DESIGN PATTERNS TO SUPPORT THE MIGRATION BETWEEN EVENT-TRIGGERED AND TIME-TRIGGERED SOFTWARE ARCHITECTURES

Thesis submitted for the degree of
Doctor of Philosophy
at the University of Leicester

by
Farha Lakhani
Embedded Systems Research Group
Department of Engineering
University of Leicester
Leicester, UK

June 2013

Farha Lakhani

Design patterns to support the migration between event-triggered and time-triggered software architectures

Abstract

There are two main architectures used to develop software for modern embedded systems: these can be labelled as “event-triggered” (ET) and “time-triggered” (TT). This thesis is concerned with the issues involved in migration between these two architectures.

Although TT architectures are widely used in safety-critical applications (for example, in aerospace and medical systems) they are less familiar to developers of mainstream embedded systems. The work in this thesis began from the premise that – for a broad class of systems that have been implemented using an ET architecture – migration to a TT architecture would improve reliability.

It may be tempting to assume that conversion between ET and TT designs will simply involve converting all event-handling software routines into periodic activities. However, the required changes to the software architecture are, in many cases rather more profound. The main contribution of the work presented in this thesis is to identify ways in which the significant effort involved in migrating between existing ET architectures and “equivalent” (and effective) TT architectures could be reduced. The research has taken an innovative step in this regard by introducing the use of ‘Design patterns’ for this purpose for the first time.

This thesis describes the development, experimental testing and preliminary assessment of a novel set of design patterns. The thesis goes on to evaluate the effectiveness of some of the key patterns in the development of some representative systems. The pattern evaluation process involved both controlled laboratory experiments on real-time applications, and comprehensive feedback from experts in industry.

The results presented in this thesis suggest that pattern-based approaches have the potential to simplify the migration process between ET and TT architectures.

The thesis concludes by presenting suggestions for future work in this important area.

Acknowledgements

This thesis is an accumulation of three years of work and this journey would not have been possible for me without the support from a number of people I met during this period.

Firstly, my research supervisor Professor Michael J. Pont deserves the highest gratitude for providing his guidance, feedback and for his encouragement throughout my research. I am highly indebted to him for providing me the opportunity to present and defend my work at various events during the research. Apart from technical guidance there was much for me to learn from his professionalism, passion and zeal in his inspirational work which I will always remember and appreciate.

Next, my heart-felt thanks to Dr. Kevin Byron for providing me his kind help and support in the early stages of the research which enabled my research flight take off at the right time and to be a pleasant experience. Also thanks are due to PG tutor Dr. Fernando Schlindwein for his help and support.

I feel myself fortunate for having a highly competent and professional team of people working around me during the course of my research. I take all these people as stars lighting up the path and making the walk easier for me. I am thankful to Dr. Anjali Das, Dr. Ayman Gendy, Azura, Dr. Devaraj Ayavoo, Huiyan Wang, Dr. Imran Sheikh, Ioannis Kyriakopoulos, Dr. Kam Chan, Dr. Keith Athaide, Mohammad Aamir, Dr. Musharraf Hanif, Peter Vidler, Dr. Susan Kurian, Syed Aley Imran Rizvi, and Dr. Zemian Hughes.

A special thanks to Georgy Holden, Hooriyah Begg, Dr. Keith Athaide, Dr. Kevin Byron and Dr. Susan Kurian for their help in proof reading my thesis and for giving valuable feedback, and to Dr. Keith Athaide and Dr. Zemian Hughes for helping in experimental work during the research.

I can never forget my lovely parents to whom I am always indebted for everything they gave me and did for me. Compliments are due to my late father for providing me support and inspiration throughout his life that motivates me to achieve the best even today. Thanks to my mum for her countless prayers for my success. I am also grateful to every member in my family.

Thanks to my enduring friend Anila Soomro for all her love and support during the last three years and for lifting up my morale from abroad.

Also, I would like to take this opportunity to thank my funding body - the Higher Education Commission of Pakistan government and Bahria University.

Dedicated to my loving mother

and

to my very dear departed father

Mushtaq Ahmed Lakhani

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF PUBLICATIONS

A number of papers were published during the course of the work described in this thesis. These are listed below in reverse chronological order.

Directly-related publications

1. Lakhani, F. and Pont, M. J. (2012) “Empirical studies for the assessment of the effectiveness of design patterns in migration between software architectures of embedded applications”. Journal of Software Engineering International Scholarly Research Network (ISRN), Volume 2012 Article ID: 259064 ISSN: 2090-7680 doi:10.5402/2012/259064

2. Lakhani, F. and Pont, M. J. (2012) “Applying design patterns to improve the reliability of embedded applications through a process of architecture migration”. Proceedings of the 9th IEEE International Conference on Embedded Systems and Software (ICESS 2012), Liverpool, UK. IEEE Computer Society: pp1563-1570

3. Lakhani, F., Wang, H. and Pont, M. J. (2011) “Supporting the migration between event-triggered and time-triggered software architectures: A small pattern collection intended for use by the developers of reliable embedded systems”. Technical report ESRG 2011-09-01.

4. Lakhani, F., Pont, M.J. and Das, A. (2010) “Improving the reliability of embedded systems as complexity increases: supporting the migration between event-triggered and time-triggered software architectures” Proceedings of the 15th Annual European Conference on Pattern Languages of Programming EuroPLoP’10, Irsee, Germany. Published by ACM New York, NY, USA. ISBN: 978-1-4503-0259-3

5. Lakhani, F. and Pont, M.J. (2010) “Using design patterns to support the migration between different system architectures” Proceedings of the 5th IEEE International Conference on Systems of Systems Engineering (SoSE), June 2010, Loughborough, UK: pp1-6.

6. Lakhani, F. and Pont, M.J. (2010) “Code balancing as a philosophy for change: Helping developers to migrate from event-triggered to time-triggered architectures” Proceedings of the 2010 UK Electronics Forum, 30 June - 1 July 2010, Newcastle, UK, Published by Newcastle University. ISBN: 978-0-7017-0232-8

7. Lakhani, F., Pont, M.J. and Das, A. (2009) “Towards a pattern language which supports the migration of systems from an event-triggered pre-emptive to a time-triggered co-operative software architecture” Proceedings of the 14th Annual European Conference on Pattern Languages of Programming EuroPLoP’09, Irsee, Germany, 8-12 July

2009. Published by CEUR, volume 566, ISSN: 1613-0073: pp. F2-2 to F2-25

8. Lakhani, F., Pont, M.J. and Das, A. (2009) “Can we support the migration from event-triggered to time-triggered architectures using design patterns?” Proceedings of the 5th UK Embedded Forum, Leicester, UK, pp. 62-67. Published by Newcastle University. ISBN: 978-0-7017-0222-9.

Associated publications

9. Das, A., Lakhani, F., Gendy, A.K and Pont, M.J (2009). “Two simple patterns to support the development of reliable real-time embedded systems” 14th European Conference on Pattern Languages of Programs, EuroPLoP’09 (Kloster Irsee, Germany, 8 – 12 July 2009)

Poster publications

10. Lakhani, F. and Pont, M.J. “Best practices for engineers to design safer systems”, poster selected for presentation at the University of Leicester, Festival of Post Graduate Research (Leicester, UK, June 2011).

11. Lakhani, F. and Pont, M.J. “’Building in’ safety in the electronic age”, poster selected for presentation at the University of Leicester, Festival of Post Graduate Research (Leicester, UK, June 2010).

12. Lakhani, F. and Pont, M.J. “Staying safe in a world full of silicon chips”, poster selected for presentation at the University of Leicester, Festival of Post graduate Research (Leicester, UK, June 2009).

LIST OF ABBREVIATIONS, SYMBOLS AND UNITS

Abbreviations

ADC	Analogue-to-Digital Converter
BCET	Best-Case Execution Time
CAGR	Compound Annual Growth Rate
CAN	Controller Area Network
CCS	Cruise-Control System
CDRH	Center for Devices and Radiological Health
CPU	Central Processing Unit
DAQ	Data Acquisition
DM	Deadline Monotonic
EDF	Earliest Deadline First
ET	Event-Triggered
FFT	Fast Fourier Transform
GCD	Greatest Common Divisor
GoF	Gang of Four
GPIO	General Purpose Input Output
LCM	Least Common Multiple
LLF	Least Laxity First
MC	Mission Computer
NOP	No Operation
NPG	Non- Pattern Group
NPP	Non-Pattern Participant
OED	Oxford English Dictionary
OFP	Operational Flight Program
PG	Pattern Group
PIE	Pattern Implementation Example
PLoP	Pattern Languages of Programming
PMES	Patterns for Migrating Embedded Systems
PP	Pattern Participant
PTTES	Patterns for Time-Triggered Embedded Systems
QWAN	Quality Without A Name
RM	Rate Monotonic
RTSC	Real-Time System Compiler
SPE	Synergistic Processor Element
TT	Time-Triggered
TTC	Time-Triggered Co-operative
TTCAN	Time-Triggered Controller Area Network
TTH	Time-Triggered Hybrid
TTP	Time-Triggered Pre-emptive
UML	Unified Modelling Language
WCET	Worst-Case Execution Time

Symbols

ARJ_i	Absolute release jitter observed for task *i*
C_i	Execution time of Task *i*
CPU_{active}	Processor busy time
CPU_{idle}	Processor idle time

D_i	Relative Deadline of task *i*
I_{CPU}	Average current consumption by the processor
$I_{DD(DCDC)active}$	Active mode DC-to-DC converter supply current
$I_{DD(DCDC)pd}$	Power-down mode DC-to-DC converter supply current
n	Number of tasks
P_{CPU}	Average power consumption by the processor
p_i	Period of task *i*
r_i	Release time of task *i*
s_{i}	Start time of task *i*
U	Processor utilisation
V_{CPU}	Average voltage across the processor

Units

ms	Milliseconds 10^{-3} seconds
µsecs	Microseconds 10^{-6} seconds

PART A: INTRODUCTION

This thesis is divided into several parts and this first part provides an introduction to the work presented as follows:

1. The overview of the embedded systems field and the description of the research problem and contributions are described in CHAPTER 1.

