage 3 - change sensitivity for time-triggered and event-triggered (jitter-controlled) transactions.

At each stage, a single test-set example is given before the main assessment involving multiple test-sets. This serves to illustrate the test-set generation method and gives some initial indications of results.

7.1 Timing Analysis Models Under Assessment

This section describes the forms of timing analysis (and associated implementation schemes) used in the assessment. Firstly, the RBA cyclic server implementation scheme is used as described in section 6.3. A simple cyclic server assignment method has been encoded whereby low utilisation activities can be executed via a server with cycle time at most 8 times longer than the cycle time of the base level server. This method will only

contribute, in terms of reducing the bandwidth over-allocation, when there are multiple low utilisation activities allocated to the same resource.

Three different forms of timing analysis have been encoded for the static priority-based implementation. The first form of analysis is ‘exact’, in the sense that it will provide the tightest possible upper bound on the response time value provided that the assumptions upon which it is based are valid in the implementation. This form of analysis is as given previously in section 2.4 by equation *(2-7)*, restated for the purposes of this assessment in a notation that is consistent with that used for the RBA implementation scheme :

***FP-Exact* :**

$$R_j = C_j + B_j + \sum_{\lambda_k \in hp(\lambda_i)} \left\lceil \frac{R_j + J_k}{T_k} \right\rceil C_k \qquad (2\text{-}7a)$$

where the function $hp(\lambda_j)$ returns the set of activities that are strictly higher priority than λ_j.

Various non-recursive forms of response time analysis can be derived at the expense of introducing pessimism into the predicted delays. These models are termed 'sufficient but not necessary' (abbreviated to SbNN hereafter) based on the following reasoning :

- the existence of a worst-case response time value for a given λ_j that is pessimistic but still less than its designated deadline is *sufficient* to guarantee feasibility;
- it is not *necessary* for λ_j to be designated feasible by the test for λ_j to actually be feasibility - whilst the predicted worst-case response time could be greater than the deadline, the true worst-case response time could be less than the deadline (a more accurate test could therefore give the correct result);

The benefit of using SbNN forms of analysis is in their relative simplicity. This serves to illustrate a trade-off between the simplicity and accuracy of an analytical model. Two different forms of SbNN analysis have been considered. The first of these is a newly derived test as stated in equation *(7-1)* (see Appendix A2 for derivation) :

***FP-SbNN1* :**

$$R_j = \left\lceil \frac{B_j + \sum_{\lambda_k \in hpeq(\lambda_j)} C_k}{1 - \sum_{\lambda_k \in hp(\lambda_j)} U_k} \right\rceil \qquad (7\text{-}1)$$

where the function $hpeq(\lambda_j)$ returns the set of activities that are higher or equal priority to λ_j. This test was developed in an attempt to desensitise the response time analysis for the static priority-based implementation by linking the response time of each activity λ_j to the *combined* utilisation and computation times of the set of higher priority activities.

The second SbNN test, as stated in equation *(7-2)*, is taken directly from Audsley’s thesis [Audsley93b] :

***FP-SbNN2* :**

$$R_j = C_j + B_j + \sum_{\lambda_k \in hp(\lambda_j)} \left\lceil \frac{D_j}{T_k} \right\rceil C_k \qquad (7\text{-}2)$$

Note that the two SbNN forms of analysis make no allowance for release jitter and so these tests can only be applied during the assessment in the context of time-triggered transactions - this is the first stage of the assessment. When release jitter is introduced, *ie.* transactions can become event-triggered, the *RBA-Cyclic* scheme can still be compared against *FP-Exact* but with the jitter restricted, as explained in section 7.3.

Each of the tests stated for the static priority-based implementation is independent from the method of priority assignment, *ie.* the tests are applicable across all possible priority assignments (rate monotonic, deadline monotonic or any other). The actual response time values calculated by each test, however, *are* dependent on the results of priority assignment - a change of priority assignment will change the resulting response times. Deadline monotonic priority assignment has been used throughout this assessment.

7.2 Assessment Stage 1 : Time-Triggered Transactions - Responsiveness and Bandwidth Utilisation

This section describes the first stage of assessment in which all of the forms of analysis described above are applied in the context of time-triggered transactions. In section 7.2.2, the results for multiple, randomly generated test-sets are given. Before that, in section 7.2.1, the method by which these test-sets are generated is described, with some basic observations about the different timing analysis and implementation models based on a single test-set.

7.2.1 Test-Set Generation and Example Results

Each test-set is intended to be representative of a distributed real-time system - a set of distributed transactions that are decomposed into component activities and allocated to a set of shared resources. The following procedure is applied :

- Generate a set of activity utilisation requirements where each utilisation value U_j is a positive integer randomly sampled from a uniform distribution with a specified upper bound, *eg.* 20%. This limit can be varied between each series of test-runs in order to examine the impact of this parameter on the results. A total of *n* activities are defined in each test-set, where *n* is sampled from a random uniform distribution defined over a specified range, *eg.* [10, 500].
- Perform a simple, utilisation-based mapping of activities to resources (without any preconceived notion of a resource architecture), observing a specified maximum total utilisation bound per resource, *eg.* 80%. This limit can also be varied between each series of test-runs in order to examine the impact of this parameter on results.

- Generate a set of precedence relationships between the mapped activities using a probabilistic approach. An inter-connection probability is specified for the task-set as a whole and this is used as a threshold value for a random variable sampled from a uniform distribution defined over the range [0, 1] for each pair of activities within a specified 'neighbourhood' - a range of '*j*' values. This range is typically set to a maximum of a tenth of the total number of activities in the test set and so an activity λ_j can potentially become a predecessor to any other in the set $\{\lambda_{j+1}, \lambda_{\min(j+n/10,\ n)}\}$. If the sampled value for a given pair of activities is greater than the threshold probability than a precedence constraint is introduced. Precedence constraints always run from lower to higher numbered activities - this guarantees the absence of cyclic constraints within the test-set.
- Identify the individual transactions contained in this set of mapped and inter-connected activities, given that any two activities that are inter-connected (whether directly or indirectly via other activities and precedence constraints) belong to the same transaction. Assign all activities belonging to the same transaction a common period T_j that is an integer value sampled from a random uniform distribution with specified range, *eg.* [4, 1000]. Similarly, assign different T_j values to the remaining (independent) activities.
- Given the set of utilisation requirements and execution periods, derive the remaining timing attributes. For each activity, a worst-case computation time C_j is calculated as the (integer) product of its utilisation and period. The U_j value must then be re-calculated from C_j and T_j in order to make the timing attributes consistent. Similarly, the resource utilisation values must be updated accordingly. Each activity deadline D_j is then set to its period T_j (note that this restriction is relaxed to allow $D_j \leq T_j$ in assessment stage two).

Table 7-1 defines a test-set that has been generated by the above procedure with a maximum activity utilisation value of 20% and a maximum resource utilisation (prior to transaction definition) of 80%. The columns labelled 'trans.' and 'resource' give the transaction and resource identifiers, respectively, to which activity λ_j is assigned. A value of zero in the 'transaction' column means that λ_j is not part of any transaction.

j	*trans.*	*resource*	U_j *(%)*	C_j	T_j
1	1	1	5	34	720
2	0	2	13	61	495
3	2	2	10	63	644
4	0	3	18	84	470
5	0	4	16	126	822
6	1	5	10	66	720
7	3	6	1	2	438
8	0	6	17	97	590
9	0	7	9	70	778
10	0	8	13	30	240
11	4	9	16	136	854
12	5	10	19	108	585
13	0	11	15	76	512
14	3	12	4	16	438
15	4	12	9	70	854
16	6	13	20	29	150
17	0	1	19	168	916
18	0	2	9	46	560
19	5	2	18	105	585
20	3	3	2	6	438
21	0	4	8	41	571
22	6	5	17	25	150
23	7	6	1	1	410
24	6	6	4	5	150
25	3	7	5	18	438
26	4	8	4	34	854
27	8	9	13	116	960
28	7	10	6	23	410
29	2	11	5	26	644
30	2	12	1	2	644
31	0	12	16	95	631
32	9	13	9	88	986
33	0	1	20	12	60
34	8	2	10	90	960
35	0	3	9	83	983
36	4	4	17	145	854
37	0	4	14	2	15
38	0	5	5	23	556
39	0	6	14	45	328
40	10	7	18	8	46
41	0	7	1	3	618

42	0	8	14	94	684
43	0	9	8	52	681
44	0	10	17	53	316
45	11	11	11	12	115
46	12	12	15	53	373
47	0	13	9	28	324
48	0	14	7	50	809
49	7	1	4	16	410
50	0	2	4	20	506
51	11	3	14	16	115
52	0	4	7	26	429
53	0	4	11	27	248
54	13	5	5	3	68
55	10	6	16	7	46
56	0	7	8	70	924
57	14	8	18	99	573
58	0	9	18	89	521
59	9	9	12	118	986
60	12	10	11	38	373
61	15	11	19	175	972
62	16	12	17	26	157
63	12	13	14	49	373
64	0	14	17	72	438
65	17	1	14	61	457
66	18	2	8	2	27
67	11	3	7	7	115
68	15	4	7	67	972
69	13	5	12	8	68
70	0	5	17	4	24
71	18	6	8	2	27
72	0	7	17	94	558
73	9	8	12	113	986
74	9	9	8	74	986
75	9	10	15	139	986
76	19	10	13	4	32
77	0	11	9	64	714
78	0	12	15	11	74
79	14	13	13	72	573
80	11	14	17	19	115
81	11	1	20	23	115
82	19	2	13	4	32
83	0	3	18	141	797

84	15	4	4	34	972
85	0	5	20	168	857
86	0	6	6	35	633
87	0	6	6	5	84
88	0	7	18	41	233
89	0	8	15	94	634
90	0	9	3	6	201
91	0	10	1	1	310
92	20	11	18	174	972
93	16	11	5	7	157
94	20	12	7	59	972
95	17	13	14	61	457
96	0	14	6	20	343

Table 7-1 : Example Test-Set - Timing Requirements

Figure 7-1 depicts the set of precedence constraints (transactions) generated.

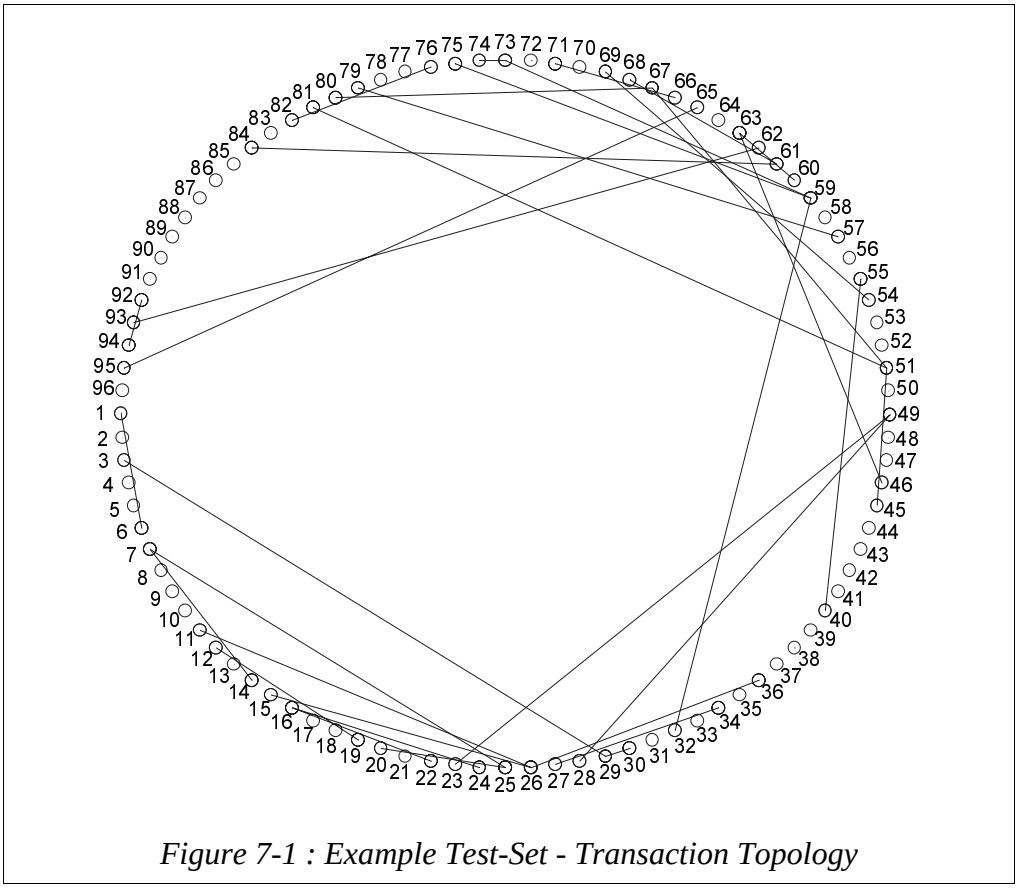


Figure 7-1 : Example Test-Set - Transaction Topology

Following the allocation of activities to resources, as indicated in table 7-2, the set of direct inter-connections required at the resource level to support the above transaction topology are depicted in figure 7-2.

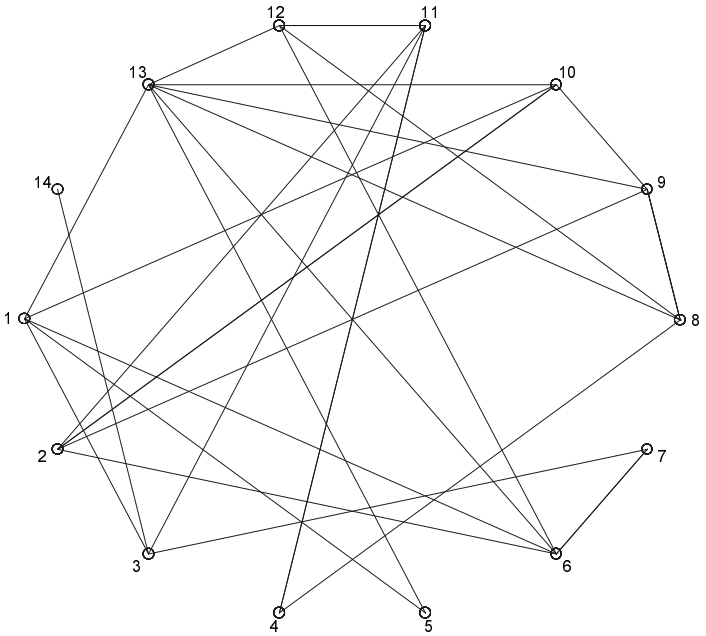


Figure 7-2 : Example Test-Set - Resource Topology

Applying each of the timing analysis models to the set of activities allocated to each shared resource gives the worst-case (localised) response times shown in table 7-2, where a ✗ indicates that the activity is declared infeasible by the corresponding test.

resource	λ_j	$R_j^{RBA-Cyclic}$	$R_j^{FP-Exact}$	$R_j^{FP-SbNN1}$		$R_j^{FP-SbNN2}$	
1	1	684	205	342		493	
	17	912	626	826		843	
	33	57	12	12		12	
	49	342	51	85		192	
	65	456	159	200		281	
	81	114	35	44		47	
2	2	432	79	84		163	
	18	432	159	209		322	
	34	810	485	1403	✗	872	
	50	270	105	129		244	
	66	27	2	2		2	
	82	27	6	7		8	
	3	567	371	800	✗	659	✗
	19	567	294	429		479	
3	35	904	406	783		842	
	51	113	16	16		16	
	67	113	23	27		23	
	83	791	300	419		482	
	4	452	113	144		211	
	20	339	29	37		98	

4	36	637	484	778		753	
	52	338	63	73		138	
	68	871	639	1436	✗	1007	✗
	84	884	679	2006	✗	1041	✗
	5	819	285	355		478	
	21	533	110	138		252	
	37	13	2	2		2	
	53	182	33	34		61	
5	69	60	12	15		20	
	85	840	537	799		783	
	6	660	223	279		478	
	22	140	44	60		86	
	38	460	86	125		318	
	54	60	15	21		23	
	70	20	4	4		4	
6	86	414	274	535		536	
	7	368	78	124		243	
	23	184	76	120		226	
	39	276	75	94		162	
	55	46	9	10		11	
	71	23	2	2		2	
	87	69	14	19		27	
	8	575	223	306		388	
	24	115	19	27		55	
7	40	46	8	8		8	
	56	828	409	880		790	
	72	552	193	265		357	
	88	230	57	60		89	
	9	644	331	538		600	
	25	414	75	104		180	
	41	552	196	373		462	
8	57	480	129	148		189	
	73	960	524	1232	✗	905	
	89	600	223	318		382	
	10	240	30	30		30	
	26	840	381	843		728	
	42	600	347	573		570	
9	58	480	95	98		107	
	74	960	485	1069	✗	890	
	90	160	6	6		6	
	11	800	289	392		448	
	27	960	411	708		700	
	43	640	147	184		254	
	59	960	750	1607	✗	1008	✗

10	75	896	549	1013	✗	878
	91	256	5	6		41
	12	576	255	416		414
	28	384	135	198		259
	44	288	62	67		95
	60	320	108	137		194
	76	32	4	4		4
11	92	920	409	627		663
	13	460	95	112		164
	29	575	133	173		285
	45	115	12	12		12
	61	920	819	1357	✗	838
	77	690	204	280		387
	93	115	19	22		31
12	14	438	117	195		266
	30	584	262	570		559
	46	365	101	132		197
	62	146	37	44		59
	78	73	11	11		11
	94	876	556	1223	✗	936
	15	730	369	773		743
	31	584	260	397		462
13	47	286	57	71		115
	63	286	106	148		192
	79	572	268	525		464
	95	429	196	284		331
	16	143	29	29		29
	32	858	552	992	✗	877
14	48	763	180	264		406
	64	436	111	143		188
	80	109	19	19		19
	96	327	39	47		77

Table 7-2 : Example Test-Set - Allocation and Localised Response Time Analysis

The following observations can be made regarding the performance of the schemes :

- all activities are declared feasible by the *RBA-Cyclic* and *FP-Exact* schemes but a small number are given infeasible by each of the SbNN tests (although the number of activities declared infeasible by *FP-SbNN2* is less than that for *FP-SbNN1*);
- all *FP-Exact* worst-case response times and most (though not all) SbNN results are noticeably better (shorter) than those for *RBA-Cyclic* - this is not surprising since the priority-based implementation is work-conserving whereas the RBA scheme is not;
- for all of the priority-based tests, the response time of the highest priority activity allocated to each resource is equal to its computation time.

The behaviour of the *RBA-Cyclic* scheme can further be characterised by the degree of bandwidth over-allocation observed. For each resource in the test-set, table 7-3 gives the total required utilisation figures, *ie.* the true resource utilisation requirement for the set of activities allocated, followed by the final allocation figures observed for the RBA cyclic implementation scheme. The final column in the table gives the resulting percentage over-allocation. Note that all figures have been rounded up.

resource	*required % utilisation*	*allocated % utilisation*	*% over-allocation*
1	81	86	8
2	82	87	19
3	66	70	6
4	81	100	25
5	83	95	16
6	69	87	28
7	73	81	11
8	74	82	11
9	76	82	9
10	78	88	13
11	79	87	10
12	79	87	10
13	76	86	13
14	45	48	7

Table 7-3 : Example Test-Set - Resource Utilisation (Required v Allocated)

These bandwidth over-allocation figures are of similar magnitude to that obtained for the GAP task-set example in chapter 6. Further observations can be made by looking more closely at the localised response time analysis results for one particular resource. Choosing resource 6, purely because it has the most activities allocated, the results of the *RBA-Cyclic* implementation scheme are given in table 7-4.

λ_j	R_j^{Δ}	δ_j	N_j^S	U_j	v_j^{δ}
86	621	2	1	0.0553	0.0870
7	437	1	8	0.0046	0.0435
23	391	1	8	0.0024	0.0435
39	322	4	1	0.1372	0.1739
55	46	4	1	0.1522	0.1739
71	23	2	1	0.0741	0.0870
87	69	2	1	0.0595	0.0870
8	575	4	1	0.1644	0.1739
24	138	1	1	0.0333	0.0435

Table 7-4 : Example Test-Set - RBA-Cyclic Results for Resource 6

These figures are based on a cycle time of $\Delta = 23$. The total required utilisation is 0.6830 (68.3%) and the final allocated utilisation is 0.8696 (86.96%). The final bandwidth allocation figure includes a 'reclaimed' utilisation of 0.0435 (4.35%) due to the fact that activities λ_7 and λ_{23} are executed via a single cyclic server $\lambda^S(1,8)$.

Cumulative response times (and associated deadlines) for the test-set are given in table 7-5. These figures have been calculated according to the same method used in the RBA end-to-end timing analysis (as described in section 4.2). In general, the worst-case cumulative response time of an activity is equal to its own worst-case response time plus the worst-case combined response time of its predecessors. The combined predecessor response time is dependent on the transaction topology, *eg.* must account for one-to-many and many-to-one precedence constraints. Clearly, where an activity has previously been declared infeasible by the localised response time analysis, indicated by a '✗' in the cumulative response time column, this affects the feasibility of any transactions in which the activity is involved. For a transaction to be feasible, all of its component activities must be feasible at the localised analysis level.

trans.	λ_j	CD_j	$CR_j^{RBA-Cyclic}$	$CR_j^{FP-Exact}$	$CR_j^{FP-SbNN1}$		$CR_j^{FP-SbNN2}$	
1	1	720	684	205	342		493	
	6	1440	1344	428	621		971	
2	3	644	567	371	800	✗	659	✗
	29	1288	1142	504	973		944	
	30	1932	1726	766	1543		1503	
3	7	438	368	78	124		243	
	14	876	806	195	319		509	
	20	438	339	29	37		98	
	25	876	782	153	228		423	
4	11	854	800	289	392		448	
	15	854	730	369	773		743	
	26	1708	1640	750	1616		1471	
	36	2562	2277	1234	2394		2224	
5	12	585	576	255	416		414	
	19	1170	1143	549	845		893	
6	16	150	143	29	29		29	
	22	300	283	73	89		115	
	24	300	258	48	56		84	
7	23	410	184	76	120		226	
	28	410	384	135	198		259	
	49	820	726	186	283		451	
8	27	960	960	411	708		700	
	34	1920	1770	896	2111	✗	1572	

9	32	986	858	552	992	✗	877	
	59	1972	1818	1302	2599	✗	1885	✗
	73	2958	2778	1826	3831	✗	2790	
	74	3944	3738	2311	4900	✗	3680	
	75	2958	2714	1851	3612	✗	2763	
10	40	46	46	8	8		8	
	55	92	92	17	18		19	
11	45	115	115	12	12		12	
	51	230	228	28	28		28	
	67	345	341	51	55		51	
	80	460	450	70	74		70	
	81	345	342	63	72		75	
12	46	373	365	101	132		197	
	60	373	320	108	137		194	
	63	746	651	214	285		389	
13	54	68	60	15	21		23	
	69	136	120	27	36		43	
14	57	573	480	129	148		189	
	79	1146	1052	397	673		653	
15	61	972	920	819	1357	✗	838	
	68	1944	1791	1458	2793	✗	1845	✗
	84	1944	1804	1498	3363	✗	1879	✗
16	62	157	146	37	44		59	
	93	314	261	56	66		90	
17	65	457	456	159	200		281	
	95	914	885	355	484		612	
18	66	27	27	2	2		2	
	71	54	50	4	4		4	
19	76	32	32	4	4		4	
	82	64	59	10	11		12	
20	92	972	920	409	627		663	
	94	1944	1796	965	1850	✗	1599	

Table 7-5 : Example Test-Set - End-To-End Response Time Analysis

Notice from table 7-5 that the activities declared infeasible at the localised analysis level (by the SbNN tests) relate to only a small number of transactions - those with longer periods which will tend to be assigned lower priorities in the priority-based implementation (since deadline monotonic priority assignment is used and, at this stage of the assessment, activity deadlines are equal to their periods).

7.2.2 Multiple Test-Set Results

This section gives the results from applying the above test-set generation and analysis procedure for a total of 500 test-sets. As before, the maximum utilisation of any one activity is limited to 20% and the maximum resource utilisation (prior to transaction definition) is limited to 80%.

The first set of results relate to the *responsiveness* of each of the schemes *RBA-Cyclic*, *FP-Exact*, *FP-SbNN1* and *FP-SbNN2*. A responsiveness value is calculated per activity per scheme as the predicted worst-case response time divided by the deadline. Hence, a value of 1 or less means that the activity is feasible, the closer to zero the shorter the response time compared to the deadline. The responsiveness values for all activities from all 500 test-sets are then plotted for each scheme in the form of a histogram, as shown in figure 7-3. The 'frequency' axis in each graph gives the number of responsiveness values found to lie in each interval [0.0, 0.1), [0.1, 0.2),…, [1.3, 1.4), [1.4, +∞).

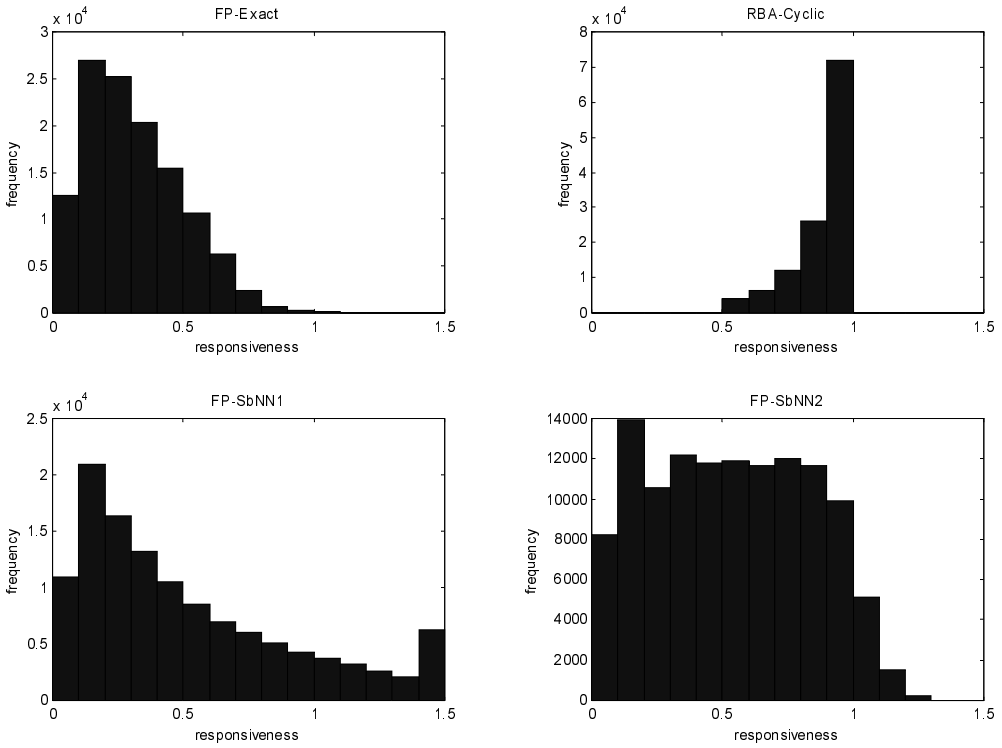


Figure 7-3 : Multiple Test-Set Results - Responsiveness

The following observations can be made :

- all activity response times are less than or equal to their corresponding deadline under *RBA-Cyclic* and, hence, these activities will be feasible wherever that the final bandwidth allocation for the resource is less than 100%;
- a very small number of activities are infeasible under *FP-Exact* (so few that this is hardly visible in the plot);

- for feasible activities, responsiveness values are generally lower under *FP-Exact* than under *RBA-Cyclic*, correspond to shorter response times relative to the same deadline;
- a number of activities are infeasible under *FP-SbNN2* and moreso under *FP-SbNN1* (notice that many responsiveness values are found to lie in the range [1.4, +∞));
- the general shape of each histogram is different - for both *FP-Exact* and *FP-SbNN1*, the frequency distribution appears *Gaussian* with positive ('left') skew, the former being spread over a smaller range; for *RBA-Cyclic*, the frequency distribution appears *J-shaped* with negative ('right') skew and with an even smaller range than *FP-Exact*; for *FP-SbNN2*, the frequency distribution is approaching *rectangular* with a similar range to *FP-Exact*.

Note that the responsiveness value is optimistic for *FP-Exact* when this test fails to converge to a value $R_j < T_j$ since the response time value itself is optimistic under such circumstances [Tindell94]. This situation does not arise for *RBA-Cyclic* (at this stage of assessment) since the response time analysis will always produce a result that is less than or equal to the deadline (feasibility is then dependent upon the final bandwidth allocation being granted on a particular resource).

The next set of results relate to the resource utilisation characteristics of the *RBA-Cyclic* scheme. The histograms in figure 7-4 show the true utilisation requirements, the final *RBA-Cyclic* allocation, the resulting over-allocation (the difference between the previous two values) and the percentage over-allocation.

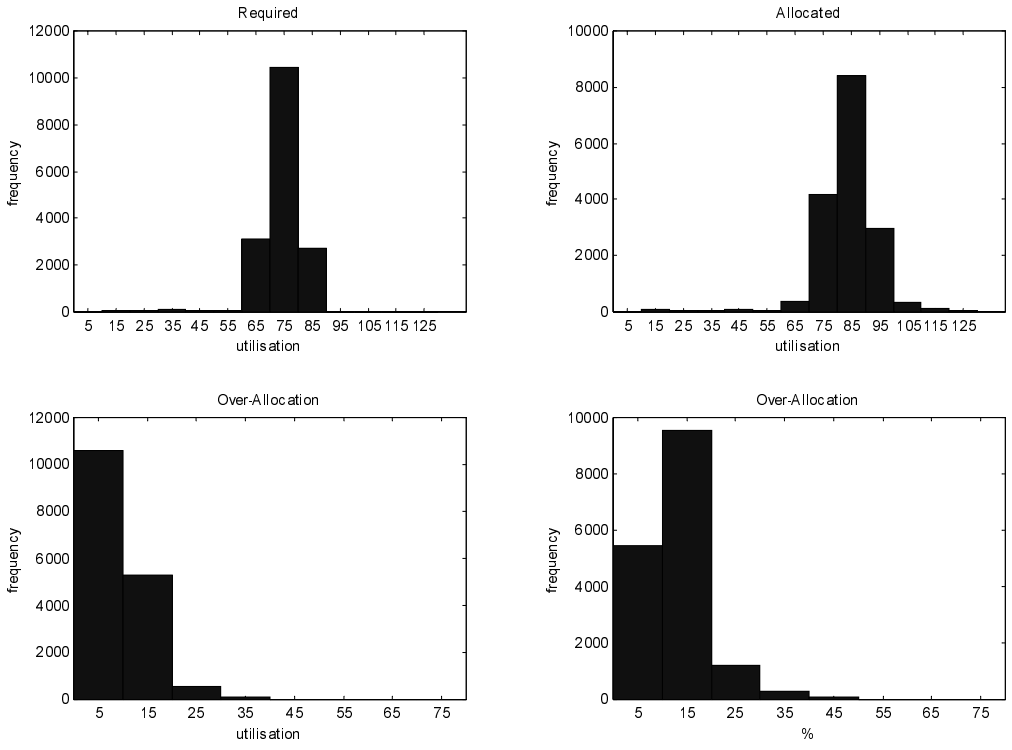


Figure 7-4 : Multiple Test-Set Results - Resource Utilisation for RBA-Cyclic Scheme

The following observations can be made :

- the total required utilisation per resource is variable between 60-90% (plus a small number of additional resources with lower utilisation requirements in the range 10-60% - these represent the last resources allocated during the test-set generation process, where only a small number of activities remain to be assigned);
- the final bandwidth allocation for *RBA-Cyclic* is less than 100% for most but not all resources - hence, for those resources with greater than 100% allocation, activities would need to be re-allocated elsewhere to reduce this figure to 100% or less;
- the total bandwidth over-allocation is less than 20%, in absolute terms, or 30%, in relative terms, for the majority of resources (and no resource has been over-allocated by more than 40% absolute or 50% relative).

Effect of reducing the maximum utilisation per activity :

Keeping the maximum resource utilisation at 80% but reducing the maximum activity utilisation to 10%, *ie.* increasing the number of activities per resource, gives the results shown in figures 7-5 and 7-6.

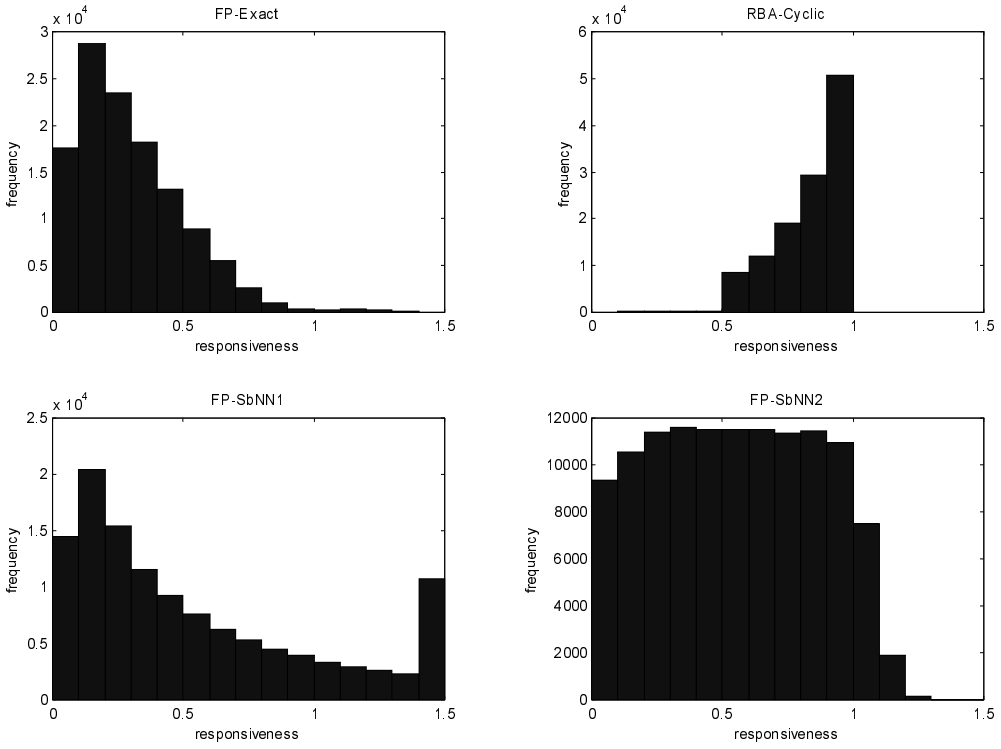


Figure 7-5 : Multiple Test-Set Results - Responsiveness

The impact of reducing the maximum activity utilisation from 20% to 10% on the responsiveness results can be summarised as follows :

- all activity response times are still less than or equal to their corresponding deadline under *RBA-Cyclic* (the vast majority of responsiveness values are in the range [0.5, 1]) -

as before, the final bandwidth allocation figures must be checked for each resource to determine feasibility;

- most, but not all, activities are feasible under *FP-Exact* and response times are mostly good compared to their corresponding deadline (many of the responsiveness values are in the range (0, 0.5]);
- the total number of activities found to be infeasible by the tests *FP-Exact*, *FP-SbNN1* and *FP-SbNN2* are all increased compared to the previous results;
- the basic shape of each of the histograms is consistent with previous results as illustrated in figure 7-3.

The histograms in figure 7-6 show the true utilisation requirements, the final *RBA-Cyclic* allocation and the resulting over-allocation (in absolute and percentage terms).

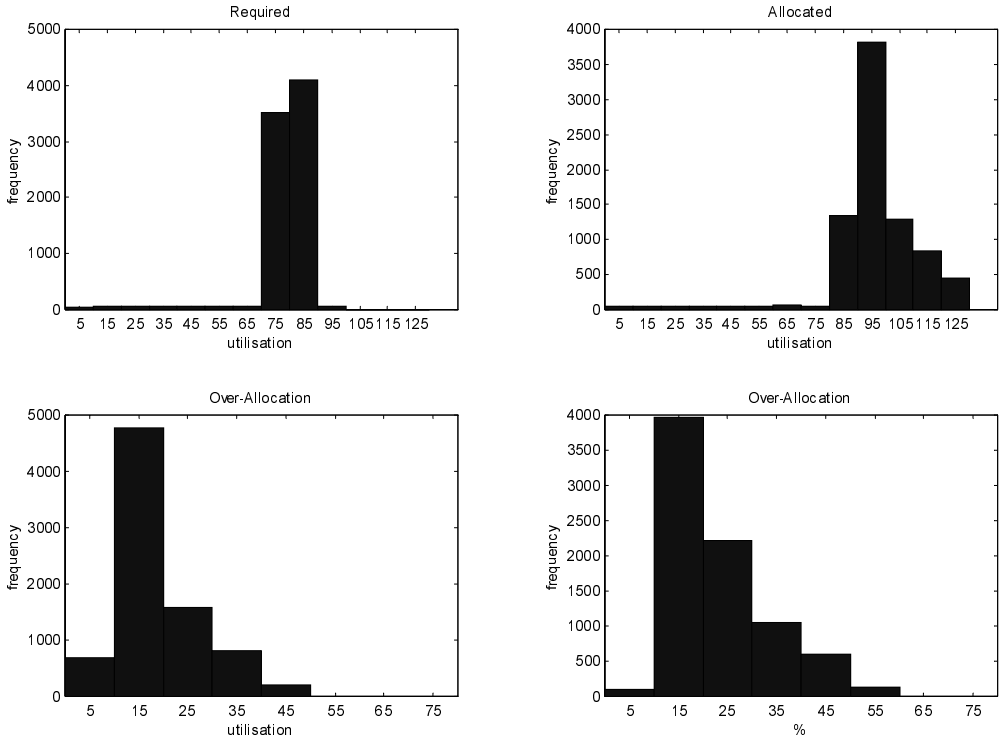


Figure 7-6 : Multiple Test-Set Results - Resource Utilisation for RBA-Cyclic Scheme

The impact of reducing the maximum activity utilisation from 20% to 10% on the resource utilisation results for the *RBA-Cyclic* scheme can be summarised as follows :

- the range of utilisation requirements per resource is compressed to between 70-90% in the majority of cases, compared to 60-90% previously (again, a small number of additional resources with lower utilisation requirements result from the test-set generation process);

- the final bandwidth allocation for *RBA-Cyclic* is over 100% for a significant number of resources - for such resources, activities would need to be re-allocated to reduce this figure to 100% or less in order to establish feasibility;
- the bandwidth over-allocation is less than 40%, in absolute terms, or 50%, in relative terms, for the majority of resources (no resource has been over-allocated by more than 50% absolute or 60% relative).

Effect of reducing the maximum resource utilisation :

Keeping the maximum activity utilisation at 10% but reducing the maximum resource utilisation to from 80% to 60% gives the results shown in figures 7-7 and 7-8.

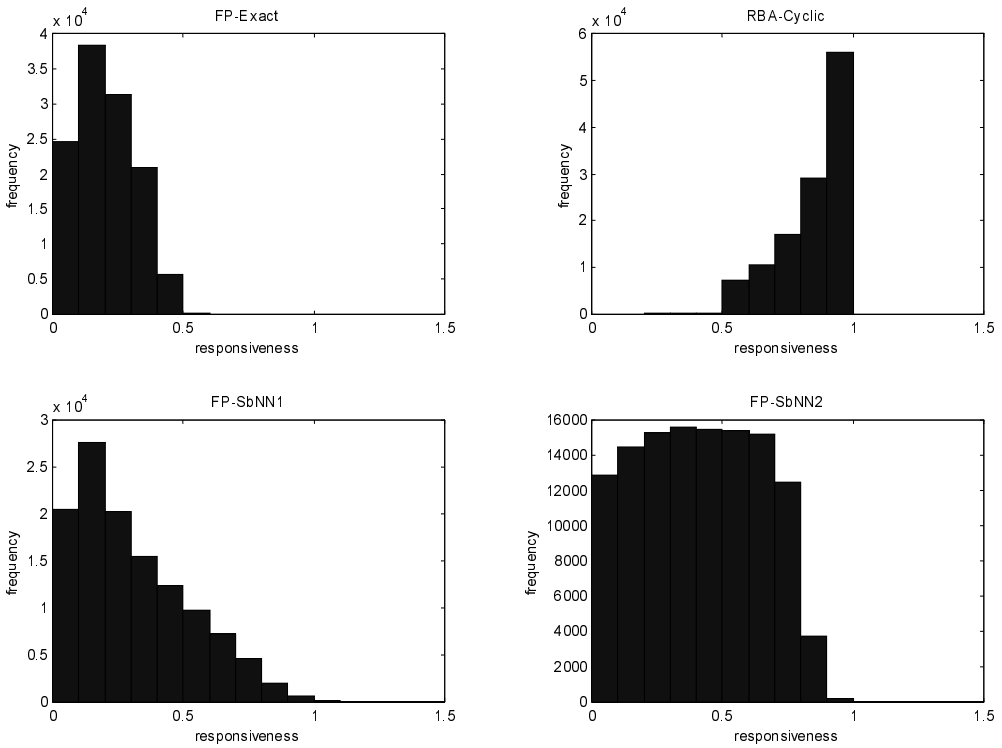


Figure 7-7 : Multiple Test-Set Results - Responsiveness

The impact of reducing the maximum resource utilisation from 80% to 60% on the responsiveness results can be summarised as follows :

- all activity response times are still less than or equal to their corresponding deadline under *RBA-Cyclic* (the vast majority of responsiveness values are still in the range [0.5, 1]) - as before, the final bandwidth allocation figures must be checked for each resource to determine feasibility;
- all activities are now feasible under *FP-Exact* and response times are good compared to their corresponding deadline (the vast majority of responsiveness values are in the range (0, 0.5]);

- all activities have now become feasible under *FP-SbNN2* and only a small number are infeasible under *FP-SbNN1*.

The histograms in figure 7-8 show the true utilisation requirements, the final *RBA-Cyclic* allocation and the resulting over-allocation (in absolute and percentage terms).

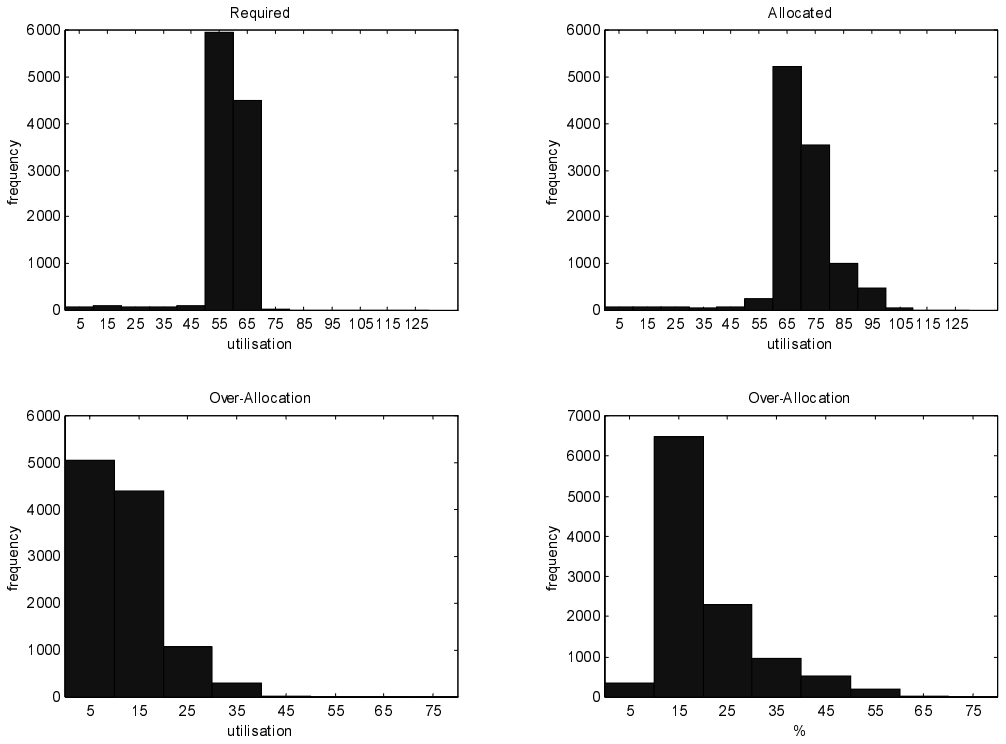


Figure 7-8 : Multiple Test-Set Results - Resource Utilisation for RBA-Cyclic Scheme

The impact of reducing the maximum resource utilisation from 80% to 60% on the resource utilisation results for the *RBA-Cyclic* scheme can be summarised as follows :

- the range of utilisation requirements per resource is reduced to between 50-70% in the majority of cases, compared to 70-90% previously;
- the final bandwidth allocation for *RBA-Cyclic* is less than 100% for all but a small number of resources, for which activities would need to be re-allocated to reduce this figure to 100% or less in order to establish feasibility;
- the bandwidth over-allocation, in both absolute and relative terms, is reduced compared to the previous results.

Conclusions from the first stage of assessment :

The following overall conclusions can be drawn from this first stage of assessment :

- the extent of the bandwidth over-allocation for the *RBA-Cyclic* scheme appears to be manageable for practical applications - averaging around 20% absolute and around 25% relative and peaking between 2-3 times the average;

- the bandwidth over-allocation for the *RBA-Cyclic* scheme is reduced when the total resource utilisation requirements are reduced - this is due (other than to random variations in the test-set characteristics) to the number of activities allocated to each resource being reduced, so increasing the chances of finding a more efficient schedule.

7.3 Assessment Stage 2 : Event-Triggered (Jitter-Controlled) Transactions - Responsiveness and Bandwidth Utilisation

In this section, the *RBA-Cyclic* and *FP-Exact* schemes are considered for event-triggered transactions - the two SbNN tests are not appropriate for this stage of assessment since they make no allowance for release jitter. Even for the *RBA-Cyclic* and *FP-Exact* schemes, jitter is only tolerable to a specifiable extent and jitter controls must be enforced according to the approach described in section 5.1 in order to comply with certain constraints as outlined below. Beyond that, further adaptation is required to tolerate jitter to an arbitrary extent.

Both of the schemes *RBA-Cyclic* and *FP-Exact* require $R_j \leq a_j$, *ie.* at most one invocation of λ_j can be ready for execution at any time. Since $a_j = T_j - J_j^{in}$, this constraint translates into a release jitter constraint $J_j^{in} \leq T_j - R_j$. When the precise value of R_j is unknown initially, the stronger constraint $J_j^{in} \leq T_j - D_j$ can be applied to ensure that the true constraint is met. If the release jitter is permitted to invalidate the former constraint, it becomes possible for multiple invocations of λ_j to be ready for execution at the same time. In such circumstances, λ_j is said to become *backlogged* and the *RBA-Cyclic* scheme (as currently defined) gives rise to a final bandwidth allocation that is insufficient to meet the true requirements. Analogously, the worst-case response time predicted by *FP-Exact* becomes optimistic, *ie.* may be less than the true worst-case response time. The more the release jitter is allowed to increase, the greater the extent to which an activity can potentially become backlogged, successive invocations effectively joining a release queue for the activity where the response time analyses for *RBA-Cyclic* and *FP-Exact* (given in equations *(2-7a)* and *(6-19)*, respectively) merely give the time to service the entry at the head of the queue, *ie.* do not account for the time spent in the release queue.