2. CHAPTER 2 describes the motivating examples from the literature which have provided the driving force to establish the research problem.

3. CHAPTER 3 discusses the methodology adopted for the research.

CHAPTER 1. INTRODUCTION

1.1. Introduction to the problem

This thesis is concerned with the development and evaluation of novel techniques that will help to improve the reliability of what are known as "embedded systems".

In the 1930-40s, computers were very large and expensive. Given this, it is not surprising that Thomas Watson (1943) said: *"I think there is a world market for may be five computers."* The world market has changed beyond recognition since that time. The huge computers of Watson's time have now been replaced by literally millions of much smaller "desktop", "laptop" and "handheld" computers powered by *embedded* processors.

While desktop computers perform multiple functions, an embedded system is a special-purpose computer system which is designed to perform a small number of dedicated functions for a specific application. More formally, "An embedded system is an application that contains at least one programmable computer (typically in the form of a microcontroller, a microprocessor or digital signal processor chip) and which is used by individuals who are, in the main, unaware that the system is computer based" (Pont, 2001).

Despite the ubiquitous nature of desktop and laptop computers, they represent only the tip of the iceberg when it comes to estimating computer numbers. For every processor used in a familiar desktop or laptop computer, around 100 processors are "embedded" in computer systems as widespread as aircraft, automotive vehicles, medical equipment, children's toys and DVD players (Barr, 1999; Li and Yao, 2003). They are also part of many of the electrical appliances used at home, for example, air-conditioners, irons, kettles, microwaves, refrigerators and washing machines etc. Outside homes, these processors work in automatic doors, CCTVs, escalators, intruder alarms, lifts, traffic light signals, vending machines and many other devices. In most cases, these embedded processors are a key part of the systems they inhabit, for example, around a third of the cost of developing many new cars is spent on the vehicle electronic and software systems (Bouyssounouse and Sifakis, 2005). The luxury 7-Series BMW and S-class Mercedes boast about 100 processors apiece (Turley, 2003) and these processors are in many cases serving to provide either comfort or safety, for example window and engine control and anti-braking systems in the cars.

Being in such widespread use, embedded processors have a huge international market. According to a recent study by BCC research (Business Communications Company – a publisher of technology market research reports), the global market for embedded systems is expected to increase from US$101.6 billion in 2009 to an estimated US$158.6 billion by the end of 2015, a compound annual growth rate (CAGR) of 7%. Embedded hardware was worth US$108.8 billion in 2010 and is expected to grow at a CAGR of 7% to reach US$152.4 billion in 2015. Embedded software generated US$4.2 billion in 2010 and is expected to increase to US$6.1 billion in 2015 (BCC, 2012). The growth statistics are shown in Figure 1-1.

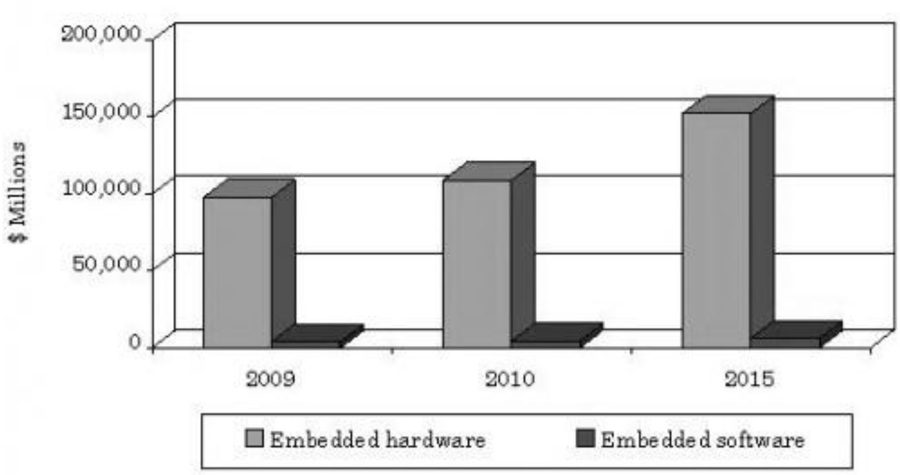


Figure 1-1 Embedded systems market 2009-2015 (Source: BCC research)

The huge growth in demand for these systems and the great increase in their complexity mean that reliability is a critical issue in the design of such systems. This is important not just to meet the needs of businesses involved in manufacturing reliable products, but also because people in developed countries are reliant on many embedded designs in systems such as aircraft, cars and medical equipment for their safety.

Although embedded applications are an essential (though mostly hidden) part of everyday life, the process of developing safe and reliable applications remains a highly challenging and complex aspect of the design and test process. Developers and designers of embedded applications face huge technical challenges such as meeting all the timing constraints, limited memory space, and restrictions on power usage. Graaf *et al* (2003) have argued that despite all the advancements such as application development tools and techniques, existing software development techniques have failed to address the challenges faced by the developers of embedded applications. This argument is supported by the fact that firms producing such devices still find it difficult to produce completely defect-free devices. For example, the Centre for Devices and Radiological Health (CDRH) reported that in 2006, 21% of all medical device recalls were for software defects (Krasner, 2010).

1.2. Description of the problem addressed in this thesis

The main focus of the research presented in this thesis revolves around two key software architectures used in developing modern embedded applications: these are termed as “event-triggered” and “time-triggered”.

For many developers, event-triggered (or “ET”) architectures are more familiar. ET designs involve creating systems which handle multiple interrupts. For example, interrupts may arise from periodic timer overflows, the arrival of messages on a serial communication bus, the pressing of a switch, the completion of an analogue-to-digital conversion (ADC) and so on. To create ET systems, the developer may write code to handle the various events either directly: this will typically involve creating an “Interrupt Service Routine” (ISR) to deal with each event. Alternatively, the event will be handled slightly less directly through use of a real-time operating system.

The alternative to an event-triggered architecture is a time-triggered (“TT”) architecture. When implementing TT systems, there is only one interrupt enabled. This single interrupt is usually linked to a timer “Tick”, which might occur (for example) every millisecond: this tick, in turn, drives all software activity in the system.

Both ET and TT architectures have their own strengths and weaknesses. The main strength of TT architecture is its ability to enable systems to be more 'predictable' (Nissanke, 1997; Pont, 2001; Kopetz and Bauer, 2002; Albert and Bosch GmbH, 2004). Since highly predictable system behaviour is an important design requirement for many embedded systems, TT software architectures have become the subject of considerable attention. It has been widely accepted that the TT architecture is a suitable candidate for many safety-critical applications since they help to improve overall safety and reliability (Allworth, 1981; Nissanke, 1997; Bate, 1998). For example, TT architectures have been accepted as a generic solution for highly dependable[1] systems such as X-by-Wire[2] systems (Ayavoo, Pont *et al.*, 2005; Ayavoo, 2006). However, even in more mundane domestic applications (e.g. an alarm clock that fails to sound on time or a video recorder that operates intermittently) where failure is relatively inconsequential, poor reliability can have other impacts such as reduced sales etc. Besides being predictable in nature, TT systems are also easy to validate, test and certify (Liu, 2000).

The published literature provides many examples of embedded applications such as those discussed in (Shepard and Gagne, 1990; Kubinger and Humenberger, 2004; Turley, 2009) for which the ET architecture is initially preferred because of the perceived benefits they offer such as flexibility in design and their ability to respond more quickly upon arrival of any internal or

[1] Dependability is a measure of whether a system can be relied upon to perform the desired action. Kopetz (1997) and Hanmer (2007) define the different attributes of dependability such as reliability, safety, availability and security.

[2] X-by-Wire systems are electronic systems without mechanical backup where the 'X' stands for the safety related applications such as steering and braking. Some keywords in automotive systems are "accelerate-by-wire", "steer-by-wire" and "brake-by-wire".

external events. Later in their life cycle these applications exhibit problems related to reliability and therefore migrated to TT architecture.

It is therefore realised that to accomplish reliable software systems especially where safety is a critical issue migration between software architectures becomes crucial to achieve systems that are more predictable and safer. However, it is also accepted by the expert's community that migrating software architectures in embedded applications offers various challenges. It may be tempting to assume that conversion between ET and TT designs will simply involve converting all event-handling software routines into periodic activities. However, the required software changes are – in many cases – much more profound. For example changes to one part of the software may affect other inter-linked parts and so a thorough analysis of the migrating system is necessary before making any substantial changes. The overall goal of the work presented in this thesis is to identify ways in which the work involved in migrating between existing ET architectures and "equivalent" and effective TT architectures could be reduced.