In order to generate appropriate test-sets for this stage of assessment, the methodology described in section 7.2.1 is extended as follows :

- instead of simply assigning deadlines such that $D_j = T_j$, a value for each D_j is sampled at random from a uniform distribution defined over the range $[C_j, T_j]$;
- a value for each J_j^{in} is then sampled at random from a uniform distribution defined over the range $[0, T_j - D_j]$, *ie.* observing the constraint $J_j^{in} \leq T_j - D_j$.

The results shown in figures 7-9 and 7-10 are obtained when the total resource utilisation is limited to just 30% (prior to allocation), keeping the maximum activity utilisation at 10%.

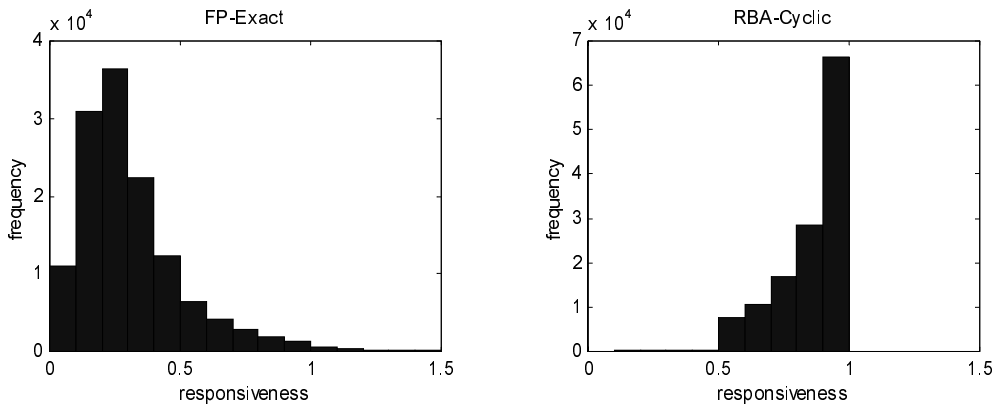


Figure 7-9 : Multiple Test-Set Results - Responsiveness

The following observations can be made :

- for *RBA-Cyclic*, all activity response times are less than or equal to their corresponding deadline (the vast majority of responsiveness values are still in the range [0.5, 1]) - the final bandwidth allocation figures must be checked for each resource to determine feasibility;
- for *FP-Exact*, a small number of activities are infeasible but, for feasible activities, response times are generally good compared to their corresponding deadline (a large proportion of responsiveness values are in the range (0, 0.5]).

The histograms in figure 7-10 show the true utilisation requirements, the final *RBA-Cyclic* allocation and the resulting over-allocation (in absolute and percentage terms). The following observations can thus be made for *RBA-Cyclic* :

- the range of utilisation requirements per resource is between 20-40% in the majority of cases;
- the final bandwidth allocation for *RBA-Cyclic* is over 100% for around a third of all resources, for which activities would need to be re-allocated to reduce this figure to 100% or less in order to establish feasibility;
- the bandwidth over-allocation is greater than 100% in absolute terms for a number of activities and greater than 100% in relative terms for the majority.

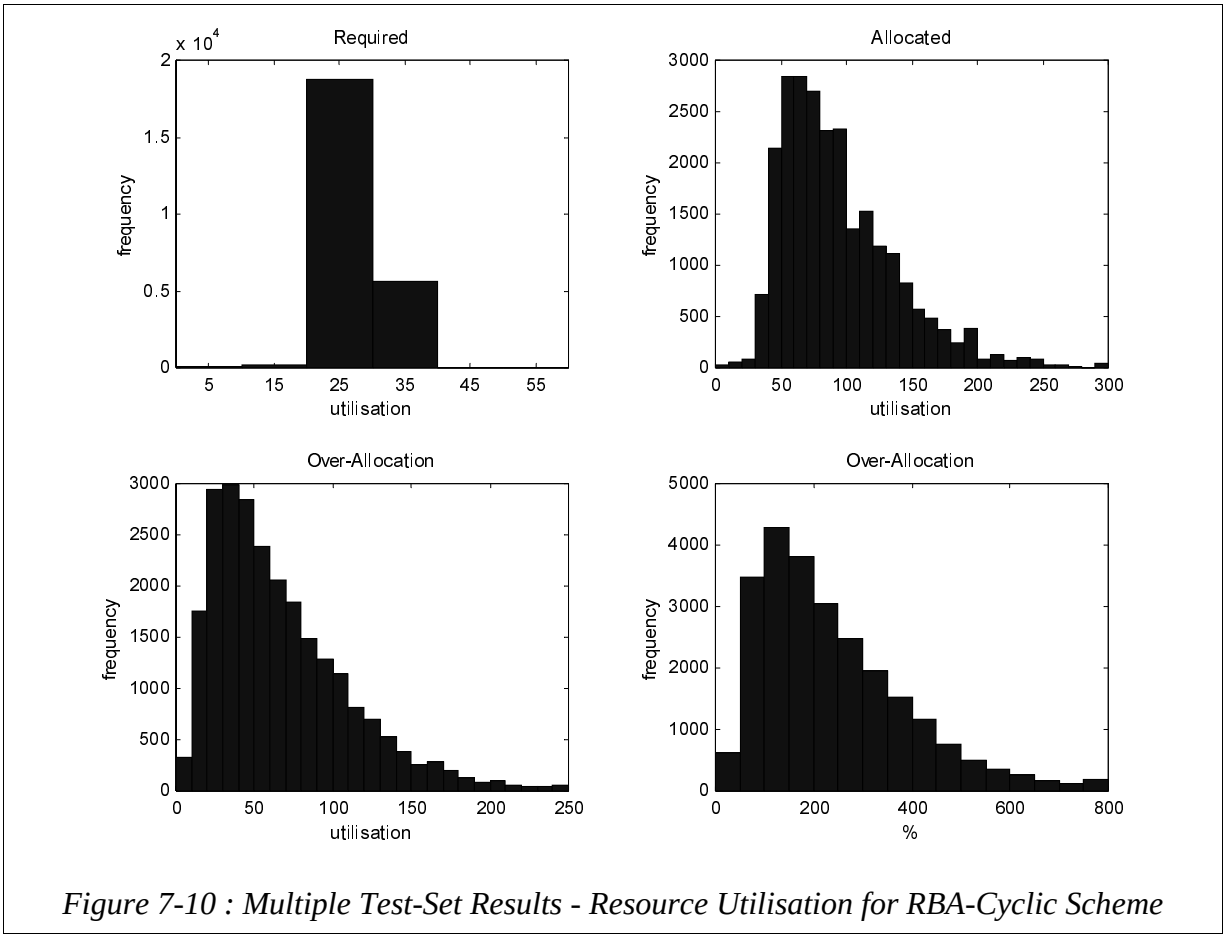


Figure 7-10 : Multiple Test-Set Results - Resource Utilisation for RBA-Cyclic Scheme

The detrimental effect on the behaviour of the *RBA-Cyclic* scheme can be seen by considering the two factors that have been changed in the test-set generation process compared to the previous stage of assessment :

- the constraint $D_j = T_j$ has been relaxed to $C_j \leq D_j \leq T_j$ - the bandwidth allocation scheme operates on the basis of D_j rather than T_j, *ie.* allocates sufficient bandwidth to meet the deadline constraint in response to an invocation request at any time provided that no more than one invocation is active at any one time; hence, the shorter the deadline compared to the period, the greater the bandwidth over-allocation;
- release jitter has been introduced subject to the constraint $J_j^{in} \leq T_j - D_j$ - the impact on *RBA-Cyclic* is null since the scheme allocates sufficient bandwidth to cope with the stronger constraint $J_j^{in} \leq T_j - R_j$.

This illustrates both a strength and a weakness of the RBA-Cyclic scheme. The strength being the ability to tolerate jitter (or, perhaps more significantly, changes in jitter) within the range [0, T_j - R_j]. The weakness is that the bandwidth over-allocation can be significant for activities with short deadlines compared to their period (or, more generally, their minimum inter-arrival time). This weakness will be addressed through the provision of an extended implementation scheme in chapter 8.

Effect of restricting shortness of deadlines :

The results shown in figures 7-11 and 7-12 further illustrate the above points. These results are obtained when the total resource utilisation is increased from 30% to 60% (prior to allocation), keeping the maximum activity utilisation at 10%, but reducing the range of possible deadline values. Each D_j value is restricted to the upper half of the range $[C_j, T_j]$, which has the effect of removing activities with shorter deadlines from the test-set. Release jitter values are generated as before.

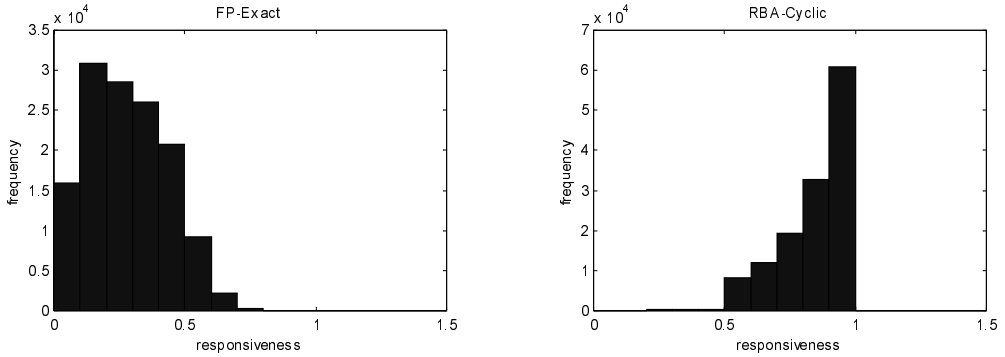


Figure 7-11 : Multiple Test-Set Results - Responsiveness

The impact on the responsiveness results can be summarised as follows :

- for *FP-Exact*, all activities are now feasible - this is largely due to a corresponding decrease in release jitter rather than the removal of activities with shorter deadlines from the test-set;
- for *RBA-Cyclic*, the impact on responsiveness is minimal and entirely due to random variations in the test-set generation process.

The histograms in figure 7-12 show the true utilisation requirements, the final *RBA-Cyclic* allocation and the resulting over-allocation (in absolute and percentage terms). The impact on the resource utilisation results for *RBA-Cyclic* can be summarised as follows :

- the range of utilisation requirements per resource is increased to between 50-70% in the majority of cases, compared to 20-40% previously;
- the final bandwidth allocation for *RBA-Cyclic* is reduced considerably compared to the previous results, even though the true requirement has doubled on average;
- the bandwidth over-allocation is also reduced to what are considered to be manageable levels in practice - around 35% average (90% peak) in absolute terms or around 60% average (130% peak) in relative terms.

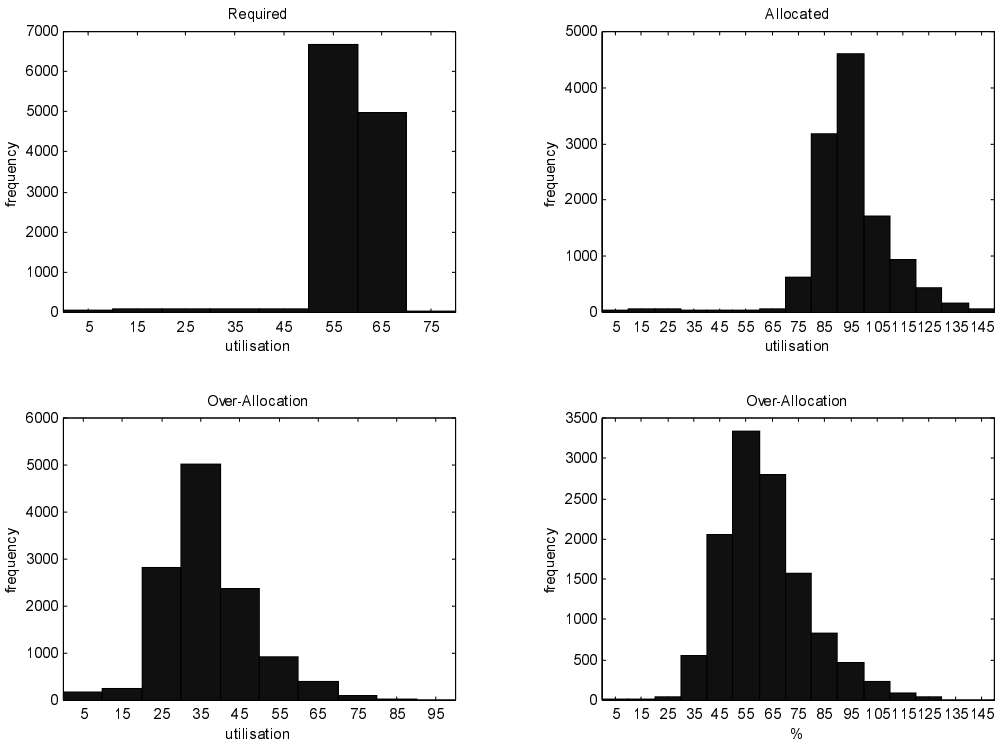


Figure 7-12 : Multiple Test-Set Results - Resource Utilisation for RBA-Cyclic Scheme

Conclusions from the second stage of assessment :

The following overall conclusions can be drawn from this stage of assessment :

- the extent of the bandwidth over-allocation for the *RBA-Cyclic* scheme is increased when the constraint $D_j = T_j$ is relaxed to $C_j \leq D_j \leq T_j$;
- given that efficiency of bandwidth utilisation is not the primary goal of the *RBA-Cyclic* scheme the extent of the bandwidth over-allocation is still considered to be reasonable when offset by the ability to perform independent response time analysis.

Note that other reservation-based scheduling schemes have been shown to suffer significant performance degradation as jitter increases. In [Jeffay00], for example, the schemes EEVDF [Stoica96] and CBS [Abeni98] are both shown to give rise to significantly increased response times and, consequently, missed deadlines. Notice the difference between these schemes and RBA in such circumstances is that RBA sacrifices resource bandwidth rather than timeliness.

7.4 Assessment Stage 3 : Sensitivity to Change - Time-Triggered and Event-Triggered (Jitter Controlled) Transactions

This section considers the impact of various changes in system timing attributes and implementation details on the different implementation/analysis schemes under assessment. Firstly, a single test-set example is given to illustrate the impact of a computation time change on a set of localised response time analysis results. Subsequently, the impact of

computation time changes and transaction period changes are considered for multiple test-sets.

7.4.1 Example Test-Set Results

This section examines the impact of a simple change in timing attributes on the response time analysis results for the example test-set used in section 7.2.1. Consider a change to a single activity computation time value on a single resource - for activity λ_{55} on resource 6 increase the computation time from C_{55} =7 to C_{55} =9. The impact on the localised response time analysis is shown in table 7-6, which gives the new response time values followed by the size of the increase relative to the old value (from table 7-2).

resource	λ_j	$R_j^{RBA-Cyclic}$		$R_j^{FP-Exact}$		$R_j^{FP-SbNN1}$		$R_j^{FP-SbNN2}$	
6	86	414	*0*	295	*+21*	612	*+77*	564	*+28*
	7	368	*0*	84	*+6*	139	*+15*	263	*+20*
	23	184	*0*	80	*+4*	134	*+14*	244	*+18*
	39	276	*0*	79	*+4*	104	*+10*	178	*+16*
	55	46	*0*	11	*+2*	12	*+2*	13	*+2*
	71	23	*0*	2	*0*	2	*0*	2	*0*
	87	69	*0*	16	*+2*	22	*+3*	31	*+4*
	8	575	*0*	242	*+19*	337	*+31*	414	*+76*
	24	115	*0*	21	*+2*	32	*+5*	63	*+8*

Table 7-6 : Example Test-Set - Localised Analysis Results for Resource 6 (After Change)

All activities remain feasible for all tests but the response times predicted by the priority-based tests are increased for all but the highest priority activity. Clearly, there is also a corresponding impact on the cumulative response times of the end-to-end transactions to which these activities contribute in the example test-set, *ie.* transactions 3, 6, 7 and 10. The response time analysis results are still valid for all other resources, however, since all of the priority-based models examined assume critical instant release, *ie.* do not explicitly account for release offsets. The change has no impact on the *RBA-Cyclic* response times and, consequently, no impact on transaction response times.

In terms of bandwidth utilisation, the true requirement for resource 6 is increased from 69% to 73%. The final bandwidth allocation for the *RBA-Cyclic* scheme is increased from 87% (an over-allocation of 28%) to 91% (an over-allocation of 26%). Notice that the over-allocation has actually been reduced - this is because the actual increase in utilisation is the same for both the required and allocated values, *ie.* 4% (4.35% in precise terms given that the figures in table 7-6 are rounded to the nearest integer). The same amount of bandwidth is reclaimed as before since the change does not affect the low utilisation activities allocated to cyclic servers. Significantly, the increase in the overall bandwidth allocation is entirely due to the increase in the bandwidth allocation for activity λ_{55} - the value of δ_{55} is increased from 4 to 5 and, in turn, the value of v_{55}^{δ} is increased from 0.1739 to 0.2174, an increase of

0.0435, *ie.* 4.35%. Hence, the impact of the change can be assessed for feasibility by merely checking that the increase in bandwidth allocation for the affected activity is within the spare capacity of the resource. This illustrates a considerable benefit of the *RBA-Cyclic* scheme (and RBA in general) for practical applications.

This example has examined the impact of a single computation time change within a system. Clearly, more widespread changes can result in more widespread impact. For example, a change to the period of a transaction can potentially affect all localised response time analysis results for all resources to which its component activities are allocated. The impact of transaction-wide changes will be investigated for multiple, randomly generated test-sets later in the next section.

7.4.2 Multiple Test-Set Results

This section considers the impact of activity computation time changes and transaction period changes for multiple, randomly generated test-sets involving firstly time-triggered and then event-triggered (jitter controlled) transactions. Such changes are injected into each test-set and the impact observed in terms of the scope of impact relative to the system (test-set) as a whole, *ie.* the overall percentage of activities affected, and relative to the scope of the initial change. The latter observation gives an indication of the *sensitivity* of the scheme and is referred to as such in the results tables that follow. From a systems engineering perspective, the sensitivity value gives an indication of the extent to which re-work/re-verification would be required and is more significant than the actual magnitude of the response time increases involved. More precisely, it is only those effects that involve response time increases, *ie.* not decreases, that are truly significant and so the sensitivity of each scheme is expressed in two ways - firstly, it is evaluated over all response time changes and then over the response time increases only.

Effect of increasing activity computation times (time-triggered transactions) :

This experiment considers the impact of making a change to a single activity computation time value per test-set for the case of time-triggered transaction test-sets where all forms of analysis given in section 7.1 are applicable. Table 7-7 gives the results for test-sets that are subject to the constraints that the overall resource utilisation limit is 60% and the maximum activity utilisation is 10%. Injected changes involve increasing the computation time of one activity selected at random per test-set by 10%. The first row of the table gives the total number of changes injected (over 500 test-sets) followed by the number of changes observed for each scheme, the second row gives the percentage of the test-set affected by the change and the third row gives the sensitivity of each scheme based on all observed changes in response times.

	Test-Set	*RBA-Cyclic*	*FP-Exact*	*FP-SbNN1*	*FP-SbNN2*
No. of Changes	500	245	3228	3228	3228
% Test-Set Affected	0.3957	0.1939	2.5546	2.5546	2.5546
Sensitivity	1.0000	0.4900	6.4560	6.4560	6.4560

Table 7-7 : Multiple Test-Set Results - Sensitivity (All Changes)

These figures can be interpreted as follows :

- a total of 500 changes were injected, *ie.* one per test-set examined, which represents 0.3957% of activities affected per test-set on average;
- for *RBA-Cyclic*, these 500 computation time changes induced a total of only 245 response time changes, which represents 0.1939% of each test-set on average, giving an overall sensitivity value of 0.49 (the scope of the impact is 49% of that of the initial injected change);
- for *FP-Exact*, the same changes induced a total of only 3228 response time changes, which represents 2.5546% of each test-set on average, giving an overall sensitivity value of 6.4560 (the scope of the impact is 645.6% of that of the initial injected change);
- figures identical to those obtained for *FP-Exact* were obtained for each of the two SbNN tests.

The scope of impact is considerably less for the *RBA-Cyclic* scheme than for any of the priority-based schemes. Further details of the impact can be seen by examining the magnitude of the response time changes, as illustrated in figure 7-13, which shows the percentage change (increase or decrease) in responsiveness values under each scheme. Note that these figures are similar but not identical for the different form of priority-based models.

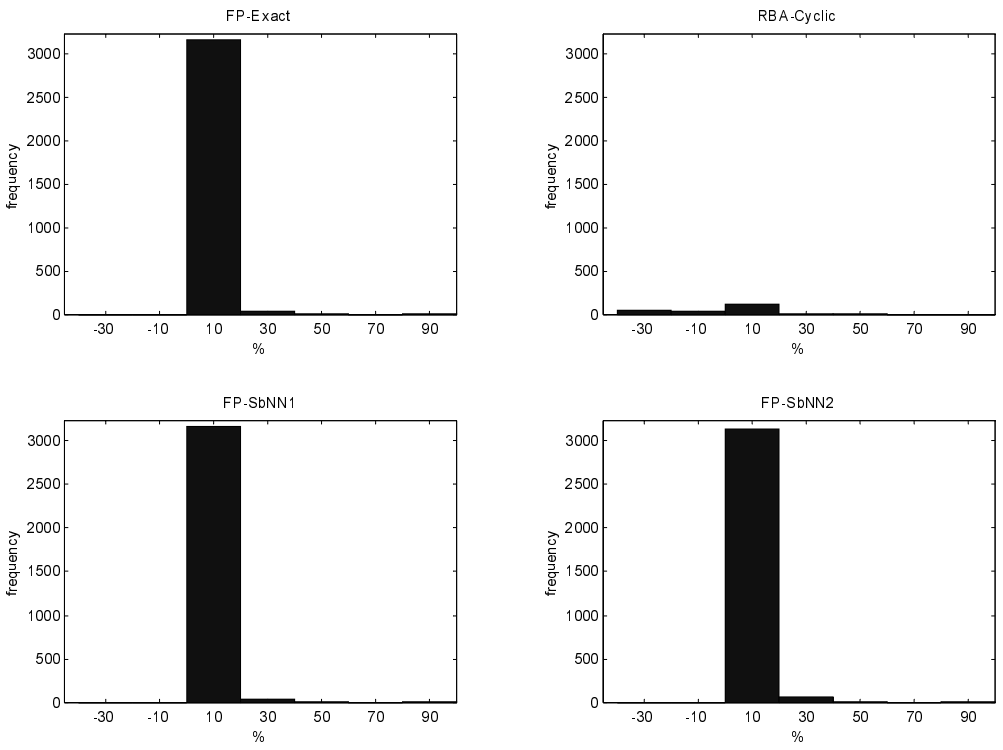


Figure 7-13 : Multiple Test-Set Results - Changes in Responsiveness

The figure shows that, under *RBA-Cyclic*, some of the response time changes are actually decreases, even though the original injected change is an increase in computation time. This is due to the need to allocate bandwidth at a finite granularity and can be seen from the following example extracted from one of the test-sets used in the experiment. Consider an activity with the timing attributes $\{C_i = 35, D_i = 536\}$ and subjected to a 10% increase in computation time (where the cycle time for the overall resource schedule is $\Delta = 44$). The response time prior to the change is calculated as follows :

original computation time : C = 35;

normalised deadline : $D^{\Delta} = \lceil 536/44 \rceil * 44 = 528$;

normalised execution rate : $v^{\Delta} = 35/528 = 0.0663$;

execution time per Δ : $\delta = \lceil 44*0.0663 \rceil = 3$;

response time : $R^{\delta} = \lceil 35/3 \rceil * 44 = 528$.

The response time following the change is calculated as follows :

new computation time : $C = \lceil 35*1.1 \rceil = 39$;

normalised deadline : $D^{\Delta} = \lceil 536/44 \rceil * 44 = 528$;

normalised execution rate : $v^{\Delta} = 39/528 = 0.0739$;

execution time per Δ : $\delta = \lceil 44*0.0739 \rceil = 4$;

response time : $R^{\delta} = \lceil 39/4 \rceil * 44 = 440$.

Consequently, table 7-8, which gives the sensitivity results calculated on the basis of response time increases only, shows a reduced sensitivity value for *RBA-Cyclic*. The results for the priority-based schemes are the same as in table 7-7, *ie.* no decreases are observed.

	Test-Set	*RBA-Cyclic*	*FP-Exact*	*FP-SbNN1*	*FP-SbNN2*
No. of Changes	500	153	3228	3228	3228
% Test-Set Affected	0.3957	0.1211	2.5546	2.5546	2.5546
Sensitivity	1.0000	0.3060	6.4560	6.4560	6.4560

Table 7-8 : Multiple Test-Set Results - Sensitivity (Responsiveness Increases Only)

Each of the response time decreases actually represents a bandwidth reservation increase and so, in practice, the final resource utilisation figures must be checked for those resources affected by the change (one per test-set in this experiment). Also in practice, the required extent of re-analysis under *RBA-Cyclic* is greater than that suggested by these figures. A more practical value for the sensitivity parameter would be 1.0, *ie.* the response times (and final bandwidth allocation) of all activities directly involved in the injected changes (500 in all) would need to be re-calculated. No activity outside the direct scope of the changes, however, needs to be re-analysed. For the priority-based schemes the minimum requirement in practice would be to re-establish feasibility for all activities directly affected by the change and all lower priority activities on the shared resources in question.

Effect of increasing activity computation times (event-triggered transactions) :

Repeating the above experiment for event-triggered (jitter controlled) transactions, keeping the resource utilisation limit at 60%, the activity utilisation limit at 10% and the magnitude of the injected computation time changes at 10% (one activity per test-set), gives the results shown in tables 7-9. Recall that the two SbNN tests are not valid for event-triggered transactions.

	Test-Set	*RBA-Cyclic*	*FP-Exact*
No. of Changes	500	257	3125
% Test-Set Affected	0.4012	0.2062	2.5077
Sensitivity	1.0000	0.5140	6.2500

Table 7-9 : Multiple Test-Set Results - Sensitivity (All Changes)

Figure 7-14 shows the percentage change (increase or decrease) in responsiveness values under each scheme.

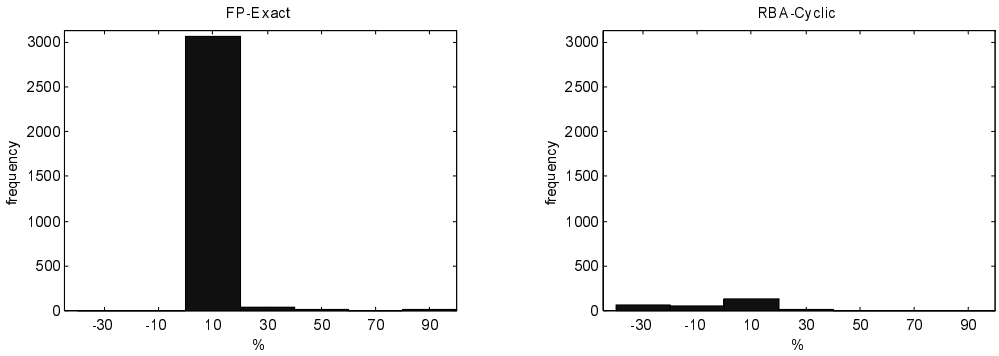


Figure 7-14 : Multiple Test-Set Results - Changes in Responsiveness

Considering the impact of the changes in terms of resulting response time increases only, gives the results shown in table 7-10.

	Test-Set	*RBA-Cyclic*	*FP-Exact*
No. of Changes	500	150	3125
% Test-Set Affected	0.4012	0.1204	2.5077
Sensitivity	1.0000	0.3000	6.2500

Table 7-10 : Multiple Test-Set Results - Sensitivity (Responsiveness Increases Only)

The results obtained for event-triggered (jitter-controlled) transactions are very similar to those obtained for time-triggered transactions. This might be expected when the injected changes involve the computation time attributes only, *ie.* do not affect the transaction level attributes such as period and jitter.

Effect of decreasing transaction periods (time-triggered transactions) :

This experiment considers the impact of making a change to a single transaction period per test-set for the case of time-triggered transaction test-sets where all forms of analysis given

in section 7.1 are applicable. Table 7-11 gives the results for test-sets that are subject to the constraints that the overall resource utilisation limit is 60% and the maximum activity utilisation is 10%. Injected changes involve decreasing the period of one transaction selected at random per test-set by 10% - the timing attributes of all activities associated with the transaction are modified accordingly. At this stage, the deadline of an activity is equal to its period and so a decrease in the period of an activity will mean a corresponding decrease in its deadline.

	Test-Set	*RBA-Cyclic*	*FP-Exact*	*FP-SbNN1*	*FP-SbNN2*
No. of Changes	3400	5117	6601	18441	10192
% Test-Set Affected	2.8160	4.2381	5.4672	15.2736	8.4414
Sensitivity	1.0000	1.5050	1.9415	5.4238	2.9976

Table 7-11 : Multiple Test-Set Results - Sensitivity (All Changes)

These figures can be interpreted as follows :

- a total of 3400 changes were injected (over 500 test-sets) which represents 2.8160% of activities affected per test-set on average;
- for *RBA-Cyclic*, these changes induced a total of 5117 response time changes, which represents 4.2381% of each test-set on average, giving an overall sensitivity value of 1.5050 (the scope of the impact is 150.5% of that of the initial injected change);
- for *FP-Exact*, the changes induced a total of 6601 response time changes, which represents 5.4672% of each test-set on average, giving an overall sensitivity value of 1.9415 (the scope of the impact is 194.15% of that of the initial injected change);
- the figures differ but can be similarly interpreted for the two SbNN tests - in general, *FP-SbNN2* does worse than *RBA-Cyclic* and *FP-Exact*, and *FP-SbNN1* does considerably worse than any of the other three tests.

The reason why the number of changes differs between the three forms of priority-based test can be seen by inspecting the equations given in section 7.1. For the *FP-Exact* test, neither the period or deadline of a task is recognised in the response time expression and so a change to either or both of these attributes will not affect the activity itself, only lower priority activities. There are more changes in response times for *FP-SbNN2* (than *FP-Exact*) since the deadline is explicitly recognised in the test. The reason that the number of response time changes is even larger for *FP-SbNN1* is that the denominator in the response time expression is a continuous quantity that changes whenever the period changes - in the former tests, the ceiling function around the part of the expression where the period attribute is a denominator provides a degree of robustness against changes to that value, *ie.* unless the period changes by more than a certain amount, the result of that part of the expression will be the same integer value.

For *RBA-Cyclic*, a decrease in the period of an activity will not, in itself, affect the response time of that activity but, since a decrease in the period of an activity causes a corresponding reduction in its deadline, the response time can be affected, since this is based on the deadline. Figure 7-15 shows the percentage change (increase or decrease) in responsiveness values under each scheme.

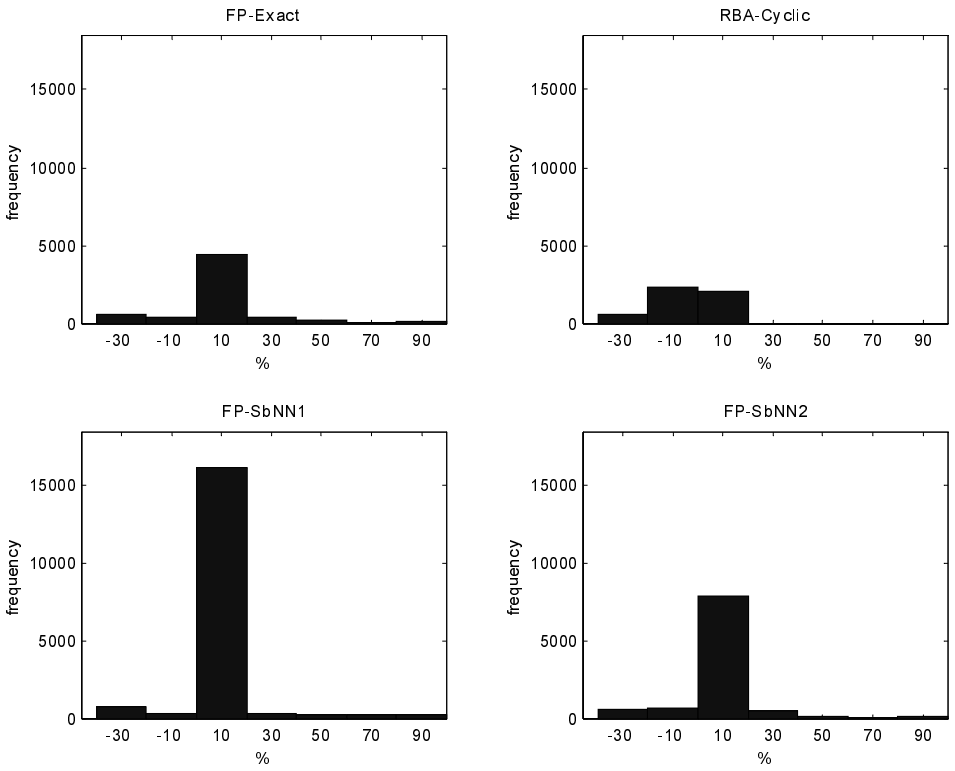


Figure 7-15 : Multiple Test-Set Results - Responsiveness

Whilst the total number of changes observed under the different priority-based schemes is different, the magnitude of the changes involved are similar and mostly in the 0-10% range. Response time changes for other activities that share the same resources can be explained as follows :

- for all of the priority-based schemes, a decrease in the period of an activity can increase the response time of a lower priority activity allocated to the same resource due to increased interference;
- for *RBA-Cyclic*, a decrease in the period of an activity will not, in itself, affect the response times of other activities that share the same resource but, since deadlines are also affected at this stage, a deadline that is reduced to a value below the current schedule cycle time will force a new cycle time to be chosen and so impact on some or all activity response times for that resource.

Considering the sensitivity of the tests in terms of resulting response time increases only, gives the results shown in table 7-12. Since the activities directly affected by the reduced period change will not be included in these figures, since their response times will not be

increased under any of the tests, these figures give an indication of the impact on other activities through resource sharing effects.

	Test-Set	*RBA-Cyclic*	*FP-Exact*	*FP-SbNN1*	*FP-SbNN2*
No. of Changes	3400	2189	5528	17328	8842
% Test-Set Affected	2.8160	1.8130	4.5785	14.3517	7.3233
Sensitivity	1.0000	0.6438	1.6259	5.0965	2.6006

Table 7-12 : Multiple Test-Set Results - Sensitivity (Responsiveness Increases Only)

The table shows that the sensitivity values are reduced only slightly for each of the priority-based tests - this reflects the extent to which activities with reduced periods impose increased interference on lower priority activities that share resources and also the fact that the priority orderings may be changed due to modified deadlines. For *RBA-Cyclic*, the sensitivity value is more than halved - this reflects the extent to which changes to activity deadlines have resulted in new values that are less than the corresponding schedule cycle time.

Effect of decreasing transaction periods (event-triggered transactions) :

Repeating the above experiment for event-triggered (jitter controlled) transactions, keeping the resource utilisation limit at 60%, the activity utilisation limit at 10% and the magnitude of the injected transaction period changes at 10% (one transaction per test-set), gives the results shown in tables 7-13. Recall that the two SbNN tests are not valid for event-triggered transactions. In this experiment, a decrease in activity period may or may not mean a decrease in its deadline - the deadline is only decreased if the new period is less than the original deadline (to prevent the deadline from becoming greater than the period and invalidating the tests).

	Test-Set	*RBA-Cyclic*	*FP-Exact*
No. of Changes	3778	877	2942
% Test-Set Affected	2.8822	0.6690	2.2444
Sensitivity	1.0000	0.2321	0.7787

Table 7-13 : Multiple Test-Set Results - Sensitivity (All Changes)

Compared to the previous case, the results are considerably improved for *RBA-Cyclic* since there are less changes to activity deadlines. Consequently, there is less impact on those activities directly implicated in the transaction period change (since response times are dependent upon deadlines rather than periods) and, also, there is less potential to force a change in schedule cycle time (again, this is based on deadlines and not periods).

The results for *FP-Exact* also show an improvement but this is less marked than for *RBA-Cyclic*. This is due to the fact that activities with reduced periods will impose increased interference on lower priority activities that share resources, irrespective of their deadlines.

There is less potential for priority orderings to be affected since these are based on deadlines rather than periods.

Figure 7-16 shows the percentage change (increase or decrease) in responsiveness values under each scheme.

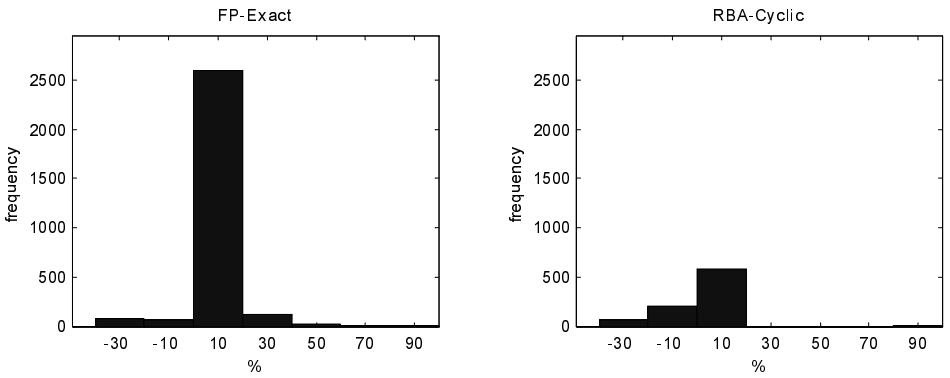


Figure 7-16 : Multiple Test-Set Results - Responsiveness

Considering the impact of the changes in terms of resulting response time increases only, gives the results shown in table 7-14.

	Test-Set	*RBA-Cyclic*	*FP-Exact*
No. of Changes	3778	600	2781
% Test-Set Affected	2.8822	0.4577	2.1216
Sensitivity	1.0000	0.1588	0.7361

Table 7-14 : Multiple Test-Set Results - Sensitivity (Responsiveness Increases Only)

The sensitivity value is further reduced by around one third for *RBA-Cyclic* and by around one twentieth for *FP-Exact*. This shows that the activities directly affected by the injected changes are the subject of a greater proportion of the observed effects for *RBA-Cyclic* than for *FP-Exact*.

Conclusions from the third stage of assessment :

The following conclusions can be drawn from this stage of assessment. Firstly, for the priority-based implementation schemes :

- *FP-Exact* is reasonably robust to changes in terms of the magnitude of the resulting response time increases (or decreases);
- the two sufficient-but-not-necessary tests turn out to be near identical in terms of sensitivity to computation time changes but *FP-SbNN2* is noticeably less sensitive to changes in transaction periods than *FP-SbNN1*;
- in practice, it would be necessary to re-verify response times for all activities directly affected by changes (even though the new response time may turn out to be the same as

before) and also all lower priority activities allocated to the corresponding shared resources.

Secondly, for the RBA cyclic implementation scheme :

- an increase in computation time can give rise to a decrease in response time for the same activity (given a fixed schedule cycle time);
- only changes that affect schedule cycle times can be propagated to activities other than those directly implicated in the change;
- in practice, it would be necessary to re-verify feasibility of all activities directly affected by changes (even though the new response time and associated bandwidth allocation may turn out to be the same as before).

7.5 Summary

This chapter has investigated the behaviour of the proposed RBA cyclic implementation scheme in comparison to those of a static priority-based implementation subject to different forms of response time analysis, namely exact analysis and two different forms of sufficient-but-not-necessary analysis. The schemes have been compared in terms of responsiveness, bandwidth utilisation (for the RBA implementation scheme only) and change sensitivity. These characteristics have been investigated in the context of time-triggered transactions (with deadline equal to period at the activity level) and event-triggered transactions (with constrained release jitter at the activity level).

In terms of responsiveness, the priority-based implementation is superior, which can be expected due to its work-conserving nature. The *RBA-Cyclic* scheme always produces a response time less than or equal to the deadline (given that the deadline is no less than the underlying schedule cycle time) but feasibility also requires the final bandwidth allocation figures to be checked. For this reason, the bandwidth over-allocation associated with the RBA implementation has also been examined. In this respect, the main factor to affect the efficiency of the RBA implementation is the difference between the deadline and the period of an activity - the following results were obtained :

- when activity deadlines are equal to their periods, the bandwidth over-allocation is at a level that is reasonable for practical applications, averaging around 25% relative to the true utilisation requirements and peaking at 2-3 times the average;
- as the deadline of an activity decreases relative to its period, the bandwidth over-allocation increases correspondingly - the over-allocation remains at a level that is reasonable for practical applications whilst the deadline is in the upper half of the interval $[C_j, T_j]$.

In general, the bandwidth over-allocation is reduced when the total number of activities allocated per resource is reduced - the chances of finding a more efficient schedule is

increased. Overall, given that efficiency of bandwidth utilisation is not the primary goal of the RBA implementation scheme, the extent of the bandwidth over-allocation is considered to be reasonable when offset by the ability to perform independent response time analysis and feasibility assessment for individual activities allocated to shared resources.

In terms of change sensitivity, the RBA implementation scheme is superior to the priority-based schemes investigated. The following observations have been made :

- for the priority-based schemes, the prioritisation mechanism implemented on each resource provides a limited degree of protection against changes since, in general, a change in the period of one activity will affect lower priority activities but not higher priority activities. Hence, a transaction-wide change will typically impact on all other transaction with activities allocated to, and assigned lower priority values on, the shared resources in question;
- for the RBA implementation scheme, all impact is felt in terms of the final bandwidth allocation figures (since any increased response times will still remain within specified deadlines) - hence, it is merely a question of checking that sufficient resource bandwidth is available to deal with the changes imposed.

Finally, a note regarding the sufficient-but-not-necessary test *FP-SbNN1*. The test was originally developed in an attempt to desensitise the response time analysis for the priority-based implementation by linking the response time of each activity λ_j to the *combined* utilisation and computation times of the set of higher priority activities. The hypothesis was that changes to individual activity timing attributes might create less of an impact when treated as a relative change in the combined set of computation times and utilisation requirements. This turns out not to be the case, however, for this particular form of SbNN test but further work on the development of desensitised SbNN tests may be appropriate given the lessons learned from this assessment exercise.

CHAPTER 8. Extensions to the Rate-Based Execution Model and Cyclic Implementation Scheme

In this section, the RBA rate-based execution model and cyclic implementation scheme are extended to cater for :

- prioritised activities;
- synchronous communication.

In each case, the GAP task set is extended to illustrate and assess the extensions proposed.

8.1 Prioritised Activities

In this section, it is shown that the RBA rate-based execution model and cyclic implementation scheme can co-exist alongside a static priority-based scheduling regime to provide a flexible three-tier run-time execution model as follows :

- *high priority* activities that execute according to a static priority-based regime;
- RBA-compliant activities that execute according the cyclic RBA implementation scheme, subject to interference from the set of high priority activities;
- *low priority* activities that execute according to a static priority-based regime, subject to interference from the set of high priority activities and the set of RBA-compliant activities.

The motivation for this combined scheme is two-fold. Firstly, the high priority 'band' can be used to schedule activities with short response time requirements compared to their minimum inter-arrival time without incurring bandwidth over-allocation. Secondly, the low priority band can be used to execute activities in the bandwidth that is over-allocated by the RBA implementation scheme (plus any remaining spare capacity of the resource), thus reclaiming over-allocated bandwidth. This combined scheme provides a flexible implementation approach that can be applied on a per-system basis to achieve an appropriate balance between implementation efficiency and partitioning of the timing model.

8.1.1 Introducing High Priority Activities - Dealing with Interference Effects

Activities executed in the high priority band will execute according to a static priority-based scheduling regime (in accordance with their relative priorities) and always in preference to activities in the RBA band. Hence, high priority activities can be verified by static priority-based response time analysis as given in section 2.4. The RBA rate-based execution model and cyclic implementation scheme must be extended, however, to cater for interference delays resulting from the execution of high priority activities.

The rate-based execution model can be adapted to recognise interference effects using an analogous approach to that for response time analysis for static priority-based scheduling

solutions. The solution is simply to add a worst-case interference time to the *predicted* worst-case response time or, analogously, to subtract the interference delay from the *required* worst-case response time (deadline). Adapting equation *(4-24)* thus gives the required minimum execution rate for an activity $\lambda_{i,..,k}$ subject to worst-case interference delay $I_{i,..,k}$ as follows :

$$v_{i,..,k} = \frac{C_{i,..,k}}{R_{i,..,k} - I_{i,..,k}} \qquad (8\text{-}1)$$

Or, in the resource-oriented notation introduced in chapter 6, the required minimum execution rate for an activity λ_j subjected to a worst-case interference delay I_j is stated as follows :

$$v_j = \frac{C_j}{R_j - I_j} \qquad (8\text{-}1a)$$

For an activity λ_j that shares a resource with a set of high priority activities $\{\lambda_k; k = 1,..,nH\}$, I_j equates to the interference term stated in the response time expression for static priority-based scheduling - equation *(2-7)*. Hence, given the timing attributes for the set of high priority activities, I_j can be determined as follows :

$$I_j = \sum_{k=1}^{nH} \left\lceil \frac{R_j + J_k^{in}}{T_k} \right\rceil C_k \qquad (8\text{-}2)$$

Introducing high priority activities and interference into the cyclic implementation scheme has two effects (the scheme is otherwise unchanged). Firstly, the initial assignment of Δ is now subject to the constraint $\Delta \leq \min_j(R_j - I_j)$, where I_j denotes the maximum time that the activity λ_j can be prevented from executing due to the execution of prioritised activities. Secondly, for the general case of activity execution via a cyclic server $\lambda^S(\delta^S,N^S)$, the expression for determining the normalised response time for λ_j, equation *(6-12)*, is adapted as follows :

$$R_j^{\Delta} = \left\lfloor \frac{R_j - I_j}{N^S \Delta} \right\rfloor N^S \Delta \qquad (8\text{-}3)$$

This updated model is now applied to the GAP task set to examine the impact of interference effects on bandwidth allocation and feasibility.

8.1.2 Example

Consider the extension of the GAP task set to include a new activity λ_0 with the following timing attributes :

j	C_j	T_j	R_j
0	3	200	5

Under the RBA rate-based execution model, the minimum execution rate for λ_0 would be calculated as $v_0 = 0.6$. This corresponds to a bandwidth allocation of 60% which is excessive in comparison to its true utilisation requirement of 1.5%. This occurs due to the fact that the response time requirement for λ_0 is very short in comparison to its minimum inter-arrival time. Hence, consider the execution of λ_0 as a high priority activity. The full set of results can be seen in Appendix A3; the main findings are summarised below.

For the single high priority activity λ_0, a worst-case response time of 3 is given by static priority-based response time analysis and so the activity is feasible. For the set of RBA activities, the first step is to calculate the maximum interference delay imposed by the high priority activity. From equation *(8-2)*, the following values are obtained :

$$\{I_j; j = 1,..,16\} = \{3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 15, 15\}$$

Note that all input jitter values referred to in equation *(8-2)* are set to zero since there is no input jitter specified (or implied) in the GAP timing requirements - all activities are time-triggered. The extent of the effect of non-zero input jitter (in the high priority activity) depends on the source from which this is produced. If the source of the jitter is a precedence relationship with another priority-based activity, then the interference calculation for the given RBA activity is ultimately dependent upon the source activity, which could be allocated to a different processor. In such a case, the analysis of each RBA activity is independent from each other RBA activity but dependent on the analysis of the high priority activity and its predecessor (and possibly *its* predecessors). In general, the analysis chain can be terminated wherever response time analysis can be performed for an activity independently from others allocated to the same processor (or communication medium). Hence, there is a trade-off between using prioritised activities to improve responsiveness and reduce bandwidth over-allocation and allowing the scope of the response time analysis to increase. Whilst the analysis of the system should not become holistic to the extent that all activities in the system are involved, since the RBA implementation scheme will always introduce partitions within the timing model, the use of jitter-controlled high priority activities is perhaps the best compromise when applying this composite scheme.