As the research is motivated by providing support during the process of architecture migration, for this purpose research has explored the concept of 'Design Patterns'. Design patterns emerged from the field of architecture and gained popularity in diverse disciplines. Design patterns are well-documented, time-tested solutions to classic design problems and capture significant domain knowledge. In the field of embedded systems most of the previous research work (Adams, Coplien *et al.*, 1996; Bottomley, 1999; Pont, 2001;

Herzner, Kubinger *et al.*, 2005; Cloutier and Verma, 2007; Eloranta, Koski *et al.*, 2009) focused on documenting patterns for system construction. However, design patterns which could assist embedded system practitioners in the process of migration of architecture has been somewhat neglected. The research presented in this thesis aims to explore how this gap might be bridged.

1.2.1. Research premise and goal

The work presented in this thesis began with the premise that – for many embedded systems which are implemented using an ET architecture – migration to a TT architecture would improve reliability.

From this starting point, it was accepted that altering the system architecture would not be a trivial process and the core research goal was to explore whether an appropriate set of 'design patterns' could be used to facilitate this transition.

1.3. Research contributions

The research described in this thesis makes the following contributions:

- It has explored the need for migration from existing event-triggered architectures to time-triggered architectures in order to improve system reliability.
- It has explored the challenges involved in the migration process from event-triggered to time-triggered architectures.

- This research has explored – for the first time – ways in which design patterns can be used to support the migration between event-triggered and time-triggered software architectures and resulted in the development of a pattern language to support the migration process. The pattern language is introduced by identifying links between previously proposed patterns by peers and the new patterns proposed during the course of this research.
- The research has also performed the rigorous assessment of the pattern language as follows:
    - By demonstrating the applicability of the proposed patterns on real applications through laboratory experiments.
    - By conducting controlled experiments with a target audience of users.
    - Through an industrial survey to obtain feedback from the practitioners on how the proposed patterns could be useful in the industrial context.

1.4. Outline of the thesis

This thesis is structured as follows.

- **Part A – Introduction:**
    - CHAPTER 1 provides the relevant background information to the research problem undertaken and describes the contributions made by this research.

  - CHAPTER 2 explains the motivating example systems from the literature which helped in the formulation of the research problem.
  - CHAPTER 3 describes the research methodology adopted in detail.

- **Part B – Literature Review:**
  - CHAPTER 4 provides a discussion on the two main software architectures (event-triggered and time-triggered) which is the focus of this research. Related to this the chapter also explains different scheduling schemes which constitute the software architectures. The chapter also provides a comparison of the two architectures as discussed in the published literature.
  - CHAPTER 5 describes the problem of migration that embedded applications face during their life cycle. This chapter presents details on reasons for migration in embedded systems and dependencies between components and the related work carried out by previous researchers in the field.
  - CHAPTER 6 introduces the concept of design patterns and their historical background. It briefly discusses how this concept was acknowledged by experts in diverse fields and their acceptance and implementation. The chapter also discusses the process of applying design patterns in building embedded applications, and how and why they can assist the developers of embedded applications.

- **Part C – Development and evaluation of patterns for migration:**

  - CHAPTER 7 describes the derivation process for the patterns.
  - CHAPTER 8 introduces a new collection of design patterns proposed with the aim of helping developers in migrating from event-triggered architectures to time-triggered architectures. It also discusses new patterns that are introduced by this research.
  - CHAPTER 9 demonstrates the use of the proposed pattern collection with case studies and examples of how various patterns can be applied during the migration process from event-triggered design to a time-triggered design.
  - CHAPTER 10 presents the evaluation process of the newly proposed patterns and the empirical studies used in the process.
  - CHAPTER 11 presents the evaluation of the patterns in the industrial context. This is achieved using the web-based survey with employees in the software industry and a website was also developed for this purpose.
  - CHAPTER 12 is dedicated to conclusions and the future expansion of this research that could further enhance the initial steps taken.

- **Part D – Appendices:** The appendices include supplementary materials related to this thesis.
  - Appendix A describes different pattern forms found in the literature.
  - Appendix B contains full specifications of the patterns that are derived and documented during the research.

- Appendix C contains the exercises designed for use during the empirical evaluation of the research.
- Appendix D contains the documents related to industrial evaluation of the research.
- Appendix E provides the bibliographic references to all the citations used.

1.5. Conclusions

This introductory chapter has provided a summary of the overall theme of the work described in this thesis. The discussion regarding software architectures indicated that several embedded applications may demand change in their existing architecture during the course of their life cycle due to issues with reliability and safety. This change in architecture offers various challenges to the designers/developers of embedded applications which have often been overlooked by previous research work in the field resulting in a lack of support for the designers to tackle the situation. In order to solve this and overcome some of the difficulties involved in the migration process this research presents the novel idea of using design patterns. Based on these discussions the main goal and key contributions made by this research were stated and the layout of the thesis provided.

The remainder of this thesis will describe the work undertaken by this research.

CHAPTER 2. MOTIVATING EXAMPLES

2.1. Introduction

As outlined in CHAPTER 1, the main objective of the work presented in this thesis was to develop and assess techniques that will improve the reliability of existing software applications through a process of architecture migration.

In order to explain the motivation for this work, this chapter summarises some representative examples that have been found in the literature (Section 2.2 to Section 2.4). Section 2.5 provides a discussion based on the examples described and the relevant concerns of the research and conclusions are given in Section 2.6.

2.2. The F-18 Mission Computer

The case study related to a F/A-18 aircraft has provided the motivation to explore the need for migration between different software architectures for complex embedded applications. The F/A-18 aircraft is also referred to as CF-188 by the Canadian Forces and was manufactured by McDonnell Douglas Corporation. The case study is presented in detail in (Shepard and Gagne, 1990).

Maintenance problems were reported for F/A-18 Mission Computer (MC) Operational Flight Program (OFP) when some modifications were required in the system.

A block diagram representing the runtime scheduler for this system is shown in Figure 2-1.

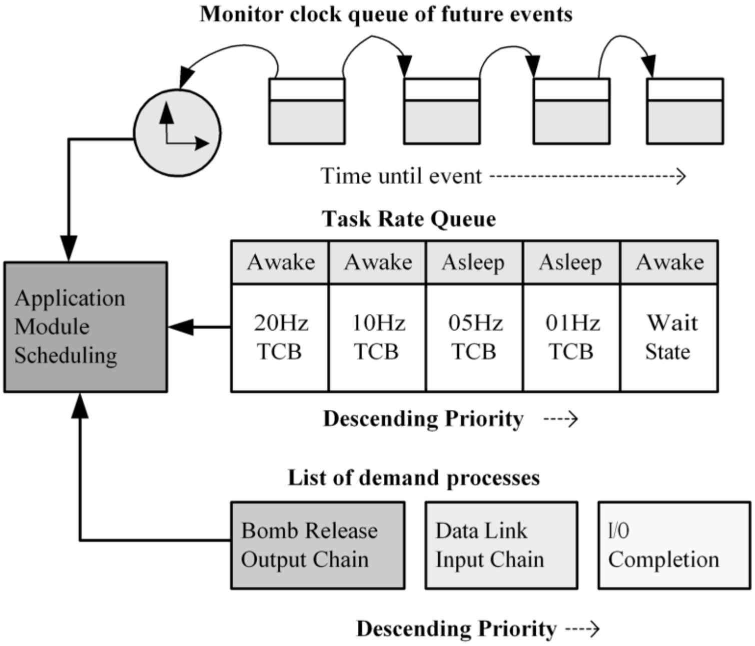


Figure 2-1 Model of F-18 MC [adapted from (Shepard and Gagne, 1990)]

The system model was based on a pre-emptive scheduling process (such a process is described in more detail in Section 4.2.2: the details are not relevant here). The model was based on a design that was required to run periodic processes (grouped into four separate task rates of 20, 10, 5 and 1 Hz) as well as interrupt service routines (termed as demand routines by authors) for bomb release, data link input, I/O completion, and I/O fault recovery.

The relative priorities of the task rates is reflected in the structure of the 'Task Rate Queue', implemented as a linked list. This queue contains one Task Control Block (TCB) for each task rate. The TCB indicates the status of the task rate (awake or asleep) and provides storage for control information related to that rate. The occurrence of interrupts pre-empts the currently running tasks. Once the interrupt processing is finished, the pre-empted task processing is resumed.

2.2.1. Problems with the existing model of the F-18 MC

The major problem reported with the system was failure to observe all the timing constraints when new software components were added. With the existing model it was also difficult to guarantee that any alterations in the system did not alter the logical correctness of the system. Further, the testing methodology was not comprehensive enough to satisfy all constraints. All these factors highlighted the need for a scheduling methodology which would enable verification of the timing constraints of the system before its operation and easy testing of the system. In other words, predictable behaviour was the main requirement of the modified system.