Applying the rate-based execution model now gives the set of minimum execution rates shown in table 8-1. For comparison, the table gives values determined with and without interference effects present - the notation v_j^I is used to denote the value *with* interference. These figures show the extent to which RBA activities are affected by the introduction of the high priority activity. Whilst the impact is not great, all RBA activities are affected by the presence of the prioritised activity and so, in practical applications of this implementation scheme, prioritised activities would need to be allocated to resources before RBA activities in order to maintain ‘evolvability’ of the RBA cyclic schedule.

j	v_j^I	v_j
1	0.0909	0.0800
2	0.2273	0.2000
3	0.0270	0.0250
4	0.0638	0.0600
5	0.1064	0.1000
6	0.1429	0.1356
7	0.1169	0.1125
8	0.0260	0.0250
9	0.0515	0.0500
10	0.0051	0.0050
11	0.0152	0.0150
12	0.0051	0.0050
13	0.0051	0.0050
14	0.0152	0.0150
15	0.0010	0.0010
16	0.0010	0.0010
Σ	0.9004	0.8351

Table 8-1 : GAP Example - RBA Minimum Execution Rates (With and Without Interference)

Executing the RBA activities according to the cyclic implementation scheme, the choice of Δ is now subject to the constraint $\Delta \leq \min_j(R_j - I_j)$, giving an upper limit of 22 - a value of 18 is chosen for illustration. Cyclic servers are then assigned at 2Δ (for activities λ_3, λ_4, λ_5, λ_7 and λ_{14}), 10Δ (for activities λ_{10}, λ_{11}, λ_{12} and λ_{13}) and 50Δ (for activities λ_{15} and λ_{16}), with all other activities being executed by the base level server. This gives rise to the following bandwidth allocation figures for the set of RBA activities :

	U_j	v_j	v_j^Δ	v_j^δ
Σ	0.8351	0.9004	1.0476	1.1189

The final bandwidth allocation figure for direct execution of RBA activities, *ie.* ignoring spare server capacity, is 111.89%, compared to a true utilisation requirement of 83.51%. The required server capacities can be determined using the approach described in section 6.3. Equation *(6-19)* gives the following set of server execution times within Δ (including the base level server) :

$$\{ \delta_i^S ;\ i = 1,..,4\} = \{\lceil 12/1 \rceil, \lceil 15/2 \rceil, \lceil 6/10 \rceil, \lceil 2/50 \rceil\} = \{12, 8, 1, 1\}$$

Equation *(6-20)* then gives a value of $\Psi = 22/18 = 1.2222$ which corresponds to a total bandwidth requirement of 122.22%. It was suggested in section 6.3, however, that the bandwidth requirement of a lower rate server can, in certain cases, be completely absorbed within the spare capacity of a higher rate server and so be eliminated from the total bandwidth calculation as given by equation *(6-20)*. Looking more closely at the server periods and utilisation figures for this example, this actually turns out to be the case - the bandwidth requirement of the server $\lambda_4^S(1,50)$ can be completely absorbed within the spare capacity of the higher rate server $\lambda_3^S(1,10)$, since the latter is over-allocated by 4 units of execution time every 10Δ. Hence, the bandwidth requirement of the server $\lambda_4^S(1,50)$ can be eliminated from the total bandwidth calculation, giving a final value of $\Psi = 21/18 = 1.1667$ which corresponds to a total bandwidth requirement of 116.67%.

Whilst this indicates that the complete set of RBA activities is not schedulable on a single processor, the set can easily be reduced to one which is schedulable on a single processor without re-analysis by removing activities until the final bandwidth requirement is no more than 100%, since the removal of a given RBA activity will not affect the execution rate or feasibility of other RBA activities nor any high priority activity. The only implication is for the RBA activities that are removed in that a different interference delay will apply (if any at all) when re-allocated to a different processor, dependent of the presence of high priority tasks on the new processor.

8.1.3 Introducing Low Priority Activities - Reclaiming Over-Allocated Bandwidth

It has been seen in the series of examples thus far that bandwidth over-allocation occurs during the application of the target-independent rate-based execution model and also the cyclic implementation scheme :

- in the target-independent RBA rate-based execution model due to the calculation of activity execution rates based on response time requirements rather than minimum inter-arrival times;
- in the cyclic implementation scheme due to the need for a common cycle time and integer execution times within this cycle time (or some multiple of the cycle time when cyclic servers are used).

This motivates the consideration of bandwidth reclamation via the execution of activities outside the RBA implementation scheme according to a static priority-based regime within the bandwidth that is allocated but unused by RBA activities. This new set of priority-based activities is referred to as ‘low priority’ since none of these can pre-empt any RBA activity nor any high priority activity. The RBA implementation scheme itself does not actually require modification. The low priority activities can be guaranteed (or rejected) by adapting

the response time analysis for static priority-based scheduling, as described in section 2.4. Given a set of high priority activities $\{\lambda_k; k = 1,..,nH\}$, a set of RBA activities $\{\lambda_j; j = 1,..,n\}$ and a set of low priority activities $\{\lambda_i; i = 1,..,nL\}$, the following response time analysis can be stated for a given low priority activity λ_l :

$$R_l = C_l + \sum_{k=1}^{nH} \left\lceil \frac{R_l + J_k^{in}}{T_k} \right\rceil C_k + \sum_{j=1}^{n} \left\lceil \frac{R_l + J_j^{in}}{T_j} \right\rceil C_j + \sum_{i=1}^{l-1} \left\lceil \frac{R_l + J_i^{in}}{T_i} \right\rceil C_i \qquad (8\text{-}4)$$

The difference between this expression and the response time expression for static priority-based scheduling given in equation *(2-7)* is merely notational - the interference term is decomposed into three 'bands' to reflect the composite nature of the scheme. The three interference terms could be combined into a single term that covers all activities that are executed in preference to each low priority activity λ_l.

8.1.4 Example

Consider the GAP task set extended by the introduction of a set of low priority activities subject to deadline monotonic priority assignment (removing the high priority activity λ_0 introduced in the previous example) :

i	C_i	T_i	D_i	U_i
1	3	200	150	0.015
2	8	200	180	0.04
3	25	500	400	0.05
4	4	500	450	0.008
5	15	1000	800	0.015

Table 8-2 : GAP Example - Low Priority Activity Timing Requirements

Equation *(8-4)* gives the following worst-case response times for the set of low priority activities :

i	R_i	*Feasible*
1	140	✔
2	148	✔
3	384	✔
4	388	✔
5	789	✔

Table 8-3 : GAP Example - Low Priority Activity Response Times

All low priority activities are feasible since all response times are less than their corresponding deadline requirements. The total bandwidth requirement for the set of low priority activities is 12.8%. This has effectively been reclaimed from the over-allocated

bandwidth associated with the set of RBA activities, whose true requirement is 83.5% but final allocation is 97.5% (or, including spare server capacity, 100%).

8.2 Synchronous Communication

An often studied technique for supporting communication between tasks on a single processor is the use of 'synchronous' shared data resources - shared data resources that require mutually exclusive access and the CPU resource must be held simultaneously. In such cases, a worst-case *blocking* delay must be determined for each task that needs to access such a resource - this time must bound the delay that the task can experience when it requires access to the resource when it is already held by another task. Consequently, this form of communication tends to introduce inter-dependencies into the response time analysis of tasks that share access to the same communication resources. Furthermore, when resource access is permitted to restrict pre-emption on the CPU resource, these inter-dependencies can potentially affect the whole task set. Hence, this form of communication does not naturally fit into the partitioned approach of RBA. It is interesting to consider the precise impact on the RBA model, however, with a view to supporting limited use of this form of communication.

The impact of blocking delays on the target-independent rate-based execution model can be accounted for in the same way that interference delays were accounted for in section 8.1. Hence, the required minimum execution rate for an activity $\lambda_{i,..,k}$ that is subject to a worst-case blocking delay $B_{i,..,k}$ (and worst-case interference delay $I_{i,..,k}$) can thus be stated by extending equation *(8-1)* as follows :

$$v_{i,..,k} = \frac{C_{i,..,k}}{R_{i,..,k} - I_{i,..,k} - B_{i,..,k}} \qquad (8\text{-}5)$$

Or, in the resource-oriented notation, the required minimum execution rate for an activity λ_j subjected to a worst-case blocking factor B_j (and worst-case interference delay I_j) can be stated by extending equation *(8-1a)* as follows :

$$v_j = \frac{C_j}{R_j - I_j - B_j} \qquad (8\text{-}5a)$$

Regarding the cyclic implementation scheme, two changes are required (the scheme is otherwise unaffected). Firstly, the initial assignment of Δ is now subject to the constraint $\Delta \le \min_j(R_j - I_j - B_j)$. Secondly, for the general case of activity execution via a cyclic server $\lambda^S(\delta^S,N^S)$, the expression for determining the normalised response time for λ_j, equation *(8-3)*, is extended as follows :

$$R_j^{\Delta} = \left\lfloor \frac{R_j - I_j - B_j}{N^S \Delta} \right\rfloor N^S \Delta \qquad (8\text{-}6)$$

The above analysis assumes that worst-case blocking times can be bounded but does not prescribe any particular method for ensuring that this is the case. Such a method must be provided as part of the implementation scheme and would typically involve a set of rules that govern access to the shared (set of) synchronous resource(s) involved. The issue is discussed to an extent below but, more generally, this represents an area for further work.

Three possible approaches can be immediately identified :

- implement the scheduling of shared synchronous resources according to the RBA rate-based execution model - this would allow activities that access shared synchronous resources to be pre-empted over the CPU resource by any other activity allocated to the same processor (even those that share synchronous resources) but lead to prolonged blocking times, since blocking delays would be proportional to execution rates;
- apply Mok's non-pre-emptive *critical sections* approach [Mok83], executing all synchronous resource accesses atomically - this would lead to short blocking delays but remove the partitioning in the scheduling solution, since any synchronous resource access could potentially block any other activity allocated to the processor;
- execute synchronous resource accesses non-pre-emptively within groups of activities that share synchronous resource(s), allowing other activities to pre-empt as required - this is the preferred solution since the partitioning of the scheduling solution is only compromised for activities that share synchronous resources.

For this preferred scheme, the blocking delay associated with activity λ_j that shares (any number of) synchronous resources with m other activities $\{\lambda_j; j = 1,..,m\}$ is given by :

$$B_j = \sum_{k=1}^{m} \left\lceil \frac{R_j + J_k^{in}}{T_k} \right\rceil \beta_k \qquad (8\text{-}7)$$

where β_k is the duration of the synchronous resource access (or *blocking factor*).

8.2.1 Example

The following synchronous communication characteristics have been described for the GAP task set [Punnekkat97a, Tindell94] involving four shared resources {A, B, C, D} :

Resource Id.	*Blocking Factor*	*User Tasks*
A	5	8, 14
B	4	5, 9
C	3	2, 8
D	15	10, 14

Table 8-4 : GAP Example - Synchronous Communication Requirements

The full set of analysis results for this example can be seen in Appendix A4 but the main findings are summarised below. Firstly, equation *(8-7)* gives the following worst-case blocking times :

$\{B_j; j = 1,..,16\} = \{0, 3, 0, 0, 4, 0, 0, 17, 8, 15, 0, 0, 0, 30, 0, 0\}$

Applying the rate-based execution model gives the set of minimum execution rates for GAP task set shown in table 8-5. The table gives values determined with and without synchronous communication - the notation v_j^B is used to denote the value *with* blocking.

j	v_j^B	v_j
1	0.0800	0.0800
2	0.2273	0.2000
3	0.0250	0.0250
4	0.0600	0.0600
5	0.1087	0.1000
6	0.1356	0.1356
7	0.1125	0.1125
8	0.0317	0.0250
9	0.0543	0.0500
10	0.0054	0.0050
11	0.0150	0.0150
12	0.0050	0.0050
13	0.0050	0.0050
14	0.0176	0.0150
15	0.0010	0.0010
16	0.0010	0.0010
Σ	0.8852	0.8351

Table 8-5 : GAP Example - RBA Minimum Execution Rates (With and Without Blocking)

When constructing the RBA cyclic schedule for the GAP task set, the assignment of Δ is now subject to the constraint $\Delta \leq \min_j(R_j - I_j - B_j)$ which evaluates to 22. The value of 18 is maintained from the previous example, along with the same server periods 2Δ, 10Δ and 50Δ, giving rise to the following total bandwidth allocation figures for the set of RBA activities :

	U_j	v_j	v_j^{Δ}	v_j^{δ}
Σ	0.8351	0.8852	1.0610	1.1189

The final bandwidth allocation figure for direct execution of RBA activities, *ie.* ignoring spare server capacity, is 111.89%, compared to a true utilisation requirement of 83.51%. The required server capacities can again be determined using the approach described in section 6.3. Equation *(6-19)* gives the following set of server execution times within Δ (including the base level server) :

$$\{ \delta_i^S ; i = 1,..,4\} = \{\lceil 12/1 \rceil, \lceil 15/2 \rceil, \lceil 6/10 \rceil, \lceil 2/50 \rceil\} = \{12, 8, 1, 1\}$$

Equation *(6-20)* then gives a value of Ψ = 22/18 = 1.2222 which corresponds to a total bandwidth requirement of 122.22%. Again, the bandwidth requirement of the server $\lambda_4^S(1,50)$ can be completely absorbed within the spare capacity of the higher rate server $\lambda_3^S(1,10)$ and can be eliminated from the total bandwidth calculation, giving a final value of Ψ = 21/18 = 1.1667 which corresponds to a total bandwidth requirement of 116.67%.

8.3 Further RBA Implementation Options

The proposed cyclic scheme assumes fixed times of execution of activities and servers within a cycle time of fixed duration Δ. Allowing these times to vary according to run-time demand, including the actual cycle time itself, will produce an implementation that is equivalent to a *weighted round robin* (WRR) scheme with $T_j = J_j^{in} + a_j$ - see section 2.3.3. Hence, like WRR, the cycle time Δ represents the maximum run-time cycle duration and the final run-time schedule is work-conserving. This may be attractive in terms of improving average-case response times with no adverse impact on worst-case responsiveness nor overall bandwidth allocation. The disadvantage is a significant increase in response jitter, since the responsiveness of each activity is no longer merely a function of its own resource usage requirements but becomes dependent upon the varying resource demands of every other activity.

Further run-time flexibility could be added by allowing the order of execution within Δ to vary. This is detrimental, however, in terms of bandwidth allocation. This can be seen by briefly examining the impact on the cyclic implementation model. In the worst-case, an activity can now be delayed at its release time by up to twice its server cycle time (this is slightly pessimistic - the exact duration is the worst-case jitter that the server can experience between successive Δ cycles). For the general case of activity execution via a cyclic server $\lambda^S(\delta^S, N^S)$, the normalised response time for λ_j is thus given by extending equation *(8-6)* as follows :

$$R_j^{\Delta} = \left\lfloor \frac{R_j - B_j - I_j - N^S\Delta}{N^S\Delta} \right\rfloor N^S\Delta \qquad (8\text{-}8)$$

This will give a reduced value for the normalised response time compared to equation *(8-6)* and so, given the same worst-case computation time for λ_j, an increased bandwidth allocation will result. As before, bandwidth reclaiming could be applied to reduce the impact of the over-allocation.

Finally, it would be interesting to examine a number of existing scheduling and communication schemes from the perspective of supporting the goals of RBA at the implementation stage. For example, it was observed in section 2.5.3.3 that a number of proposed *dynamic* scheduling schemes provide fine-grained bandwidth allocation *without* fine-grained time-slicing. These include the constant utilisation server [Deng97], total bandwidth server [Spuri96], proportional share [Stoica96], weighted fair queuing [Demers89] and virtual clock [Zhang91, Yau96]. Also, it would be interesting to consider the configurability of different TDMA-based communication protocols to directly support the RBA cyclic implementation scheme.

8.4 Generic RBA Implementation Model

Abstract implementation models, like the ISO-OSI communication network model [Day83] and the military IMA software model [Wake96, Kemp00] provide useful guidance for structuring the implementation of complex computing systems. Such a model, as depicted in figure 8-1, can be described to providing guidance for the implementation of RBA-compliant scheduling and communication schemes.

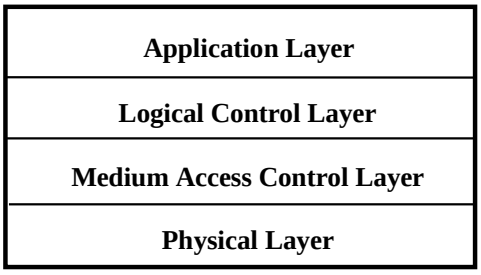


Figure 8-1 : Generic Implementation Model

The *application layer* simply embodies the set of RBA activities and their associated timing and resource requirements and precedence relationships. No timing requirements are dispatched within this layer.

The *logical control layer* implements any jitter control mechanisms that are required to ensure that activity release jitter at run-time is maintained within the bounds specified in the offline RBA end-to-end timing model. Recall that jitter control mechanisms are required when there is no explicit execution rate control for ensuring best-case response time requirements are met. Similarly, it is necessary to ensure that specified minimum inter-arrival times are enforced for both sporadic and periodic activities so that transient overload

can not be induced. Hence, the timing requirements dispatched by this layer are, for a given activity λ_j, the maximum input jitter J_j^{in} and the minimum inter-arrival time a_j.

The *medium access control layer* provides the scheduling function, *eg.* according to the RBA cyclic implementation scheme, for controlling access to one or more *synchronous* resources. The computing platform is thus partitioned into synchronous resource groups, *ie.* resources whose access control methods are inter-dependent due to either :

- the physical nature of the underlying resources, *eg.* where access to one resource is controlled from another, as in the case where mutually exclusive access to synchronous shared memory is governed by the CPU scheduling scheme;
- the requirements of the application layer, *eg.* where a single activity can simultaneously lock multiple resources.

The timing requirements dispatched by the medium access control layer are, for a given activity λ_j, the maximum computation time C_j, minimum execution rate v_j and worst-case response time R_j (plus the corresponding best-case requirements when best-case behaviour is implemented explicitly in the medium access control layer rather than in the logical control layer by jitter control mechanisms).

The *physical layer* embodies the set of physical computing resources. When the RBA cyclic implementation scheme is applied, the timing requirements dispatched by physical layer are, for a given activity λ_j, the maximum execution time δ_j per interval $N^S\Delta$ and, for synchronous communication resources, the associated maximum blocking time β.

8.5 Summary

A combined cyclic/priority-based implementation scheme has been described that allows priority-based activities to execute outside the RBA implementation scheme such that :

- high priority activities can execute (when ready to do so) in preference to any RBA activity;
- RBA activities to be maintain their independence in the presence of interference from high priority activities;
- low priority activities to execute in the bandwidth over-allocation associated with the set of RBA activities (and any remaining processor bandwidth that is otherwise unused).

The cyclic implementation scheme has also been extended to cover synchronous communications between RBA activities such that the partitioning of the timing model is only compromised for activities that share synchronous resources - independent activities continue to be independently analysable. The target-independent RBA rate-based execution model has been extended to account for both interference and blocking effects on RBA activities.

Possible future work areas can be identified as follows :

- investigate the use static priority-based bandwidth servers for execution of prioritised activities alongside an RBA implementation scheme;
- investigate a range of existing bandwidth-based scheduling and communication schemes from the perspective of supporting the goals of RBA at the implementation stage;
- investigate methods for implementing synchronous communication in conjunction with specific RBA implementation schemes.

Finally, for comparison, some notable results have previously been documented for the GAP task set (not including the low priority activities introduced in section 8.1.4) as follows :

- the task set is *unschedulable* on a single processor by static priority-based scheduling with rate monotonic priority assignment [Locke91] - λ_7 and all lower priority tasks are unschedulable, although note that λ_0 is assigned highest priority, *ie.* outside the rate monotonic assignment, in order to meet its short response time requirement;
- the task set is *unschedulable* on a single processor by static priority-based scheduling with deadline monotonic priority assignment [Tindell94], although only λ_9 misses its deadline;
- the task set is *schedulable* on a single processor by static priority-based scheduling with optimal priority assignment [Audsley93b, Tindell94];
- the task set, without its synchronous communication requirements, is *schedulable* on a single processor by the EGPS dynamic priority-based scheduling scheme [Yau98].

This implies that the GAP task-set has been shown to be on the borderline between schedulable and unschedulable on a single processor for other scheduling schemes as was found to be the case for the RBA implementation schemes investigated here.

CHAPTER 9. Research Contribution and Further Work

This section summarises the research contribution of the thesis as follows :

- problem description;
- central hypothesis;
- research contribution - reservation-based timing analysis;
- further work.

9.1 Problem Description

The timing properties of a real-time system can be verified by the formulation of mathematical *timing analysis* models that can be applied to predict the extremes of the temporal behaviour of the system. Timing analysis models for modern applications involving distributed systems, however, are prone to being *holistic* [Tindell94] - the system can be analysed as a whole but cannot, in general, be analysed in part. This arises due to the combined impact of functional and physical integration effects. Consequently, the impact of localised changes in timing attributes can be wide-spread and potentially enforce re-analysis of the entire system. This is problematic on the grounds of potential re-work and associated costs for many practical real-time system applications where changes are frequent during system development and upgrades are required in-service. The problem is compounded by current industrial trends, typified by programmes in the civil and military aerospace domains, towards more evolvable development processes and integrated target architectures.

In view of this, the following requirements have been identified for timing analysis models for future distributed real-time systems :

- the ability to model the timing behaviour of parts of the system independently from other parts and, correspondingly, limit the scope of re-analysis required in the event of localised changes;
- the ability to assess the timing behaviour of the system throughout the development process, based on the latest information available at each stage.

9.2 Central Hypothesis

The development of an appropriate timing analysis model to meet the requirements stated above has been pursued throughout the course of this thesis. The central hypothesis has been stated as follows :

> *An appropriately partitioned timing analysis model can support the stated development objectives for future real-time systems : evolvable development of real-time applications for a distributed computing platform that is functionally and physically integrated.*

9.3 Research Contribution - Reservation-Based Timing Analysis

A timing analysis model has been constructed to meet the requirements stated above by placing constraints on the means by which functional dependencies (execution ordering and data sharing constraints) and/or physical dependencies (resource sharing constraints) are represented in the model and subsequently implemented in the final system. A *physical partitioning* approach, whereby constraints are imposed on the resource sharing policies for the system, has been identified as being in line with the true partitioning requirements for the system which relate to the independent development and analysis of functionally-related application components. The structure of the timing model is thus driven by the structure of the specific applications involved rather than that of the underlying target hardware architecture. This serves as a basis for applying the timing analysis model throughout the life-cycle of the system, starting much earlier in the development process than is usual (typically during the implementation stage). The following two-stage method has been proposed for performing timing analysis throughout the system life-cycle :

- *Abstract timing analysis* :
  Abstract (implementation-independent) timing analysis is performed during the definition and decomposition stages of system development by adopting an abstract model of system resources. The net result being a set of best- and worst-case *guarantees* regarding the timing behaviour of the system subject to a set of *obligations* being met by the final implementation.

- *Target-specific timing analysis* :
  Target-specific (implementation-dependent) timing analysis is performed during the implementation and integration stages of system development, the aim being to demonstrate that the set of obligations imposed by the abstract timing analysis model have actually been met in practice.