2.2.2. Migration to pre-runtime scheduling

Because of the problems with the existing model, it was decided to shift the entire system to pre-runtime scheduling to make it possible to verify all the timing constraints in advance. However, it was a big challenge to convert such a complex system to pre-runtime scheduling. This involved an extensive review of the complete OFP source code, a study of 2500 pages of technical documentation and a review of 2400 software routines. Flow charts were generated to understand the scheduling requirements of the OFP and all the interrupt service routines were converted to periodic processes (this is described as a ‘polling solution’). This resulted in a 95% reduction of the interrupts being generated by the system. The schedules were generated to run all the tasks in the system along with their release times, execution times and deadlines.

2.2.3. Performance improvement after migration

Shepard and Gagne (1990) have reported the following improvements in the transformed pre-runtime model of the F-18 MC:

- As the major problem with the system is testing and verification after adding new modules, migration to pre-runtime scheduling has made the testing process far easier and less labour intensive.
- To guarantee reliable system behaviour in the case of complex embedded applications it is important to predict maximum CPU load and I/O processing capacities during the design stage. Transformation of the system from pre-emptive scheduling to pre-runtime scheduling helped in achieving this.
- As pre-runtime scheduling enable offline schedules to be made this helped in facilitating the timing requirements for all the system processes in advance.
- Pre-runtime scheduling also significantly reduced the number of context switches in the system. With the previous model based on pre-emptive scheduling approximately 1000 context switches were estimated per second incurring significant overhead. Pre-runtime scheduling was helpful in countering this problem.

2.3. Migration towards time-triggered image acquisition

The second example found in the literature that justified this research is a case study based on replacing a time-triggered model for the existing event-triggered model for an image processing application (Kubinger and Humenberger, 2004). For this application there was a preference for TT architecture as the existing design based on ET architecture was observed to have variations in the sampling time which increased the probability of data (images) loss. Also the TT architecture can have fixed latency, low jitter values and predictable behaviour. The researchers aimed to investigate the interfacing issues between cameras and TT architectures and to achieve a jitter-free and synchronized sequence of images. The objective was to achieve a guaranteed behaviour of the camera such that the system must finish processing of image ‘n’ when image ‘n+1’ arrives. Their proposed TT model (distributed in nature) makes use of a TT bus (for example TTP/C, FlexRay or TTCAN). The bus used the TDMA (Time Division Multiple Access) principle and each message is assigned a periodic time slot which made it possible to precisely control the triggering of the camera and acquisition and transfer of the images from the camera to the node. The layout of the concept is shown in Figure 2-2.

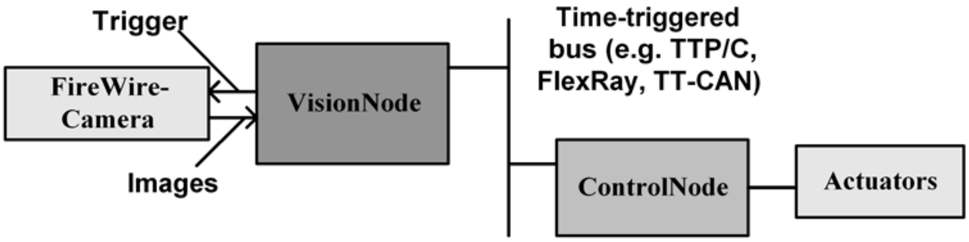


Figure 2-2 Layout of the vision-based distributed embedded control application [Adapted from (Kubinger and Humenberger, 2004)]

They have demonstrated the application of their proposed approach using a SONY FireWire-camera DFW-VL500 on a 400 Mbps FireWire bus. They also presented the comparative results of ET and TT models. The results showed less latency values (highly varying) for the ET model and jitter[3] values of 33.33ms. On the other hand, the TT model showed high but fixed values of latency and jitter values in the microseconds range only.

2.4. Sony® cell processor

This example is quoted by Jim Turley in (Turley, 2009). He reported an interesting case observed by the programmers of Sony Corporation published in ‘Embedded use of the Cell processor’ (Kawamura, Yamazaki *et al.*, 2008). Half of the cell chip is dedicated to eight identical Synergistic Processor Elements (SPEs) having their own instruction set unique to the cell. Each SPE is a 128-bit single-instruction, multiple-data (SIMD) vector processing machine and has its own 256K block of private RAM for executing code or storing local data. The Linux distribution for the cell treats the SPEs as virtualized resources; this resulted in the creation of more SPE threads than there are SPEs. The overhead of this feature is that one SPE might be swapped out while it is running. To avoid this, the Sony team “pinned” the network stack to one SPE, effectively prohibiting Linux from swapping it out and dedicating that SPE exclusively to network processing.

The problem observed with the system can best be illustrated by citing a passage from (Turley, 2009). *“Given the high packet rates Sony was hoping for, frequent interrupts turned from being a necessity to being a problem. In their experience, most network stacks are interrupt-driven, especially from the*

[3] Jitter refers to the deviation from the ideal timing of an event and can have a serious impact on system reliability. More details are explained in Chapter 6

hardware interface when it needs servicing. As data rates climb, these interrupts (and their attendant context switching) become so frequent that the overhead overwhelms the actual task. The faster it works, the slower it goes." The solution reported in the article was based on the removal of interrupts from the system. *"To fix this, the team decided to switch from an interrupt-driven to software polled design. They kept the same hardware components but just tweaked the network driver to poll the chip at regular timer tick-intervals. The resulting efficiency was dramatic."*

This example shows that an interrupt-driven architecture initially chosen to make the system more responsive can ultimately jeopardise the whole system in situations of high load. Finally a software based polling mechanism was adopted to overcome the problem and was found to be very effective.

2.5. Discussion

In this chapter three examples from the literature are described that highlight the need for migration of existing event-triggered and/or interrupt-driven software architectures in embedded applications to time-triggered architectures in order to improve system reliability.

The F-18 aircraft example discussed in Section 2.2 is an example of high reliability high integrity embedded application for which all the necessary issues were considered at the design time and so a priority based pre-emptive task model was chosen. The selected architecture did not proved a wise choice in the later stages when the application required new software components to be added. The testing of the application was another issue that arose when alterations were made to the existing model. The aircraft was then migrated to pre-runtime scheduling model which facilitated the testing and verification process and showed improvement in system reliability along with the reduced system overheads such as CPU and memory utilization. The transformation process itself however was labour intensive involving heavy technical documentation reviews in order to make necessary changes in the code.

The second example discussed in Section 2.3 for the image processing application also showed considerable improvement in jitter values (from the millisecond range to the microsecond range) when transformed to time-triggered architecture from its existing event-triggered design. The rationale for the transformation was described as possible loss of data with the existing model. The transformation process involved the use of a time-based communication protocol and periodic scheduling of task at the design time which itself involved a number of design decisions for optimised system performance which are not considered in the study described by the authors.

The third example is a commercial product developed by the Sony Corporation in which the migration from interrupt-driven architecture to polled-input design helped to overcome the problems when the application was subjected to peak load situations. However it is assumed that for commercially confidential reasons the example does not discuss how the challenge was faced and what the major concerns were during the transformation process.

The examples discussed have provided the motivation to:

- Further explore the need of migration from event-triggered architectures to time-triggered architectures in order to improve system reliability.
- Explore the possibilities of some kind of support which could assist the developers in making better choices among a variety of possible time-triggered architectures. This would enable developers to take better informed decisions during the migration process from event-triggered design to time-triggered designs.

2.6. Conclusion

This chapter has presented three examples from the literature to highlight the requirements for changes in software architecture of an application later in its life cycle. All of the three applications were designed with an architecture where multiple interrupts are made active to handle any input data. The examples have demonstrated that in such architectures the need to change may arise for problems due to system performance and reliability in peak load situations, when new components are added and where further testing or verification processes are needed. In all the three examples the architecture was transformed to remove multiple interrupts and so a time-based architecture was chosen in which offline schedule plans can be enabled to predict system behaviour in advance. These examples helped in building the overall argument for the research problem that proposed that migrating applications to an architecture where single interrupt can handle all system activities may help in improving system reliability.

The next chapter explains the overall research process and the choice of methodology adopted to meet the challenges of this research.

CHAPTER 3. METHODOLOGY

3.1. Introduction

A methodology in research refers to the theoretical argument and investigative framework that researchers use in order to justify their research methods and the design of their experimental investigations (Case and Light, 2011). This chapter describes the methodology adopted to conduct the research presented in this thesis.

This chapter organized as follows. Section 3.2 presents a general overview of the methodology adopted by the research, Section 3.3 discusses the rationale behind the formulation of the research problem, and Section 3.4 describes the steps involved in building the hypothesis and setting goals for the research in order to test the hypothesis. Following this, Section 3.5 explains the research work that involves derivation of design patterns and experimental work to test and implement the patterns. Section 3.6 explains the methodology adopted to evaluate the research and the chapter conclusions are presented in Section 3.7.

3.2. General approach to the investigation

In common with most research projects, the work presented in this thesis has evolved in stages and each stage has been subject to several iterations. In a wider sense the process of identifying a valid research problem, identifying a proposed solution to a problem, developing the solution, and then testing and verifying the solution, is identical to the process generally described as 'The

Scientific Method' first formulated over 400 years ago (Gower, 1997). The Scientific Method of course also tacitly underpinned and guided the technical work described in the thesis. The definition of the modern scientific method as described in (Bock, 2001) is: *"The Scientific method comprises four sequential phases – Analysis, Hypothesis, Synthesis and Validation – which are applied to a task iteratively and recursively to achieve the objective of the task."* A research process is in fact similar to undertaking a journey in which a person must decide where he/she wants to go and which route to take. The problem is compounded in deciding which one is the best to follow if there is more than one possibility to get to the destination. Similarly a research process is all about deciding what to do, planning how to do it and then doing what is considered as appropriate in the existing scenario.

The next few sections will describe how this research underwent several stages from the initial point of formulation of the research problem, to setting goals, choosing the appropriate methodology at each step during the research, and evaluating the final outcome.

The process is described as phases of analysis, hypothesis, synthesis and validation and each phase was broken down into further operational steps as depicted in Figure 3-1.

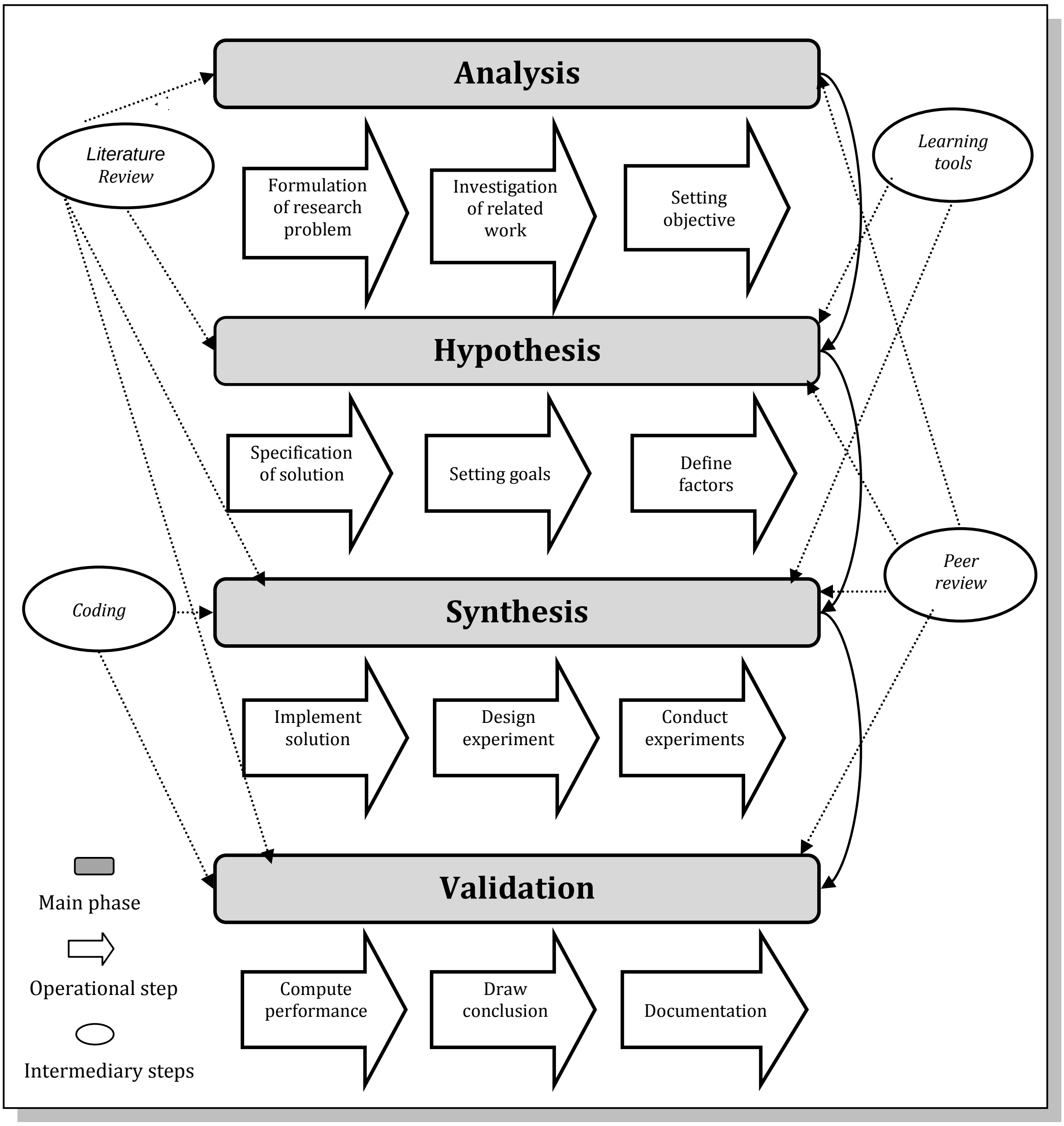


Figure 3-1 Illustrating the different steps taken during the main phases of the research process

3.3. Formulation of the research problem

The research was initiated with the literature review in order to gain a thorough understanding of the domain of embedded systems design. In this initial process both the wide ranging work undertaken by the previous research work in the larger specialized community and in the local research group (ESRG at

University of Leicester) were reviewed. The underlying aim of the process is to formulate a specific and valid research problem which can add new knowledge to the existing body of knowledge in the domain and can be acknowledged as a novel contribution. The research was focused more towards embedded software development rather than hardware because this played into the prior experience and interest of the author in desktop software development. Following the literature review, group discussions in the research group and personal interest it was agreed that the main direction of the research should be aimed at bringing improvements into the reliability of embedded software architectures. Whilst conducting investigations on this topic at this early stage it was realised that many existing software applications need to be modified and improved during their life cycle in order to improve their reliability. Examples of such applications have already been discussed in CHAPTER 2. For example the F-18 aircraft system discussed in Section 2.2 went through an entire change in the scheduling policy from pre-emptive to pre-runtime in order to improve reliability. The process to make these changes in the architecture however was extremely tedious for those who were involved in the process and alleviating this problem was seen as a worthy research challenge that would contribute to the existing knowledge in, and progress of, this field of interest. As part of this overall aim of finding ways to make software architectural migration easier, other benefits of improving the system reliability are realized, and another part of the research was involved in evaluating these benefits. The research question then can be articulated as: “Identifying the challenges involved during the migration between different software architectures of complex embedded applications, in order to improve reliability and to provide an appropriate solution to these challenges".

The research question was further condensed to two primary software architectures as event-triggered (ET) and time-triggered (TT) as various other architectures could be described as subset of these two architectures. Also in the motivating examples described in CHAPTER 2 the existing design of the applications was primarily ET which was transformed to TT in order to improve system reliability and performance.

Having established 'What to do' with the formulation of the research question, there was the next challenge of 'How to do it' in terms of a methodology.

3.4. Development of hypothesis and choosing the appropriate methodology

Improving the reliability of embedded systems was central to the research so it was important to investigate what factors contribute to the reliability of an embedded system? In general software organisations focus on the following factors in order to achieve reliable system design:

- Clear functional requirements: At the system application level the functional requirements (i.e. logic) can be expressed by using semi-formal methods such as control flow diagrams or logic/function block diagrams (Liu, 2000).

- Clear temporal requirements: This includes correct specifications related to task timings for example release time, start time and period etc (Buttazzo, 1997).
- Choice of a programming language: The programming language should be capable of being fully and unambiguously defined. The language should be used with a specific coding standard and a restricted sub-set, to minimize unsafe/unstructured use of the language (Barr, 1999).
- Choice of hardware platform: The hardware must be chosen wisely considering cost performance ratio.
- Coding guidelines: When faced with the widespread (and increasing) use of various programming languages for the production of embedded code with safety-related constraints; it was felt that there was a need to produce a set of standards to assist software developers. For example the Motor Industry Software Reliability Association (MISRA) defined a subset of the C language that can be used in critical systems (IEC, 2012).
- Code design/structure guidelines: To make embedded software code portable, re-usable and maintainable the code must follow structure guidelines so to imply clear portioning of functions and a visible hierarchy of modules and interconnections (IEC, 2012).
- Choice of software architecture: For a reliable system design this is the key component as it defines the major elements and subsystem of the software how they are interconnected and how the required attributes particularly safety integrity will be achieved (Bouyssounouse and Sifakis, 2005).

- Team expertise/experience/training: Industrial projects are usually lead by professionals with a high level of skills in the related area. However the whole team might not be at the same level of expertise and so the training requirements for less experienced individuals in the organisation need to be considered.

The points mentioned above are usually adhered to in organisations during the development phase of a product, and a great deal of time and effort is invested in documenting the design and software engineering processes with the aim to re-use the same processes in future projects. In general these documents are text-based and capture information that can be helpful for re-use of the design ideas in the future. These documents are regarded as ‘Intellectual Property’ (IP) in the organisation and often subjected to copyright/patent or privacy rules and therefore are not freely accessible to other people outside the organisation.

It can take many years for a novice software developer to acquire the tacit knowledge and skills of an expert, and this is gained through working with a variety of problems and applying best practice to solve them. By its very nature tacit knowledge is not easy to record or articulate. It is in a sense in the mind of the developer and in the complex system source code they have developed, neither of which is readily accessible by others. Consequently when skilled software developers leave an organisation they take this knowledge with them and this places a heavy burden on the organisation in terms of investment in training. If however there was a mechanism that could help the organisation to capture this domain-specific design resources and express them in a more explicit form accessible by all. This would greatly assist an organisation to efficiently sustain the capabilities of the work-force and reduce the costs of specialized training. It was with this challenge on how to develop domain-specific design resources that the idea of 'Design Patterns' emerged to as explained in the next section.