The timing analysis model is constructed and applied as follows :

- *Representation and refinement of system timing properties* :
  The end-to-end timing characteristics of the system are captured, developed and analysed in terms of *transactions*. The transaction model is hierarchical, in the form of an acyclic, directed, nested graph, reflecting the need to support an evolving definition of the system during development. The leaf nodes of the graph capture the concurrent processing and communication elements within the transaction, termed *activities*; non-leaf nodes are referred to as *nested transactions*. The edges (or arcs) of the graph capture the precedence and nesting relationships within the transaction. The parameters via which timing behaviour is represented and observed are the same for a single activity, a group of related activities, a nested transaction and a system level transaction, thus providing a highly composable and scaleable timing model.

- *End-to-end (system level) timing analysis* :
  The overall timing behaviour of the system at a given stage of evolution can be assessed by analysing the current set of transaction graphs. The end-to-end timing behaviour of each transaction can be assessed independently from other transactions in the system (including those which may eventually share processing and/or communication resources) and from the eventual target hardware implementation details. End-to-end delays and jitter are determined by a depth-first traversal of the transaction graph and accounting for localised (activity level) delays, precedence relationships and nesting relationships. Localised delay values can be specified directly in the form of *budgets* or otherwise determined via some form of localised timing analysis model. The end-to-end model also allows jitter-controlled transactions to be defined and phase relationships to be determined between multiple events within and between transactions.

- *Localised (activity level) timing analysis* :
  Activity level delays and jitter can be determined in a target-independent manner based on an abstract, rate-based model of execution. When the target platform implementation details (hardware and software) are known, these results can be verified on a per-activity basis. This supports a smooth transition between abstract timing analysis and target-dependent analysis at the appropriate time for each part of the system under development. The rate-based execution model also provides a degree of independence from the final integration details of the system, including the precise form of run-time scheduling solution.

A cyclic scheduling scheme has been described that directly implements the RBA rate-based execution model. This allows the run-time scheduling solution to be derived directly from an RBA target-independent timing analysis model for the system without compromising the original timing requirements. The implementation scheme maintains the partitioning of the response time analysis at the target-dependent stage and hence provides a basis for subsequent changes to be made with minimal impact. The final verification of the system timing behaviour (and re-analysis after changes) can be performed on a per-activity basis by simple comparison between target-dependent and target-independent resource requirements. Target-specific end-to-end response times are verified automatically by inference from the end-to-end timing model, *ie.* without the need to perform target-specific end-to-end timing analysis. The cyclic implementation scheme has been shown to map readily to a reduced form of the APEX scheduling model [AEEC96] that has been proposed for future civil IMA systems.

The same basic timing model is thus applicable throughout the system life-cycle, providing a consistent approach to analysis rather than adopting a series of separate method/tool-specific or implementation-specific techniques. With the latter approach, there is no guarantee that the resulting combination of techniques will deliver the required properties in the overall system timing analysis model. Further consistency is provided by RBA in that

diverse system resource types, specifically processing and communication resources, are treated analogously in the timing model.

The cost of delivering the proposed practical system engineering benefits has been recognised as reduced responsiveness and effectiveness of bandwidth utilisation when compared to a work-conserving static priority-based scheme. The extent of these drawbacks has been shown to be reasonable for practical applications and can actually be managed on an application-specific basis as part of an overall trade-off with the ability to perform independent timing analysis. In recognition of this trade-off, a hybrid implementation scheme has been described that supports prioritised activities along-side RBA activities. This scheme allows *high priority* activities to execute in preference to RBA activities when the bandwidth allocation arising from the cyclic implementation scheme is inefficient and responsiveness requirements are more important than independent timing analysis. The scheme also allows over-allocated bandwidth from the RBA implementation scheme to be reclaimed by the execution of *low priority* activities. Feasibility analysis can be performed for all three types of activity in the combined scheme.

9.4 Further Work

A number of areas for possible future work have been identified, primarily relating to the development of alternative implementation schemes for RBA, as follows :

- consider the development of a bandwidth-based scheme for performing the allocation stage of the process (the mapping of concurrent RBA activities to physical resources) that exploits the partitioned nature of the end-to-end analysis and the target-independent nature of the rate-based execution model;
- extend the RBA cyclic implementation scheme to support ‘nested’ or ‘hierarchical’ bandwidth servers as a means of making more efficient use of the available bandwidth (reducing the extent of bandwidth over-allocation);
- investigate the provision of an RBA-compliant implementation scheme using priority-based bandwidth servers and potentially use of these to extend the mapping to the civil IMA APEX scheduling model;
- investigate use of priority-based protocols for synchronous communication between activities on a single processor, in conjunction with the RBA cyclic implementation scheme;
- consider the development of RBA-compliant implementation schemes for different packet-switched communication network protocols.

REFERENCES

[Abdelzaher99] Abdelzaher, T., Dawson, S., Feng, WC., Jahanian, F., Johnson, S., Mehra, A., Mitton, T., Shaikh, A., Shin, K., Wang, Z., Zou, H., Bjorkland, M., Marron, P., "ARMADA Middleware and Communication Services," *Real-Time Systems Journal* 16(2/3), 1999.

[Abeni98] Albeni, L., Buttazzo, G., "Integrating Multimedia Applications in Hard Real-Time Systems," *Proceedings of Real-Time Systems Symposium*, 1998.

[Abeni98a] Albeni, L., Lipari, G., Buttazzo, G., "A Bandwidth-Based Server for Multimedia Applications," *Proceedings of Euromicro Workshop on Real-Time Systems (Work in Progress)*, 1998.

[AEEC91] "ARINC Specification 651 : Design Guidance for Integrated Modular Avionics," *Airlines Electronic Engineering Committee*, 1991.

[AEEC94] "ARINC Specification 629-3 : IMA Multi-Transmitter Databus," *Airlines Electronic Engineering Committee*, 1994.

[AEEC97] "ARINC Specification 653 : Avionics Application Software Standard Interface," *Airlines Electronic Engineering Committee*, 1997.

[AFAL87] "Architecture Specification for PAVE PILLAR Avionics," *US Air Force Avionics Laboratory* report SPA90099001A, 1987.

[ANSI95] "Ada95 Reference Manual," *ANSI/ISO/IEC* Std. 8652:1995, 1995.

[Arvind91] Arvind, K., Stankovic, JA., Ramamritham, K., "A Local Area Network Architecture for Communication in Distributed Real-Time Systems," *Real-Time Systems Journal* 3(2), 1991.

[Atlas98] Atlas, A., Bestavros, A., "Statistical Rate Monotonic Scheduling," *Proceedings of IEEE Real-Time Systems Symposium*, 1998.

[Audsley91] Audsley, NC., Burns, A., Richardson, MF., Wellings, AJ., "Hard Real-Time Scheduling : The Deadline Monotonic Approach," *Proceedings of IEEE Workshop on Real-Time Operating Systems and Software*, 1991.

[Audsley93] Audsley, NC., Burns, A., Tindell, KW., "The End of the Line for Static Cyclic Scheduling?," *Proceedings of 5th Euromicro Workshop on Real-Time Systems*, 1993.

[Audsley93a] Audsley, NC., Burns, A., Wellings, AJ., "Deadline Monotonic Scheduling Theory and Applications," *Control Engineering Practice* 1(1), 1993.

[Audsley93b] Audsley, NC., "Flexible Scheduling of Hard Real-Time Systems,"*Department of Computer Science, University of York, UK*, DPhil thesis YCST-93/07, 1993.

[Audsley93c] Audsley, NC., Burns, A., Richardson, MF., Tindell, K., Wellings, A., "Applying New Scheduling Theory to Static Priority Pre-emptive Scheduling," *Software Engineering Journal* 8(5), 1993.

[Audsley94] Audsley, NC., Davis, RI., Burns, A., "Mechanisms for Enhancing Flexibility and Utility in Hard Real-Time Systems," *Proceedings of IEEE Real-Time Systems Symposium*, 1994.

[Audsley95] Audsley, NC., Burns, A., Davis, RI., Tindell, KW., Wellings, AJ., "Fixed Priority Pre-emptive Scheduling: An Historical Perspective," *Real-Time Systems Journal* 8(3), 1995.

[Audsley96] Audsley, NC., Wellings, AJ., "Analysing APEX Applications," *Proceedings of IEEE Real-Time Systems Symposium*, 1996.

[Baker89] Baker, TP., Shaw, A., "The Cyclic Executive Model and Ada," *Real-Time Systems Journal* 1(1), 1989.

[Baruah96] Baruah, S., Cohen, N., Plaxton, CG., Varvel, D., "Proportionate Progress: A Notion of Fairness in Resource Allocation," *Algorithmica* 15(6), 1996.

[Bate97] Bate, I., Burns, A., "Timing Analysis of Fixed Priority Real-Time Systems with Offsets," *Euromicro Workshop on Real-Time Systems*, 1997.

[Bate99] Bate, IJ., "Scheduling and Timing Analysis for Safety-Critical Real-Time Systems," *Department of Computer Science, University of York, UK*, DPhil thesis YCST-99/04, 1999.

[Benedict98] Benedict, DM., "The F-22 Common Integrated Processor: A Case Study in Obsolescence Management," *Proceedings of ERA Avionics Conference and Exhibition*, 1998.

[Berger95] Berger, SJ., "ARINC 629 Digital Communication System Application on the 777 and Beyond," *Proceedings of ERA Avionics Conference and Exhibition*, 1995.

[Bernat01] Bernat, G., Burns, A., "Multiple Servers and Capacity Sharing for Implementing Flexible Scheduling," *Real-Time Systems Journal* 22(pp.49-75), 2001.

[Boisseau95] Boisseau, M., Demange, M., Munier, JM., "An Introduction to ATM Technology," *International Thomson Publishing*, 1995.

[Borky98] Borky, JM., Lachenmaier, RN., Messing, P., Frink, A. "Architectures for Next Generation Military Avionics Systems," *Proceedings of IEEE Aerospace Conference*, 1998.

[Burns91] Burns, A., Wellings, AJ., "Criticality and Utility in the Next Generation," *Real-Time Systems Journal* 3(4), 1991.

[Burns91a] Burns, A., Lister, AM., "A Framework for Building Dependable Systems," *IEEE Computer Journal* 34(pp.173-181), 1991.

[Burns94] Burns, A., McDermid, JA., “Real-Time Safety-Critical Systems: Analysis and Synthesis,” *Software Engineering Journal* 9(6), 1994.

[Burns95] Burns, A., “Pre-emptive Priority-Based Scheduling: An Appropriate Engineering Approach,” in Son, SH., “Advances in Real-Time Systems,” *Prentice Hall*, 1995.

[Burns95a] Burns, A., Audsley, NC., Wellings, AJ., “Real-Time Distributed Computing,” *Proceedings of IEEE Future Trends in Distributed Computing*, 1995.

[Burns97] Burns, A., Wellings, AJ., “Synchronous Sessions and Fixed Priority Scheduling,” *Journal of Systems Architecture* 44(pp107-118), 1997.

[Burns01] Burns, A., Wellings, AJ., “Real-Time Systems and Programming Languages,” *Addison Wesley Longmain*, 2001.

[Buttazzo95] Buttazo, GC., Stankovic, JA., “Adding Robustness in Dynamic Pre-emptive Scheduling,” in Fussell, S., Malek, M., “Responsive Computer Systems: Steps Toward Fault-Tolerant Real-Time Systems”, *Kluwer Academic*, 1995.

[Buttazzo97] Buttazzo, GC., “Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications,” *Kluwer Academic*, 1997.

[Chen90] Chen, M., Lin, K., “Dynamic Priority Ceiling: A Concurrency Control Protocol for Real-Time Systems,” *Real-Time Systems Journal* 2(1), 1990.

[Chang85] Chang, HY., Livny, M., “Priority in Distributed Systems,” *Proc of IEEE Real-Time Systems Symposium*, 1985.

[Cheng86] Cheng, S., Stankovic, JA., Ramamritham, K., “Dynamic Scheduling of Groups of Tasks with Precedence Constraints in Distributed, Hard Real-Time Systems,” *Proceedings of IEEE Real-Time Systems Symposium*, 1986.

[Chetto89] Chetto, H., Chetto, M., “Some Results of the Earliest Deadline Scheduling Algorithm,” *IEEE Transactions on software Engineering* 15(10), 1989.

[Chetto90] Chetto, H., Silly, M., Bouchentouf, T., “Dynamic Scheduling of Real-Time Tasks under Precedence Constraints,” *Real-Time Systems Journal* 2(3), 1990.

[Damm89] Damm, A., Reisinger, J., Schwabl, W., Kopetz, H., “The Real-Time Operating System of MARS,” *ACM Operating Systems Review Special Issue*, 1989.

[Davis93] Davis, RI., Tindell, KW., Burns, A., “Scheduling Slack Time in Fixed Priority Pre-emptive Systems,” *Proceedings of IEEE Real-Time Systems Symposium*, 1993.

[Day83] Day, JD., Zimmermann, H., “The OSI Reference Model,” *Proceedings of the IEEE* 71(12), 1983.

[Demers89] Demers, A., Keshav, S., Shenker, S., “Analysis and Simulation of a Fair Queuing Algorithm,” *Proceedings of ACM SIGCOMM*, 1989.

[Deng97] Deng, Z, Liu, JWS., "Scheduling Real-Time Applications in an Open Environment," *Proceedings of Real-Time Systems Symposium*, 1997.

[Deng99] Deng, Z, Liu, JWS., Zhang, L., Mouna, S., Frei, A., "An Open Environment for Real-Time Applications," *Real-Time Systems Journal* 16(2/3), 1999.

[Dertouzos74] Dertouzos, ML., "Control Robotics: The Procedural Control of Physical Processes," *Proceedings of IFIP Congress*, 1974.

[Dertouzos89] Dertouzos, ML., Mok, AKL., "Multiprocessor Online Scheduling of Hard Real-Time Tasks," *IEEE Transactions on Software Engineering* 15(12), 1989.

[DiNatale94] DiNatale, M., Stankovic, JA., "Dynamic End-to-End Guarantees in Distributed Real-Time Systems," *Proceedings of IEEE Real-Time Systems Symposium*, 1994.

[DO-178B] "Software Considerations in Airborne Systems and Equipment Certification," *RTCA-EUROCAE* DO-178B/ED-12B, 1992.

[DoD99] "DoD Technical Reference Model (Version 1.1 - Coordination Draft)," *US Department of Defence*, 1999.

[DoD99a] "DoD Joint Technical Architecture (Interim Version 3.0 Draft 2)," *US Department of Defence*, 1999.

[DS 00-55a] "Defence Standard 00-55: Requirements for Safety Related Software in Defence Equipment (Part 1: Requirements)," *UK Ministry of Defence*, 1997.

[DS 00-55b] "Defence Standard 00-55: Requirements for Safety Related Software in Defence Equipment (Part 2: Guidance)," *UK Ministry of Defence*, 1997.

[DS 00-71] "Defence Standard 00-71: Obsolescence Management," *UK Ministry of Defence*, 2001.

[Edwards94] Edwards, RA., "ASAAC Phase 1 Harmonised Concept Summary," *Proceedings of ERA Avionics Conference and Exhibition*, 1994.

[Ermedahl97] Ermedahl, A., Hansson, H., Sjodin, M., "Response Time Guarantees in ATM Networks," *Proceedings of Real-Time Systems Symposium*, 1997.

[Fabian95] Fabian, GC., Keen, PJ., "MoD UK Implementation of Integrated Modular Avionics," *Proceedings of Royal Aeronautical Society One Day Conference on Integrated Avionics*, 1995.

[Ferrari90] Ferrari, D., Verma, D., "A Scheme for Real-Time Channel Establishment in Wide-Area Networks," *IEEE Journal on Selected Areas in Communications* 8(3), 1990.

[Ferrari95] Ferrari, D., Gupta, A., Ventre, G., "Distributed Advance Reservation of Real-Time Connections," *Proceedings of International Workshop on Network and Operating System Support for Digital Audio and Video*, 1995.

[Field97] Field, D., *et al*, "Software Techniques for IMA - System Management, System Security and Blueprints," *Proceedings of ERA Avionics Conference and Exhibition*, 1997.

[Fineberg67] Fineberg, MS., Serlin, O., "Multiprogramming for Hybrid Computations," *Proceedings of AFIPS Fall Joint Computer Conference*, 1967.

[Gerber95] Gerber, R., Hong, S., Saksena, M., "Guaranteeing Real-Time Requirements with Resource-Based Calibration of Periodic Processes," *IEEE Transactions on Software Engineering* 21(7), 1995.

[Golestani91] Golestani, S., "A Framing Strategy for Congestion Management," *IEEE Journal on Selected Areas in Communications* 9(7), 1991.

[Graham69] Graham, RL., "Bounds on Multiprocessing Timing Anomalies," *SIAM Journal of Applied Mathematics* 17(2), 1969.

[Graham79] Graham, R., Lawer, EL., Lenstra, JK., and Rinnooy-Kan, AHG. "Optimisation and Approximation in Deterministic Sequencing and Scheduling: A Survey," *Annals of Discrete Mathematics* 5(pp287-326), 1979.

[Grigg94] Grigg, A., Field, D., "The Impact of Interchangeability Requirements on Operating Systems for Modular Avionics," *Proceedings of ERA Avionics Conference and Exhibition*, 1994.

[Grigg96] Grigg, A., Audsley, NC., "Timing Analysis of the ARINC 629 Databus for Real-Time Applications," *Proceedings of ERA Avionics Conference and Exhibition*, 1996; re-published in *Microprocessing and Microprogramming* 21(1), 1997

[Gutierrez98] Gutierrez, J., Garcia, J., Harbour, MG., "Best-Case Analysis for Improving the Worst-Case Schedulability Test for Distributed Hard Real-Time Systems," *Proceedings of Euromicro Workshop on Real-Time Systems*, 1998.

[Harter84] Harter, PK., "Response Times in Level Structured Systems," *Department of Computer Science, University of Colorado, USA*, report CU-CS-269-84, 1984.

[Harter87] Harter, PK., "Response Times in Level Structured Systems," *ACM Transactions on Computer Systems* 5(3), 1987.

[Harbour91] Harbour, MG., Klein, MH., Lehoczky, JP., "Fixed Priority Scheduling of Periodic Tasks with Varying Execution Priority," *Proceedings of IEEE Real-Time Systems Symposium*, 1991.

[Hutchesson98] Hutchesson, SG., "Multi-Integrity Level Software on Uniprocessor Systems," *Department of Computer Science, University of York, UK*, MSc thesis, 1998.

[IMechE96] "Eurofighter 2000: Technology for the 21st Century," *IMechE Seminar Series*, 1996.

[Jeffay92a] Jeffay, K., “Scheduling Sporadic Tasks with Shared Resources in Hard Real-Time Systems,” *Proceedings of IEEE Real-Time Systems Symposium*, 1992.

[Jeffay92b] Jeffay, K., Stone, DL., Smith, FD., “Kernel Support for Live Digital Audio and Video,” *Computer Communications* 15(6), 1992.

[Jeffay95] Jeffay, K., Bennett, D., “A Rate-Based Execution Abstraction for Multimedia Computing,” *Proceedings of International Workshop on Network and Operating System Support for Digital Audio and Video*, in *Lecture Notes in Computer Science* 1018, 1995.

[Jeffay99] Jeffay, K., Goddard, S., “A Theory of Rate-Based Execution,” *Proceedings of IEEE Real-Time Systems Symposium*, 1999.

[Jeffay00] Jeffay, K., Lamastra, G., “A Comparative Study of the Realisation of Rate-Based Computing Services in General Purpose Operating Systems,” *Proceedings of IEEE International Conference on Real-Time Computing Systems and Applications*, 2000.

[Jensen85] Jensen, ED, Locke, CD., Tokuda, H., “A Time-Value-Driven Scheduling Model for Real-Time Operating Systems,” *Proceedings of IEEE Real-Time Systems Symposium*, 1985.

[Jensen92] Jensen, ED., “Asynchronous Decentralised Real-Time Computer Systems,” *Proceedings of NATO Advanced Study Institute, Springer-Verlag*, 1992.

[Jensen97] Jensen, ED., “Eliminating the Hard/Soft Real-Time Dichotomy,” *Computing and Control Engineering Journal* 8(1), 1997.

[Jones96] Jones, MB., Barrera, JS., Forin, A., Leach, PJ., Rosu, D., Rosu, MC., “An Overview of the Rialto Real-Time Architecture,” *Proceedings of ACM SIGOPS Workshop*, 1996.

[Jones97] Jones, MB., Rosu, D., Rosu, MC., “CPU Reservations and Time Constraints: Efficient, Predictable Scheduling of Independent Activities,” *Proceedings of ACM Symposium on Operating System Principles*, 1997.

[Jones01] Jones, MB., Regehr, J., Saroiu, S., “Two Case-Studies in Predictable Application Scheduling Using Rialto/NT,” *Proceedings of IEEE Real-Time Technology and Applications Symposium*, 2001.

[Joseph86] Joseph, M., Pandya, PK., “Finding Response Times in a Real-Time System,” *BCS Computer Journal* 29(5), 1986.

[JSF97] “Joint Strike Fighter Avionics Architecture Definition (Version 2),” *JSF Program Office*, 1997.

[Kalmanek99] Kalmanek, CR., Kanakia, H., Keshav, S., “Rate-Controlled Servers for Very High Speed Networks,” *Proceedings of IEEE Telecommunications Conference*, 1999.

[Kandlur91] Kandlur, DD., Shin, KG., Ferrari, D., “Real-Time Communication in Multi-Hop Networks,” *Proceedings of International Conference on Distributed Computing Systems*, 1991.

[Kemp00] Kemp, J., Wake, A., Williams, B., “The Development of the ASAA Software Architecture,” *Proceedings of ERA Avionics Conference and Exhibition*, 2000.

[Kopetz89] Kopetz, H., Damm, A., Koza, C., Mulazzani, M., Schwabl, W., Senft., C., Zainlinger, R., “Distributed Fault-Tolerant Real-Time Systems: The MARS Approach,” *IEEE Micro* 9(1), 1989.

[Kopetz92] Kopetz, H., “Sparse Time versus Dense Time in Distributed Real-Time Systems,” *Proceedings of IEEE International Conference on Distributed Computing Systems*, 1992.

[Kopetz94] Kopetz, H., Grunsteidl, G., “TTP - A Protocol for Fault Tolerant Real-Time Systems,” *IEEE Computer* 24(1), 1994.

[Kopetz95] Kopetz, H., “Why Time-Triggered Architecture will Succeed in Large Hard Real-Time Systems,” *Proceedings of IEEE Future Trends in Distributed Computing Systems*, 1995.

[Kopetz95a] Kopetz, H., Nossal, R., “The Cluster Compiler - A Tool for the Design of Time-Triggered Real-Time Systems,” *ACM SIGPLAN Workshop on Languages, Compilers and Tools for Real-Time Systems*, 1995.

[Kopetz97] Kopetz, H., “Real-Time Systems Design Principles for Distributed Embedded Applications,” *Kluwer Academic*, 1997.

[Kopetz98] Kopetz, H., “The Time-Triggered Model of Computation,” *Proceedings of IEEE Real-Time Systems Symposium*, 1998.

[Kuo98] Kuo, TW., Yang, WR., Lin, KJ., “EGPS: A Class of Real-Time Scheduling Algorithms Based on Processor Sharing,” *Proceedings of Euromicro Workshop on Real-Time Systems*, 1998.

[Kuo99] Kuo, TW., Li, CH., “A Fixed Priority-Driven Open Environment for Real-Time Applications,” *Proceedings of IEEE Real-Time Systems Symposium*, 1999.

[Kurose84] Kurose, JF., Schwartz, M., Yemini, Y., “Multi-Access Protocols and Time Constrained Communication,” *ACM Computing Surveys* 16(1), 1984.