3.4.1. The 'Design Pattern' methodology

The concept of design patterns originated in the field of conventional (building) architecture and provided the means to explicitly highlight the hidden key design strategies and tactics to help new participants in the field. The patterns also enabled knowledge to be shared among experts more effectively. Though coming from ill-defined problems in the architecture of buildings, patterns originally gained acceptance for well-defined problems in software design such as patterns for object-oriented design methods (Gamma, Helm *et al.*, 1995), patterns for fault-tolerant software (Hanmer, 2007), and design patterns for high availability systems (Kalinsky, 2002) . In this context the principal contribution of design patterns is that they explicitly capture expert knowledge and design trade-offs and thus support the sharing of architectural knowledge among software developers. This means that a pattern language (a collection of related patterns) may provide support to software developers by addressing all the issues mentioned in Section 3.4 that contribute to the design of embedded systems with improved reliability. This helped in articulating the research goal as: "The use of design patterns to support the migration between ET and TT software architectures for embedded systems". An early draft of this research objective was published in (Lakhani, Das *et al.*, 2009b) with the accompanying research question "Can we support the migration from event-triggered to time-triggered architectures using design patterns?". Through the review process the peer community appreciated the idea on the use of design patterns which helped in building the confidence that there was novelty in the research problem.

The initial goal was set to derive patterns for migration from event-triggered design to time-triggered co-operative (TTC) design, and identify problems that are involved in this translation so that solutions could be explored and tested for documenting the related patterns. Future goals were also set to find ways in which pattern users could be offered more options for TT designs and their related problems. It was also realised during the research that along with the description of patterns for changing the architecture, it would be essential to describe ways to attain optimised TT designs after migration. In conclusion the goal was set to form a pattern language which can support practitioners in the migration process from ET to TT designs and to attain optimised TT design after migration.

3.4.2. Alternative approaches

One other possible way to tailor the research problem described in Section 3.3 could be the design of an automated tool-set such as an expert system which can help in the migration process. An account of why this approach was not adopted is presented below.

An expert system is a computer system that attempts to mimic human expertise by applying inference methods to a specific body of knowledge called a 'domain' (Darlington, 2000). Development of an expert system involves the use of special expert system languages such as CLIPS (C Language Integrated Production System) and special hardware designed to facilitate the system. These systems have a competitive edge in the sense that computer expertise (in the form of an expert system) is relatively easier to transfer as compared to human expertise. This is because an expert system could be copied to another PC on a different site or even downloaded to a network of PCs, whereas human expertise is not reproducible in this way. Human expertise is perishable in that people may switch job, retire etc., whereas expert systems are a way of retaining knowledge and can be consistently available.

A repository of design patterns in an organisation can play an important role to replace a loss of expertise or an expert system as patterns can help an organisation to “back up” key skills from a team of expert designers and provides a cheaper availability of the solution. Patterns provide solutions documented by a domain expert and could be followed by an unlimited number of experts working in the same domain. In this sense, patterns help to promote creativity and enable different experts to obtain solutions with varying dimensions. To quote from Vlissides (1997) “patterns are primarily food for the brain”. Whilst expert system programs have the capabilities for learning that transcend those available in a conventional program, these capabilities are still very primitive compared with human learning. Humans are flexible, and can easily adapt or integrate their expertise with the use of patterns in ways that exceed the capabilities of computers.

On the other side, expert systems have the competitive edge on training individuals to use patterns as the explanation capabilities of experts systems are such that users can see a chain of reasoning underlying their decisions and hence the users can gain a better understanding of the domain.

A summary of the comparison of the two approaches is presented in Table 3-1.

Table 3-1 Summary of comparison between a design patterns repository and an expert system

Design patterns repository	Expert system
Helps to promotes creativity.	Do not have the capability to stimulate make creative responses as human experts would in unusual circumstances.
Solutions provided can be adaptable	Not adaptable.
Documented/followed by human experts who can uses senses as sensors.	Based on technical knowledge and generally uses symbols as input.
Provides best practice solutions which experts have gained through experience.	Do not tend to learn from mistakes unless user feedback and human maintenance is part of it's on- going development.
Provides a relatively low cost solution to promote expertise within the organisation.	Developing an expert system involves a considerable amount of cost in the purchase of relevant hardware and software.

Based on the discussion above both design patterns and expert systems have their own strengths and limitations however in the context of the research problem described in Section 1.2 the design pattern methodology appeared as more appropriate. This is because migration of the software architecture is not limited to a certain set of rules, but actually depends on a number of circumstances that are different for each application as discussed in the different examples in CHAPTER 2. Therefore a designer might need to introduce customisation in the given solution as per application requirement, and a design patterns repository provides this flexibility.

3.5. Implementation of the chosen methodology

After the research problem was developed and the methodology was chosen the research then focused on the implementation of solutions. In this phase the patterns were derived, documented and implemented, and experiments were designed and conducted to test and validate the patterns.

The work on the derivation of patterns incorporates literature survey, discussions, code analysis and experimental work. This helped in the process of identifying the problems involved in the migration process, seeking the appropriate solutions, and testing the solutions through experiments. More details about the derivation of each individual pattern are described in CHAPTER 7. The newly documented patterns underwent the preliminary formal evaluation process during this phase as part of their publication at various PLoP (Pattern Languages of Programming) conferences. A detailed evaluation of the research was conducted in the following phase and is described in the next Section.

3.6. Evaluation

The evaluation phase is concerned with inspecting the validity of the research. A detailed evaluation of the research was conducted in three different phases and has applied both quantitative and qualitative approaches. As the patterns proposed during the research are intended to support developers and designers of embedded applications the ideal way to evaluate them is to test their application in a real ‘live’ project which is undergoing the migration process. Such experimental studies are categorised as ‘conducted in the natural environment’ (Kumar, 2005). With this approach the study population is exposed to an intervention in its own working environment of software development. However within the remit of this research such an ‘on-site’ experimental study was not practical with commercial organisations due to the long lead time in the development of new software, and confidentiality concerns that could impact on the organisations’ Intellectual Property (IP) rights. The research was also at an early stage, and this would be regarded as too risky by organisations to invest time and effort in a process that had not already demonstrated strategic potential for them. A better alternative for the experimental design at this stage is a ‘controlled’ experiment. Here the researcher (or someone else) introduces the training or educational intervention or stimulus to study its effects in a controlled environment such as a laboratory, a computer suite or training room. This approach is more suited to academic research at an early stage where the potential benefits are still unclear, and was chosen for the first phase of the validation.

Two phases of controlled experiments were conducted described separately below.

In the first phase of evaluation experiments were conducted to demonstrate the applicability of the patterns on real applications. Two commonly used but non-trivial applications were chosen and the patterns were applied to enable migration from their existing ET designs to TT designs. This evaluation phase generated mainly the quantitative data to compare the performance of the two designs. More details about these experiments are described in CHAPTER 9. The second phase of evaluation involved user trials. During this phase empirical studies with two different hypotheses were designed and conducted involving MSc. students as subjects. The output of this evaluation phase was quantitative and the details about these studies are described in CHAPTER 10.

In the final phase the research was evaluated in the industrial context. Here a web-survey was conducted which offered the benefits of wide geographical coverage of the target audience, a rapid response and faster and cheaper data analysis. This phase has provided both quantitative and qualitative data and further details about this evaluation process are explained in CHAPTER 11.

The controlled experiments conducted in the first two phases of the evaluation have already passed through rigorous peer review and are published in (Lakhani and Pont, 2012a) and (Lakhani and Pont, 2012b).

3.7. Conclusion

This chapter has described the methodologies adopted to conduct the research described in this thesis. The various steps in each of the research phases have been described and arguments provided for the chosen approaches compared with alternatives. The following sections in the thesis will describe the overall research process in detail.

PART B: LITERATURE REVIEW

This is a review of the background material related to the research presented in this thesis. Three areas are of particular interest:

1. Software architectures for embedded systems and their comparisons explained in CHAPTER 4.

2. Issues related to migration between different software architectures for embedded systems explained in CHAPTER 5.

3. Design patterns and their journey from architecture to embedded systems explained in CHAPTER 6.

CHAPTER 4. EVENT-TRIGGERED AND TIME-TRIGGERED ARCHITECTURES

4.1. Introduction

The nature of desktop computers is general-purpose i.e. they are used to design a variety of applications from browsers and word processors to games and inventory control applications involving huge databases. Embedded system applications on the other hand are designed for some special purpose to fulfill specific computing needs for that system. For example, a cruise control system (CCS) designed for a car has completely different requirements than an application designed to run a washing machine. Embedded applications are further characterised by custom user interfaces built according to the needs of each specific application. Because of the uniqueness of each system, designing and developing embedded applications is far more challenging than developing desktop applications. For instance some high reliability systems such as aircrafts and automotive vehicles have stringent real-time requirements (details about real-time are discussed in the next section). Also, unlike most desktop applications, embedded applications have resource constraints i.e. restrictions on use of memory space and power consumption. Beside these limitations on the use of memory and power, developers of embedded applications are required to incorporate timing requirements along with the functional requirements of the applications (Graaf, Lormans *et al.*, 2003).