[Lachenmaier97] Lachenmaier, RL., “Modern Fighter Mission Avionics: The Joint Strike Fighter Avionics Architecture,” *Proceedings of AGARD Conference on Future Aerospace Technology in Service of the Alliance*, 1997.

[Lamport78] Lamport, L., “Time, Clocks and Ordering of Events in a Distributed System,” *Communications of the ACM* 21(7), 1978.

[Lee96] Lee, C., Rajkumar, R., Mercer, C., “Experiences with Processor Reservation and Dynamic QOS in Real-Time Mach,” *Proceedings of Multimedia Japan*, 1996.

[Lee96a] Lee, C., Yoshida, K., Mercer, C., Rajkumar, R., “Predictable Communication Protocol Processing in Real-Time Mach,” *Proceedings of Real-Time Systems and Applications Symposium*, 1996.

[Lehoczky87] Lehoczky, JP., Sha, L., Strosnider, JK., “Enhanced Aperiodic Responsiveness in Hard Real-Time Environments,” *Proceedings of IEEE Real-Time Systems Symposium*, 1987.

[Lehoczky92] Lehoczky, JP., Ramos-Thuel, S., “An Optimal Algorithm for Scheduling Soft Aperiodic Tasks in Fixed Priority Pre-emptive Systems,” *Proceedings of IEEE Real-Time Systems Symposium*, 1992.

[Lehoczky97] Lehoczky, JP., “Real-Time Network Theory,” *Proceedings of Real-Time Systems Symposium*, 1997.

[LESC94] “Space Generic Open Avionics Architecture (SGOAA) Standard Specification,” *Lockhead Engineering & Sciences Company* report NASA-CR-118290/LESC-30354, 1994.

[Leung80] Leung, JYT., Merrill, ML., “A Note on Pre-emptive Scheduling of Periodic Real-Time Tasks,” *Information Processing Letters* 11(3), 1980.

[Leung82] Leung, JYT., Whitehead, J., “On the Complexity of Fixed Priority Scheduling of Periodic, Real-Time Tasks,” *Performance Evaluation (Netherlands)* 2(4), 1982.

[Liu73] Liu, CL., Layland, JW., “Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment,” *Journal of the ACM* 20(1), 1973.

[Liu00] Liu, JWS., “Real-Time Systems,” *Prentice Hall*, 2000.

[Locke86] Locke, CD., “Best-Effort Decision Making for Real-Time Scheduling,” *Computer Science Department, Carnegie-Mellon University, USA*, PhD thesis CMU-CS-86-134, 1986.

[Locke91] Locke, CD., Vogel, DR., Mesler, TJ., “Building a Predictable Avionics Platform in Ada: A Case Study,” *Proceedings of IEEE Real-Time Systems Symposium*, 1991.

[Locke92] Locke, CD., “Software Architecture for Hard Real-Time Applications: Cyclic Executives vs. Fixed Priority Executives,” *Real-Time Systems Journal* 4(1), 1992.

[Mercer94] Mercer, CW., Savage, S., Tokuda, H., “Processor Capacity Reserves: OS Support for Multimedia Applications,” *Proceedings of IEEE International Conference on Multimedia Computing Systems*, 1994.

[Metcalfe76] Metcalfe, RM., Boggs, DR., “Ethernet: Distributed Packet Switching for Local Computer Networks,” *Communications of the ACM* 19(7), 1976.

[MIL-STD-1553] “Military Standard - Digital Time Division Command/Response Multiplex Data Bus: MIL-STD-1553,” *US Department of Defence*, 1986.

[Moir99] Moir, M., Ramamurthy, S., “Pfair Scheduling of Fixed and Migrating Periodic Tasks on Multiple Resources,” *Proceedings of IEEE Real-Time Systems Symposium*, 1999.

[Mok78] Mok, AKL., Dertouzos, ML., “Multiprocessor Scheduling in a Hard Real-Time Environment,” *Proceedings of Texas Conference on Computer Systems*, 1978.

[Mok83] Mok, AKL., “Fundamental Design Problems of Distributed Systems for the Hard Real-Time Environment,” *Lab. of Computer Science, Massachusetts Institute of Technology, USA*, PhD thesis MIT/LCS/TR-297, 1983.

[Molle81] Molle, ML., “Extensions and Unifications of the Multiple Access Communication Problem,” *Computer Science Department, University of California (LA), USA*, report CSD-810730, 1981.

[Multedo98] Multedo, G., Jibb, D., Angel, G., “ASAAC Phase II Programme Progress on the Definition of Standards for the Core Processing Architecture of Future Military Aircraft,” *Proc. of ERA Avionics Conference*, 1998.

[Nicholson98] Nicholson, M. “Selecting a Topology for Safety-Critical Real-Time Control Systems”, *Department of Computer Science, University of York*, DPhil thesis YCST-98/08, 1998.

[Nilsson98] Nilsson, U., Streiffert, S., Torne, A., “Detailed Design of Avionics Contol Software,” *Proceedings of IEEE Real-Time Systems Symposium*, 1998.

[Nissanke01] Nissanke, N., Leulseged, A., Chillara, S., “A Framework for Probabilistic Analysis of Multiprocessor Scheduling Environments,” *Proceedings of IEEE Real-Time Embedded Systems Workshop*, 2001.

[Parekh93] Parekh, AK., Gallager, RG., “A Generalised Processor Sharing Approach to Flow Control in Integrated Services Networks: The Single Node Case,” *IEEE Transactions on Networking* 1(3), 1993.

[Parekh94] Parekh, AK., Gallager, RG., “A Generalised Processor Sharing Approach to Flow Control in Integrated Services Networks: The Multiple Node Case,” *IEEE Transactions on Networking* 2(2), 1994.

[Punnekkat97] Punnekkat, S., Davis, R., Burns, A., “Sensitivity Analysis of Real-Time Task Sets,” *Proceedings of Asian Computing Science Conference*, 1997.

[Punnekkat97a] Punnekkat, S., “Schedulability Analysis for Fault Tolerant Real-Time Systems, “ *Department of Computer Science, University of York, UK*, DPhil thesis YCST-97/05, 1997.

[Raha95] Raha, A., Kamat, S., Zhao, W., “Guaranteeing End-to-End Deadlines in ATM Networks,” *Proceedings of IEEE Distributed Computing Systems Conference*, 1995.

[Raha96] Raha, A., Kamat, S., Zhao, W., “Admission Control for Hard Real-Time Connections in ATM LANs,” *Proceedings of IEEE INFOCOM*, 1996.

[Rajkumar89] Rajkumar, R., Sha, L., Lehoczky, JP., “An Experimental Investigation of Synchronisation Protocols,” *Proceedings of 6th IEEE Workshop on Real-Time Operating Systems and Software*, 1989.

[Ramamritham84] Ramamritham, K., Stankovic, JA., “Dynamic Task Scheduling in Distributed Hard Real-Time Systems,” *IEEE Software* 1(3), 1984.

[Ramamritham96] Ramamritham, K., “Dynamic Priority Scheduling,” in Joseph, M., “Real-Time Systems Specification, Verification and Analysis,” *Prentice Hall*, 1996.

[Rodd90] Rodd, MG., Zhao, GF., Izikowitz, I., “RTMMS - An OSI-Based Real-Time Messaging System,” *Real-Time Systems Journal* 2(3), 1990.

[Rosu97] Rosu, D., Schwan, K., Yalamanchili, S., Jha, R., “On Adaptive Allocation for Complex Real-Time Applications,” *Proceedings of Real-Time Systems Symposium*, 1997.

[SAE96] “Generic Open Architecture (GOA) Framework,” *Society of Automotive Engineers* report SAE AS4893, 1996.

[Saksena96] Saksena, M., Hong, S., “Resource-Concious Design of Distributed Real-Time Systems - An End-to-End Approach,” *Proceedures of IEEE Conference on Engineering of Complex Computer Systems*, 1996.

[Serlin72] Serlin, O., “Scheduling of Time-Critical Processes,” *Proceedings of AFIPS Spring Joint Computer Conference*, 1972.

[Sha87] Sha, L., Lehoczky, JP., Rajkumar, R., “Priority Inheritance Protocols : An Approach to Real-Time Synchronisation,” *Computer Science Department, Carnegie-Mellon University, USA*, report CMU-CS-87-181, 1987.

[Sha90] Sha, L., Rajkumar, R., Lehoczky, JP., “Priority Inheritance Protocols : An Approach to Real-Time Synchronisation,” *IEEE Transactions on Computers*, 39(9), 1990.

[Sha93] Sha, L., Sathaye, SS., “A Systematic Approach to Designing Distributed Real-time Systems” *IEEE Computer Journal* 26(9), 1993.

[Shin92] Shin, KG., Kandlur, DD., Kiskis, DL., Dodd, PS., Rosenberg, HA., Indiresan, A., “A Software Overview of HARTS: A Distributed Real-Time System,” *IEEE Software Journal* 9(5), 1992.

[Shin95] Shin, K., Chang, YC., “A Reservation-Based Algorithm for Scheduling both Periodic and Aperiodic Real-Time Tasks,” *IEEE Transactions on Computers* 44(12).

[Simpson90] Simpson, HR., “A Data Interaction Architecture for Real-Time Embedded Multiprocessor Systems,” *Proceedings of RAe Conference on Computing Techniques in Guided Flight*, 1990.

[Sprunt88] Sprunt, B., Lehoczky, J., Sha, L., “Exploiting Unused Periodic Time for Aperiodic Service Using the Extended Priority Exchange Algorithm,” *Real-Time Systems Journal* 1(1), 1989.

[Sprunt89] Sprunt, B., Sha, L., Lehoczky, J., “Aperiodic Task Scheduling for Hard Real-Time Systems,” *Real-Time Systems Journal* 1(1), 1989.

[Spuri96] Spuri, M., Buttazzo, G., “Scheduling Aperiodic Tasks in Dynamic Priority Systems,” *Real-Time Systems Journal* 10(pp179-210), 1996.

[Stankovic90] Stankovic, JA., “What is Predictability for Real-Time Systems,” *Real-Time Systems Journal* 2(4), 1990.

[Stankovic91] Stankovic, JA., Ramamritham, K., “The SPRING Kernel: A New Paradigm for Real-Time Systems,” *IEEE Software Journal* 8(3), 1991.

[Stankovic92a] Stankovic, JA., “Distributed Real-Time Computing: The Next Generation,” *Journal of the Society of Instrument and Control Engineers of Japan*, 1992.

[Stankovic93] Stankovic, JA., Ramamritham, K., “Advances in Real-Time Systems (chapter 2: Scheduling),” *IEEE Computer Society Press*, 1993.

[Stankovic95] Stankovic, JA., Ramamritham, K., “A Reflective Architecture for Real-Time Systems,” in Son, SH., “Advances in Real-Time Systems,” *Prentice Hall*, 1995.

[Stankovic98] Stankovic, JA., Spuri, M., Ramamritham, K., Buttazzo, GC., “Deadline Scheduling for Real-Time Systems: EDF and Related Algorithms,” *Kluwer Academic*, 1998.

[Stoica96] Stoica, I., Abdel-Wahab, H., Jeffay, K., Baruah, SK., Gehrke, JE., Plaxton, CG., “A Proportional Share Resource Allocation Algorithm for Real-Time Time-shared Systems,” *Proceedings of Real-Time Systems Symposium*, 1996.

[Stoica97] Stoica, I., Abdel-Wahab, H., Jeffay, “On the Duality between Resource Reservation and Proportional Share Resource Allocation,” *Proceedings of Multimedia Computing and Networking*, in *SPIE Proceedings Series* 3020(pp.207-214), 1997.

[Stoyenko96] Stoyenko, AD., Marlowe, TJ., Laplante, PA., “A Description Language for Engineering of Complex Real-time Systems,” *Real-Time Systems Journal* 11(3), 1996.

[Strosnider89] Strosnider, JK., Marchok, TE., “Responsive, Deterministic IEEE 802.5 Token Ring Scheduling,” *Real-Time Systems Journal*, 1(2), 1989.

[Tanenbaum96] Tanenbaum, AS., “Computer Networks (3rd edition),” *Prentice Hall*, 1996.

[Tindell92] Tindell, KW., Burns, A., Wellings, AJ., “Allocating Hard Real-Time Tasks: An NP-Hard Problem Made Easy,” *Real-Time Systems Journal* 4(2), 1992.

[Tindell94] Tindell, KW., “Fixed Priority Scheduling of Hard Real-Time Systems,” *Department of Computer Science, University of York, UK*, DPhil thesis YCST-94/03, 1994.

[Wake96] Wake, AS., Miller, PR., Moxon, P., Fletcher, MA., “Modular Avionics Operating System – Software Concept,” *Proceedings of ERA Avionics Conference and Exhibition*, 1996.

[Waldspurger94] Waldspurger, CA., Weihl, WE., “Lottery Scheduling: Flexible Proportional Share Resource Management,” *Proceedings of 1st Usenix Symposium on OS Design and Implementation*, 1994.

[Waldspurger94] Waldspurger, CA., Weihl, WE., “Stride Scheduling: Deterministic Proportional Share Resource Management,” *MIT Laboratory for Computer Science*, report MIT/LCS/TM-528, 1995.

[Yau96] Yau, DKY., Lam, SS., “Adaptive Rate Controlled Scheduling for Multimedia Applications,” *Proceedings of ACM Multimedia Conference*, 1996.

[Zhang91] Zhang, L., “VirtualClock: A New Traffic Control Algorithm for Packet-Switched Networks,” *ACM Transactions on Computer Systems* 9(2), 1991.

[Zhang99] Zhang, LY., Liu, JWS., Deng, Z., “Hierarchical Scheduling of Periodic Messages in an Open System,” *Proceedings of IEEE Real-Time Systems Symposium*, 1999.

[Zhao87] Zhao, W., Ramamritham, K., “Simple and Integrated Heuristic Algorithms for Scheduling Tasks with Time and Resource Constraints,” *Journal of Systems and Software* 7(3), 1987.

APPENDIX

A1. Delay and Jitter Inheritance - Special Cases

This appendix provides further details of the analytical framework proposed to model end-to-end timing behaviour in section 4.2.1. When the precise form of release function $Q_{i,..,k}$ involved in a many-to-one precedence or nesting relationship is known, this information can be used to reduce the pessimism of the accrued delay and jitter bounds compared to those determined by equations *(4-10)* to *(4-15)*.

The following relationships hold for the case where $Q_{i,..,k}$ is defined such that $\lambda_{i,..,k}$ is released upon completion of *any one* $\lambda_{i,..,j}$:

- *many-to-one precedence relationship* - when $\lambda_{i,..,j}$ is one of many predecessors to $\lambda_{i,..,k}$:

$$d_{i,..,k}^{in} = \min_{j}(d_{i,..,j}^{out}) \qquad (4\text{-}10a)$$

$$J_{i,..,k}^{in} = \min_{j}(d_{i,..,j}^{out} + J_{i,..,j}^{out}) - d_{i,..,k}^{in} \qquad (4\text{-}11a)$$

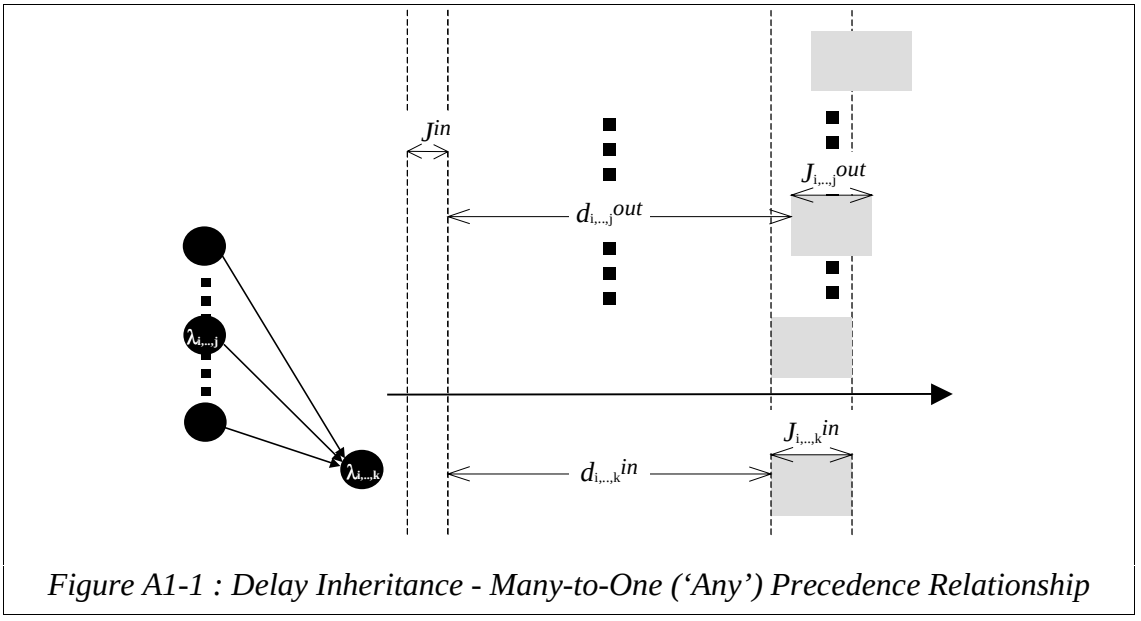


Figure A1-1 : Delay Inheritance - Many-to-One ('Any') Precedence Relationship

- *many-to-one nesting relationship (descending)* - when $\lambda_{i,..,j}$ is one of many parent transactions whose arrival is required to trigger that of the child activity $\lambda_{i,..,k}$:

$$d_{i,..,k}^{in} = \min_{j}(d_{i,..,j}^{in}) \qquad (4\text{-}12a)$$

$$J_{i,..,k}^{in} = \min_{j}(d_{i,..,j}^{in} + J_{i,..,j}^{in}) - d_{i,..,k}^{in} \qquad (4\text{-}13a)$$

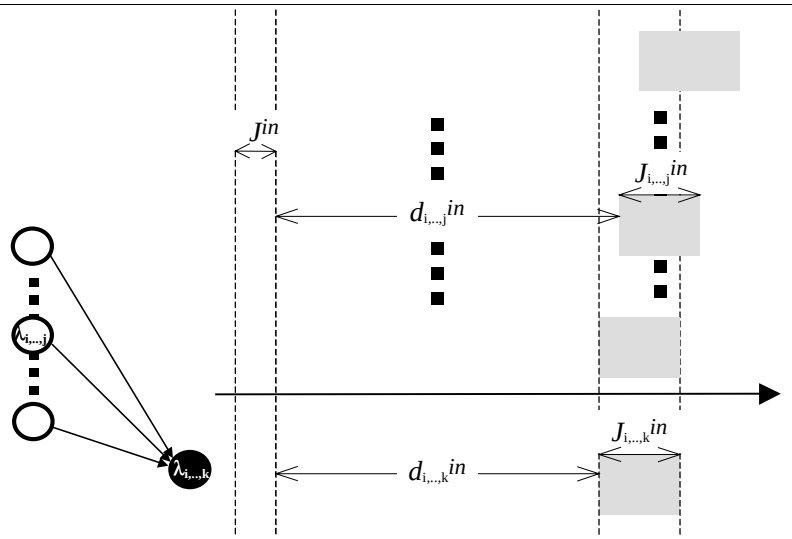


Figure A1-2 : Delay Inheritance - Many-to-One ('Any') Nesting Relationship (Descending)

- *many-to-one nesting relationship (ascending)* - when $\lambda_{i,..,j}$ is one of many child activities whose completion is required to trigger that of the parent transaction $\lambda_{i,..,k}$:

$$d_{i,..,k}^{out} = \min_{j}(d_{i,..,j}^{out}) \qquad \textit{(4-14a)}$$

$$J_{i,..,k}^{out} = \min_{j}(d_{i,..,j}^{out} + J_{i,..,j}^{out}) - d_{i,..,k}^{out} \qquad \textit{(4-15a)}$$

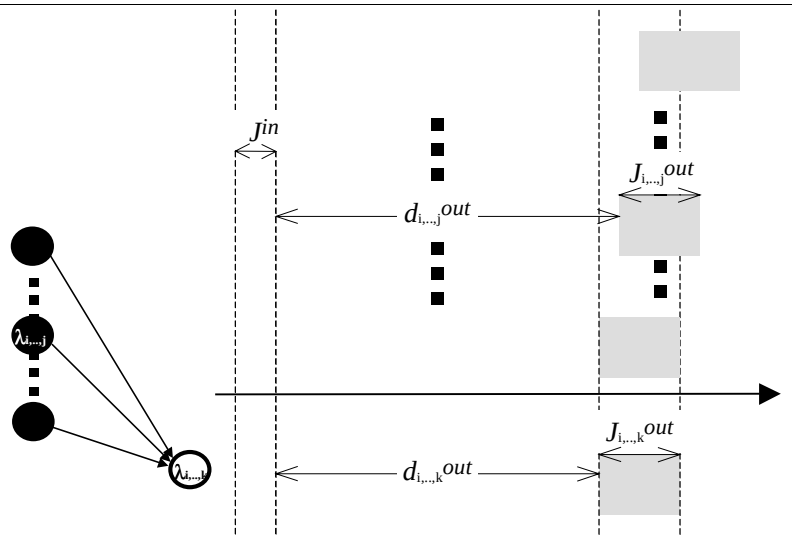


Figure A1-3 : Delay Inheritance - Many-to-One ('Any') Nesting Relationship (Ascending)

The following relationships hold for the case where $Q_{i,..,k}$ is defined such that $\lambda_{i,..,k}$ is released only upon completion of *all* $\lambda_{i,..,j}$:

- *many-to-one precedence relationship* - when $\lambda_{i,..,j}$ is one of many predecessors to $\lambda_{i,..,k}$:

$$d^{in}_{i,..,k} = \max_j(d^{out}_{i,..,j}) \qquad (4\text{-}10b)$$

$$J^{in}_{i,..,k} = \max_j(d^{out}_{i,..,j} + J^{out}_{i,..,j}) - d^{in}_{i,..,k} \qquad (4\text{-}11b)$$

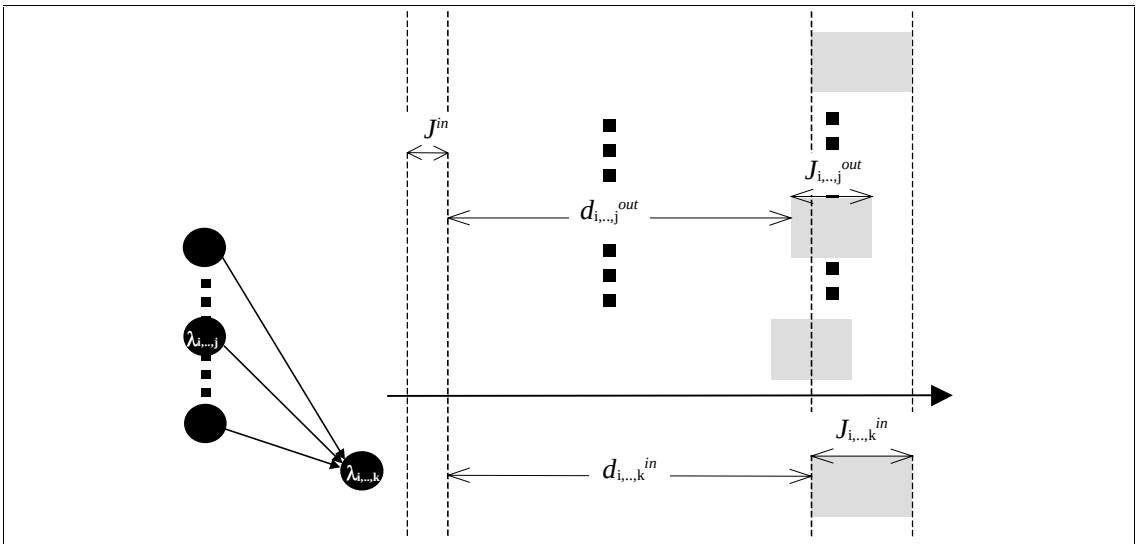


Figure A1-4 : Delay Inheritance - Many-to-One ('All') Precedence Relationship

- *many-to-one nesting relationship (descending)* - when $\lambda_{i,..,j}$ is one of many parent transactions whose arrival is required to trigger that of the child activity $\lambda_{i,..,k}$:

$$d^{in}_{i,..,k} = \max_j(d^{in}_{i,..,j}) \qquad (4\text{-}12b)$$

$$J^{in}_{i,..,k} = \max_j(d^{in}_{i,..,j} + J^{in}_{i,..,j}) - d^{in}_{i,..,k} \qquad (4\text{-}13b)$$