All embedded applications are designed and implemented as a collection of tasks. This task set provides a suitable software abstraction of the embedded application being designed (Barr, 1999). To accomplish desirable system behaviour, the triggering mechanism to activate these tasks (which determines the underlying software architecture) and the scheduling order (to execute the tasks in a certain order) are key design considerations. The appropriate choice of these elements will have an effect on the overall system reliability.

Following this brief introduction, Section 4.2 is dedicated to providing details about various scheduling schemes while designing real-time applications. Section 4.3 presents details about the two primary software architectures used when designing embedded applications – the time-triggered approach and the event-triggered approach. Section 4.4 presents a comparison of the two architectures. Finally a detailed discussion of the choice of an appropriate scheduling scheme and software architecture is presented in Section 4.5 with chapter conclusions in Section 4.6.

4.2. Taxonomy of scheduling techniques

Scheduling refers to decisions on the order and/or the execution time of a set of tasks with certain known characteristics (Balarin, Lavagno *et al.*, 1998). A scheduler is a discrete component of a real-time embedded system that makes (run-time) decisions about which of the available tasks (if any) should be executed by the processor. A scheduler consists of hardware (e.g. a timer and/or interrupt controller) and a small amount of software code. The scheduler has two main roles (see Figure 4-1):

1. Release control, i.e. deciding when tasks should be released (become active).
2. Task dispatching, i.e. when multiple tasks are active the scheduler needs to decide which tasks to execute at a given point in time.

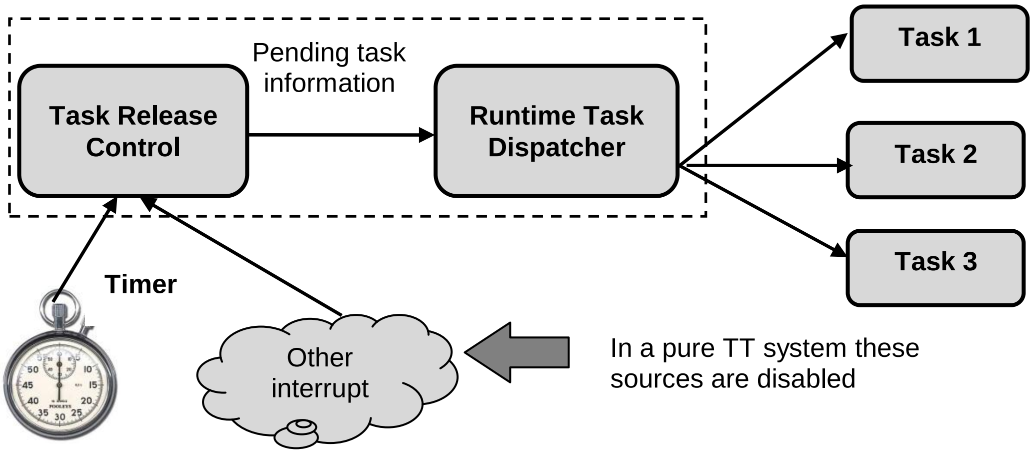


Figure 4-1 An illustration of scheduler functions

A scheduler makes use of some form of algorithm (scheduling technique) to solve these problems.

Scheduling of real-time tasks is an extensive topic and is widely discussed in the literature. This section will discuss some of the identified scheduling techniques which are related to this research.

4.2.1. Non pre-emptive or co-operative scheduling

As the name implies, this type of scheduling does not allow a currently running task to be pre-empted by another task. This is why it is also termed as co-operative, as the tasks are more ‘friendly’ and co-operative with each other. In this type of scheduling, tasks once started are executed to completion without being pre-empted (Pont, 2001). The shortcoming of co-operative scheduling is its latency in response to important events: a higher priority task will have to wait until the currently running task finishes execution (Labrosse, 2000). This is because, the task which is currently using the processor is implicitly assigned with the highest priority. Any other task must therefore wait until the currently running task finishes execution and hand over the processor control to the scheduler. The scheduler will hands over the control to next ready-to-run task in the system. Figure 4-2 shows two tasks A and B scheduled to run in a co-operative environment.

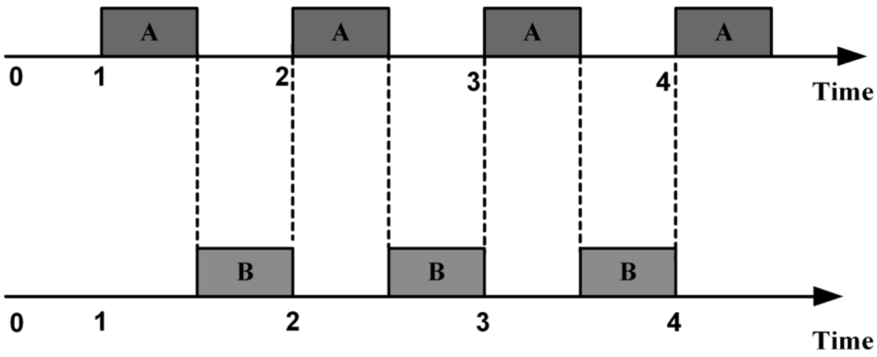


Figure 4-2 Illustrating the execution of co-operative tasks where one waits until the other finish execution

Baker and Shaw (1988) described co-operative schedulers based on the cyclic executive model (also called timeline scheduler). They defined a cyclic executive as *“a control structure or program for explicitly interleaving the program execution of several periodic processes on a single CPU; the interleaving time is done in a deterministic fashion so that execution time is predictable.”*

A co-operative scheduler achieves its basic operation by setting up a periodic timer interrupt to drive a schedule table every 't_{tick}' ms. An optimal value of t_{tick} is the greatest common divisor (GCD) of the task periods. The schedule repeats itself over a major cycle 'h' which is defined as the least common multiple (LCM) of the task periods (Liu, 2000). An example of a cyclic schedule is shown in Table 4-1 with t_{tick} = 25 ms and h = 100 ms. The timeline of the system is shown in Figure 4-3.

Table 4-1 Task specifications for a system with non-pre-emptive cyclic executive scheduler

Task ID	WCET (ms)	Period (ms)	Deadline (ms)
A	10	25	25
B	10	50	50
C	7	100	100

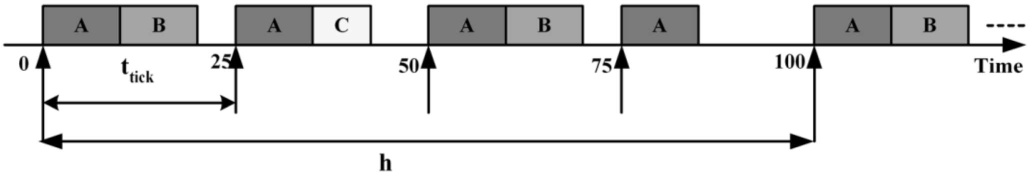


Figure 4-3 Schedule for the task set shown in Table 2-1 where t_{tick} represents the system tick and 'h' represents the major cycle for the system

The advantage of co-operative scheduling is its simple design and low context switching[4] overhead. According to Jeffay *et al.* (1991), non-pre-emptive scheduling naturally guarantees exclusive access to shared resources and data, thus eliminating both the need for synchronization and its associated

[4] It is the process of suspending the currently running task execution and starting a new one. This process involves the storage and retrieval of information (related to tasks) from the memory. Context switching takes up processor time, it is therefore regarded as an overhead (Cooling, 2003).

overhead . For such schedulers context switching only happens when the task finishes its execution (Labrosse, 1999) and the new ready to run task gets control of the processor. However, the major drawback of this approach is that systems scheduled with non-pre-emptive schedulers allow single tasking only i.e. only one task is active at a time.

4.2.2. Pre-emptive scheduling

Pre-emptive scheduling is priority-based and so it allows multi-tasking in the system. In this type of scheduling, among a set of ready tasks, one is chosen dynamically according to a priority order. Priorities may be seen as additional information that helps in determining an execution policy that satisfies all the constraints (Balarin, Lavagno *et al.*, 1998). In this type of scheduling, when a higher priority task becomes ready-to- run it can pre-empt the currently running lower priority task thereby taking control of the CPU. Once the higher priority task completes execution the pre-empted lower priority task is loaded again to complete the remaining part of its execution. As a consequence the context switching overhead is high but the main advantage is a better system response (Labrosse, 2000) compared with co-operative scheduling where the higher priority task can immediately take control of the processor. Figure 4-4 illustrates the operation of a system in which three pre-emptive tasks (A, B and C) are scheduled to run with task C as the highest priority task. Task ‘A’ being the lowest priority is allowed to run once all the high priority tasks finish execution.