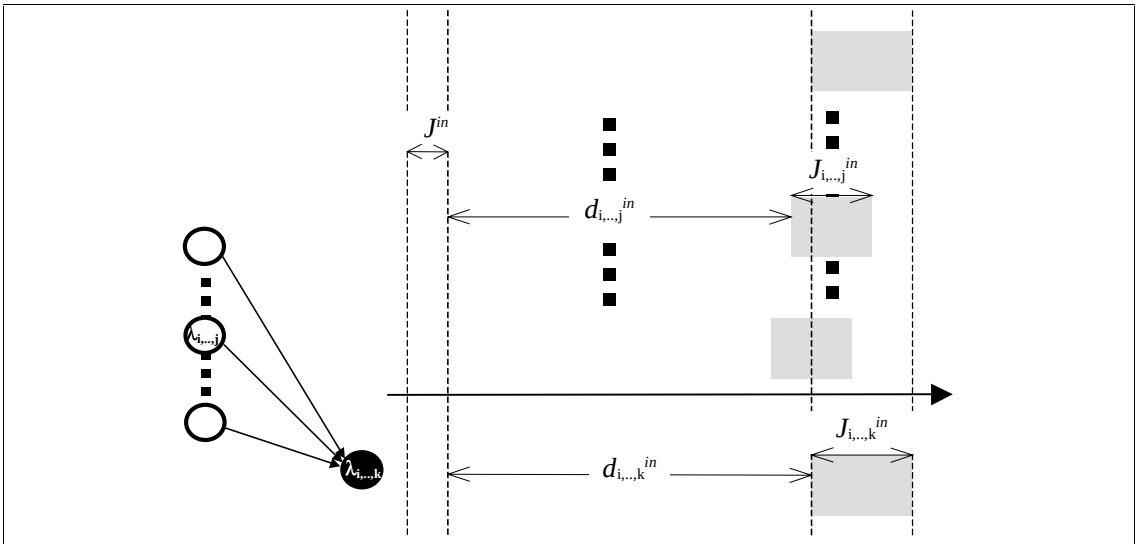


Figure A1-5 : Delay Inheritance - Many-to-One ('All') Nesting Relationship (Descending)

- *many-to-one nesting relationship (ascending)* - when $\lambda_{i,..,j}$ is one of many child activities whose completion is required to trigger that of the parent transaction $\lambda_{i,..,k}$:

$$d_{i,..,k}^{out} = \max_{j}(d_{i,..,j}^{out}) \qquad \textit{(4-14b)}$$

$$J_{i,..,k}^{out} = \max_{j}(d_{i,..,j}^{out} + J_{i,..,j}^{out}) - d_{i,..,k}^{out} \qquad \textit{(4-15b)}$$

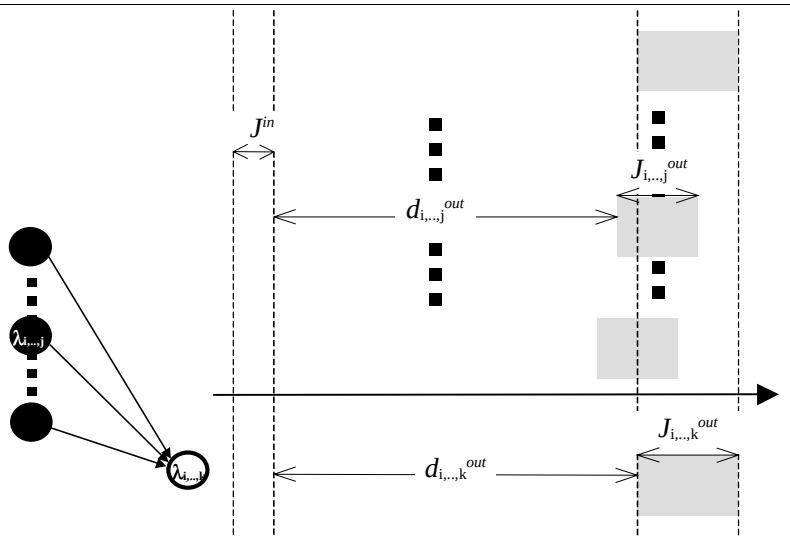


Figure A1-6 : Delay Inheritance - Many-to-One ('All') Nesting Relationship (Ascending)

A2. Derivation of Sufficient-but-Not-Necessary Test *FP-SbNN1*

The starting point for the derivation of *FP-SbNN1* is the exact feasibility test for the priority-based implementation given in equation *(2-7)* :

$$R_j = C_j + B_j + \sum_{\tau_k \in hp(\tau_j)} \left\lceil \frac{R_j}{T_k} \right\rceil C_k$$

The ceiling function in this expression can be eliminated by introducing a new term $\sigma_k < T_k$ such that the expression $\frac{R_j + \sigma_k}{T_k}$ equates to the same integer value, giving :

$$R_j = C_j + B_j + \sum_{\tau_k \in hp(\tau_j)} \left(\frac{R_j + \sigma_k}{T_k} \right) C_k$$

Since $\sigma_k < T_k$, this exact expression can be weakened to a sufficient-but-not-necessary form as follows :

$$R_j = C_j + B_j + \sum_{\tau_k \in hp(\tau_j)} \left(\frac{R_j + T_k}{T_k} \right) C_k$$

Expanding the summation term and defining the utilisation term $U_k = \frac{C_k}{T_k}$ gives :

$$R_j = C_j + B_j + \sum_{\tau_k \in hp(\tau_j)} R_j U_k + \sum_{\tau_k \in hp(\tau_j)} C_k$$

Rearranging, and introducing a ceiling function to guarantee an integer result, gives the sufficient-but-not-necessary test :

$$R_j = \left\lceil \frac{B_j + \sum_{\tau_k \in hpeq(\tau_j)} C_k}{1 - \sum_{\tau_k \in hp(\tau_j)} U_k} \right\rceil$$

A3. GAP Example - Introducing Prioritised Activities

The following tables and figures give the complete set of analysis results for the example used in section 8.1.2.

j	U_j	v_j	v_j^{Δ}	v_j^{δ}
1	0.0800	0.0909	0.1111	0.1111
2	0.2000	0.2273	0.2778	0.2778
3	0.0250	0.0270	0.0278	0.0278
4	0.0600	0.0638	0.0833	0.0833
5	0.1000	0.1064	0.1389	0.1389
6	0.1356	0.1429	0.1481	0.1667
7	0.1125	0.1169	0.1250	0.1389
8	0.0250	0.0260	0.0278	0.0556
9	0.0500	0.0515	0.0556	0.0556
10	0.0050	0.0051	0.0056	0.0056
11	0.0150	0.0152	0.0167	0.0167
12	0.0050	0.0051	0.0056	0.0056
13	0.0050	0.0051	0.0056	0.0056
14	0.0150	0.0152	0.0167	0.0278
15	0.0010	0.0010	0.0011	0.0011
16	0.0010	0.0010	0.0011	0.0011
Σ	0.8351	0.9004	1.0476	1.1189

Table A3-1 : GAP Example with Prioritised Activities (Execution Rate Parameters)

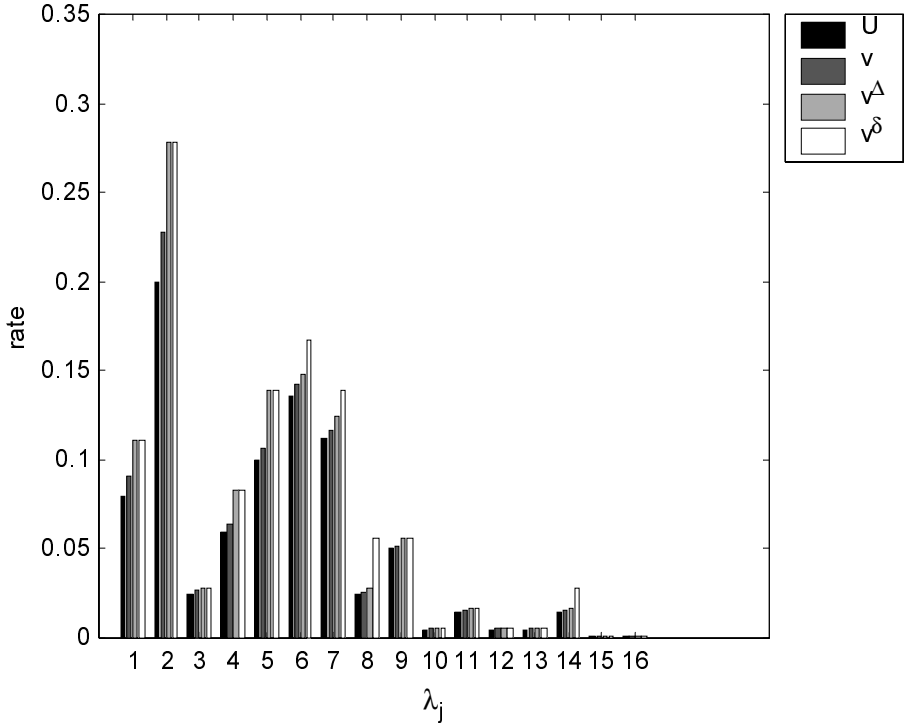


Figure A3-1 : GAP Example with Prioritised Activities (Execution Rate Parameters)

j	R_j	R_j^{Δ}	R_j^{δ}
1	25	18	18
2	25	18	18
3	40	36	36
4	50	36	36
5	50	36	36
6	59	54	54
7	80	72	72
8	80	72	36
9	100	90	90
10	200	180	180
11	200	180	180
12	200	180	180
13	200	180	180
14	200	180	108
15	1000	900	900
16	1000	900	900

Table A3-2 : GAP Example with Prioritised Activities (Response Time Parameters)

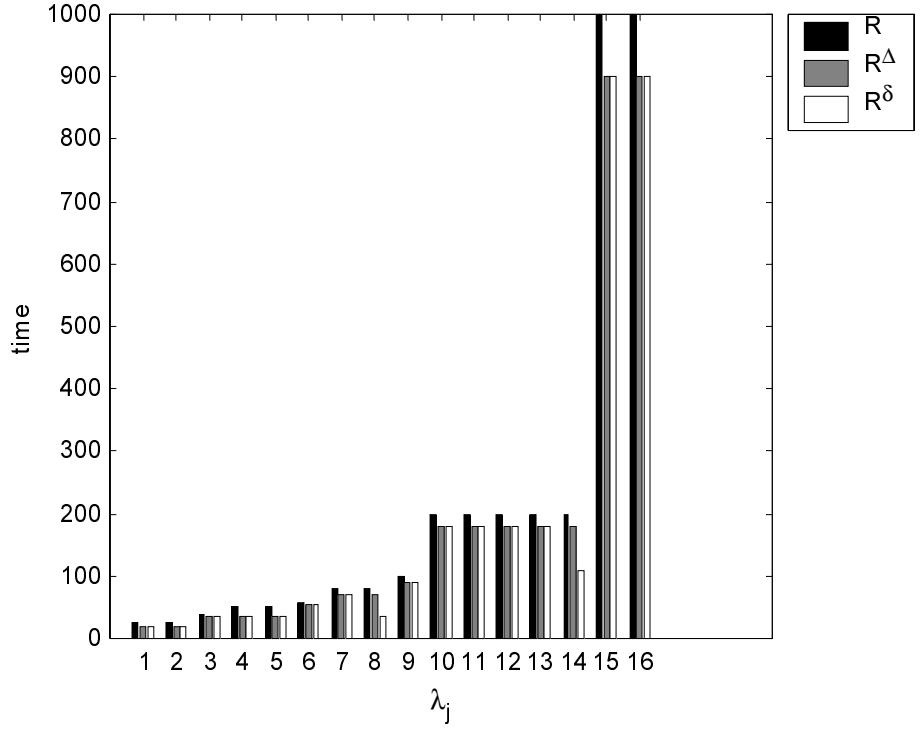


Figure A3-2 : GAP Example with Prioritised Activities (Response Time Parameters)

j	C_j	C_j^{δ}	δ_j
1	2	2	2
2	5	5	5
3	1	1	1
4	3	3	3
5	5	5	5
6	8	9	3
7	9	10	5
8	2	4	1
9	5	5	1
10	1	1	1
11	3	3	3
12	1	1	1
13	1	1	1
14	3	5	1
15	1	1	1
16	1	1	1

Table A3-3 : GAP Example with Prioritised Activities (Computation Time Parameters)

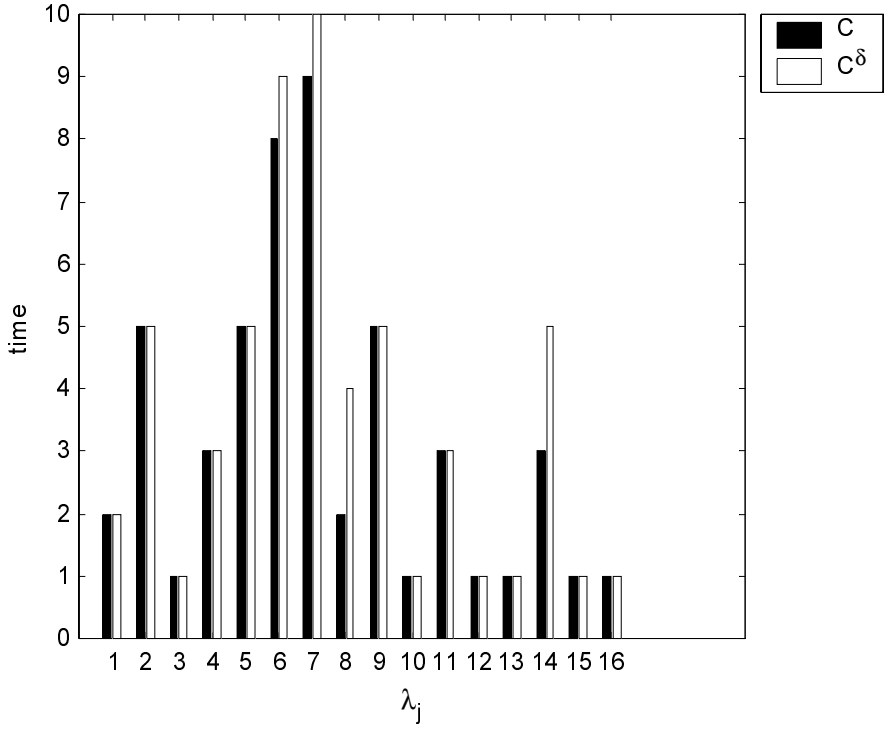


Figure A3-3 : GAP Example with Prioritised Activities (Computation Time Parameters)

A4. GAP Example - Introducing Synchronous Communication

The following tables and figures give the complete set of analysis results for the example used in section 8.2.1.

j	U_j	v_j	v_j^{Δ}	v_j^{δ}
1	0.0800	0.0800	0.1111	0.1111
2	0.2000	0.2273	0.2778	0.2778
3	0.0250	0.0250	0.0278	0.0278
4	0.0600	0.0600	0.0833	0.0833
5	0.1000	0.1087	0.1389	0.1389
6	0.1356	0.1356	0.1481	0.1667
7	0.1125	0.1125	0.1250	0.1389
8	0.0250	0.0317	0.0370	0.0556
9	0.0500	0.0543	0.0556	0.0556
10	0.0050	0.0054	0.0056	0.0056
11	0.0150	0.0150	0.0167	0.0167
12	0.0050	0.0050	0.0056	0.0056
13	0.0050	0.0050	0.0056	0.0056
14	0.0150	0.0176	0.0208	0.0278
15	0.0010	0.0010	0.0011	0.0011
16	0.0010	0.0010	0.0011	0.0011
Σ	0.8351	0.8852	1.0610	1.1189

Table A4-1 : GAP Example with Synchronous Communication (Execution Rate Parameters)

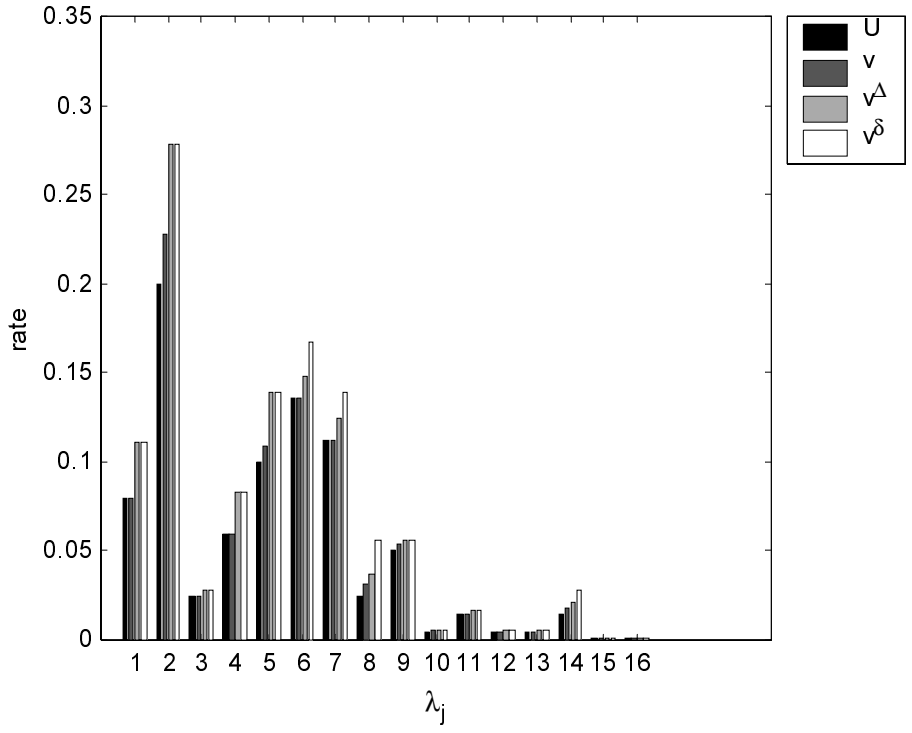


Figure A4-1 : GAP Example with Synchronous Communication (Execution Rate Parameters)

j	R_j	R_j^{Δ}	R_j^{δ}
1	25	18	18
2	25	18	18
3	40	36	36
4	50	36	36
5	50	36	36
6	59	54	54
7	80	72	72
8	80	54	36
9	100	90	90
10	200	180	180
11	200	180	180
12	200	180	180
13	200	180	180
14	200	144	108
15	1000	900	900
16	1000	900	900

Table A4-2 : GAP Example with Synchronous Communication (Response Time Parameters)

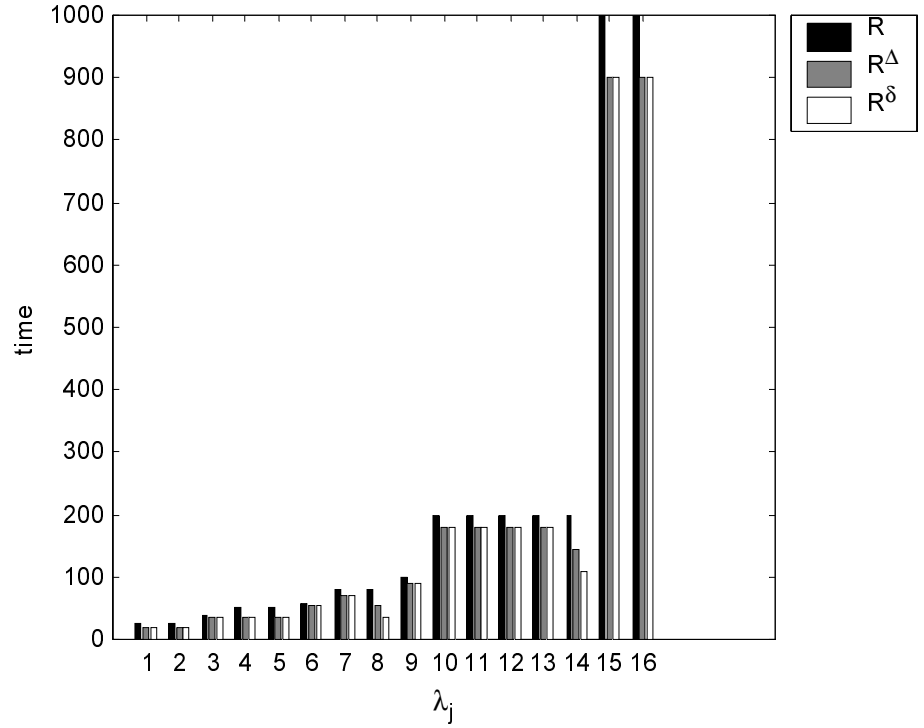


Figure A4-2 : GAP Example with Synchronous Communication (Response Time Parameters)

j	C_j	C_j^{δ}	δ_j
1	2	2	2
2	5	5	5
3	1	1	1
4	3	3	3
5	5	5	5
6	8	9	3
7	9	10	5
8	2	3	1
9	5	5	1
10	1	1	1
11	3	3	3
12	1	1	1
13	1	1	1
14	3	4	1
15	1	1	1
16	1	1	1

Table A4-3 : GAP Example with Synchronous Communication (Computation Time Parameters)

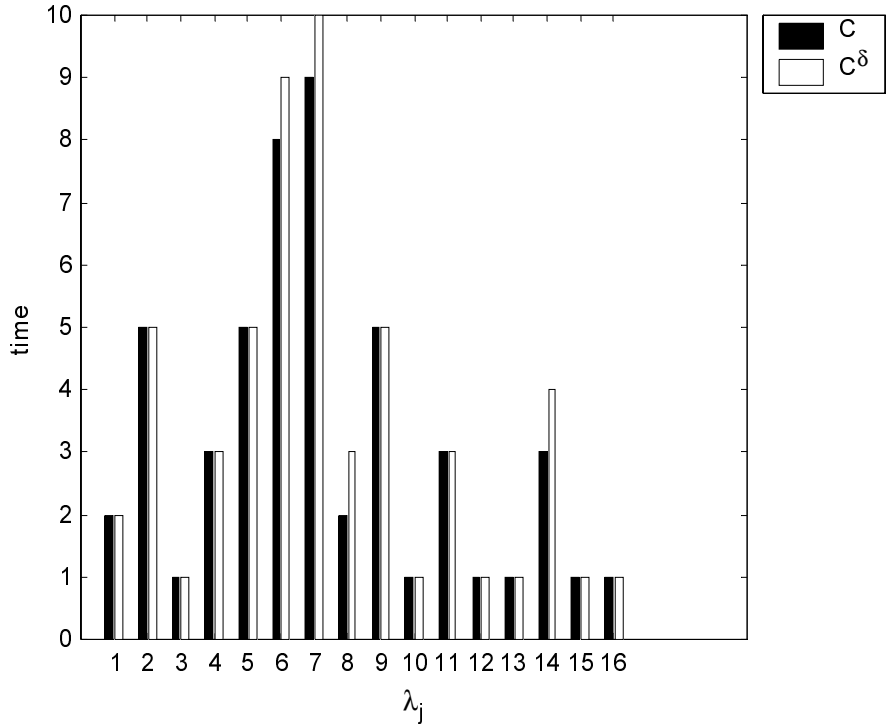


Figure A4-3 : GAP Example with Synchronous Communication (Computation Time Parameters)