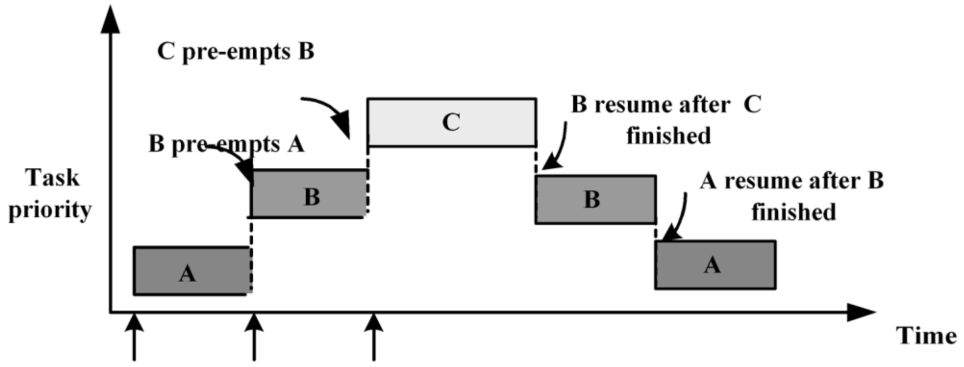


Figure 4-4 Illustrating pre-emptive tasks where a higher priority task pre-empts the lower priority task upon its arrival

Please note that pre-emption allows the scheduler to temporarily suspend the execution of a running task and allow another task to begin to execute. This requires that the context of the running task is stored (in some form of data structure) so the results of incomplete calculations are not lost. The context of the new task needs to be retrieved and switched in. This is why the process of context switching incurs overheads such as more processor time and memory utilization because of the extra use of these resources.

Researchers in the field of embedded systems development have different perspectives about pre-emption mechanisms and these mechanisms have been discussed by both opponents and proponents. Advocates of pre-emptive systems usually favor these systems because of their high responsiveness. Jia Xu appeared to be a strong opponent of priority-based/pre-emptive scheduling and opposed it because of the greater system overhead and significant difficulties involved in analyzing and predicting the system behaviour at runtime (Xu and Parnas, 1993; Xu and Parnas, 2000; Xu, 2003a; Xu, 2003b).

4.2.3. Offline scheduling

This is also known as pre-runtime or static scheduling. For offline or pre-runtime schedulers, scheduling decisions are taken at the design time and are used in cases where a complete set of task parameters are known (or calculated) in advance at the time of system design. Thus, offline schedulers are based on pre-runtime analysis of the system (Xu and Parnas, 1993). The entire plan of the system can be stored in a table and as a result the runtime overhead of the schedule is low (Cottet, Delacroix *et al.*, 2002). The main advantage of the pre-runtime scheduling is the overall reduction in the complexity of inspection and verification of timing properties of a system (Xu, 2003a). This is because the pre-runtime scheduling approach effectively reduces the number of possible cases of the actual code's timing behaviour by structuring real-time software as a set of cooperating tasks and imposing strong restrictions on the interactions between those tasks (Xu, 2003b). Due to this benefit of predictability, offline schedulers became the choice of a number of applications including flight control systems, process control systems and space shuttle avionics system (Ramamritham and Stankovic, 1994). However, the systems designed using offline scheduling are quite inflexible to environmental changes (Cottet, Delacroix *et al.*, 2002).

4.2.4. Online scheduling

This is also referred to as runtime or dynamic scheduling in the literature (Liu, 2000; Sha, 2004). In this type of scheduling, decisions are taken at runtime when a new task enters the system. In this case, no schedule is built at the design time (Cottet, Delacroix *et al.*, 2002). This approach is normally used in systems:

- In which it is likely that a task(s) can be added or removed at runtime.
- Systems that interact with evolving environments.

These types of schedulers suffer from high runtime overheads (Liu, 2000).

4.2.5. Static priority scheduling

This type of scheduling is also described as fixed priority scheduling in the literature (Leung and Whitehead, 1982). In this scheme, tasks priorities are assigned before the scheduler starts its execution and those priorities remain unchanged throughout the lifetime of the system. Examples of these types of schedulers are Rate Monotonic (RM) and Deadline Monotonic (DM).

In the case of the RM scheduler, task priorities are assigned based on their period; the shorter the period the higher will be the priority (Liu and Layland, 1973). In principle, RM is a pre-emptive algorithm which is based on fixed priority assignment (Kopetz, 1997) . The RM algorithm was proved to be optimal[5] by Liu and Layland in 1973. They demonstrated that if it is possible to schedule a task set using any fixed priority algorithm and meet all of its timing constraints, then RM can achieve this as well. Their observations about a RM scheduler are presented in Equation 4-1

$$U = \sum Ci/Pi \leq n(2^{1/n} - 1) \qquad \textbf{Equation 4-1}$$

Where:

U = CPU utilization factor

n = Number of tasks

C_i = Worst-case execution time of task T_i

[5] A schedule is feasible if all the tasks meet their deadlines. A scheduling algorithm is optimal if it is able to produce a feasible schedule for any schedulable task set (Cottet et al. 2002).

P_i = Period of task T_i

They proved theoretically[6] that with a RM scheduler every task can meet its deadline if the total CPU utilization is less than or equal to 69% and under the following assumptions:

- All tasks are periodic and independent of each other.
- The deadline of each task is equal to its period.
- The worst-case execution time of tasks is known.
- The context switching overhead can be ignored.

The main advantage of the RM algorithm is its flexibility during design and maintenance phases (Locke, 1992; Bate, 1998). A variant of RM scheduling is deadline monotonic (DM) also referred to as ‘Inverse deadline scheduling’ (Leung and Whitehead, 1982; Audsley, Burns *et al.*, 1991). It weakens the RM’s “period equals deadline” constraint by assuming that deadlines can be shorter than task periods. In this type of scheduling, fixed priorities are assigned to tasks based on their deadlines. Tasks with smaller deadlines are assigned with higher priorities. DM is optimal in the sense that if there exists a feasible fixed priority assignment schedule for a task set for which deadlines are shorter than periods then DM is also feasible for that task set. The utilization factor *‘U’* for DM is calculated in relation to a deadline rather than a period and is shown in Equation 4-2.

$$U = \sum Ci/Di \leq n\ (2^{1/n} - 1) \qquad \textbf{Equation 4-2}$$

Where:

[6] A complete schedulability test of this algorithm is derived in (Liu and Layland, 1973)

n = Number of tasks

C_i = Worst-case execution time of task T_i

D_i = Relative deadline of task T_i

DM scheduler provides the application designer with more flexible process model and is particularly useful when the task deadline is shorter than the period of the task (Audsley, Burns *et al.*, 1991).

4.2.6. Dynamic priority scheduling

In this case, the priority of each task is dynamically assigned and can be changed at runtime (Buttazzo, 1997; Buttazzo, 2005). Examples of dynamic priority schedulers are Earliest Deadline First (EDF) and Least Laxity First (LLF). In EDF, the priority of each task is dependent on its absolute deadline[7], the closer the deadline of a task is with respect to other tasks deadlines, the higher will be the priority (Liu and Layland, 1973).

The laxity of a task is the maximum time a task can be delayed without missing its deadline (Cheng, 2002). This is also referred to as slack time of a task. Buttazzo has provided a simple equation to calculate the laxity of a task in (Buttazzo, 1997) shown in Equation 4-3.

$$Laxity = Deadline - Arrival\ time - Execution\ time \qquad \textbf{Equation 4-3}$$

[7] A deadline can be measured from the start of the system power on, in which case it is called an absolute deadline, alternatively it can be measured from the start of the task period in which case it is a relative deadline.

The LLF algorithm assigns highest priority to the task that has shortest laxity or that needs to be executed most immediately.

4.3. Software architectures for embedded applications

Along with the choice of appropriate hardware, every embedded application is designed with underlying software architecture. Therefore in building reliable embedded systems software architecture is as important as its hardware counterpart. Software architecture defines the overall structure of the system in terms of components and an organisational principle that defines possible interconnections between these components (Dechering, Groenboom *et al.*, 1999). In addition the software architecture prescribes a set of rules and constraints governing the behaviour of components and their interactions (Boassan, 1995). Douglass Locke in his paper (Locke, 1992) has defined the principle design objective for the software architecture as: *"The architecture must be capable of providing a provable prediction of the ability of the application design to meet all of its time constraints."*

In the light of these definitions software architecture for embedded applications is described in terms of a task model which constitutes the system and the strategy used in scheduling tasks. As discussed briefly in Section 1.2, two main software architectures for designing embedded applications are ET and TT. The subsequent sections in this chapter are dedicated to a discussion on these two architectures in detail.

4.3.1. Event-triggered (ET) architecture

A 'trigger' is defined as a control signal that initiates an action (or a task) in an embedded computer system (Kopetz, 1997). This definition helps to draw a clear line between ET and TT software architectures. In a system designed with ET architecture, tasks are executed in response to the occurrence of particular events (Nissanke, 1997) and the peculiarity of these events are that their arrival times are not always known (or calculated) in advance. The occurrence of events may either be predictable (statistically or otherwise) but some events cannot be predicted deterministically. Therefore, Herman Kopetz (a strong supporter of TT architecture) has used a term 'chance events' (Kopetz, 1991) to describe these. Tasks in a purely ET design are of an aperiodic nature. Most ET systems that are in use today are designed with pure interrupt-driven and pre-emptive approaches. That is to say in such applications the appropriate code is directly executed inside an interrupt service routine (ISR) to handle the event raising the interrupt. A schematic view of handling interrupts in the system is shown in Figure 4-5.

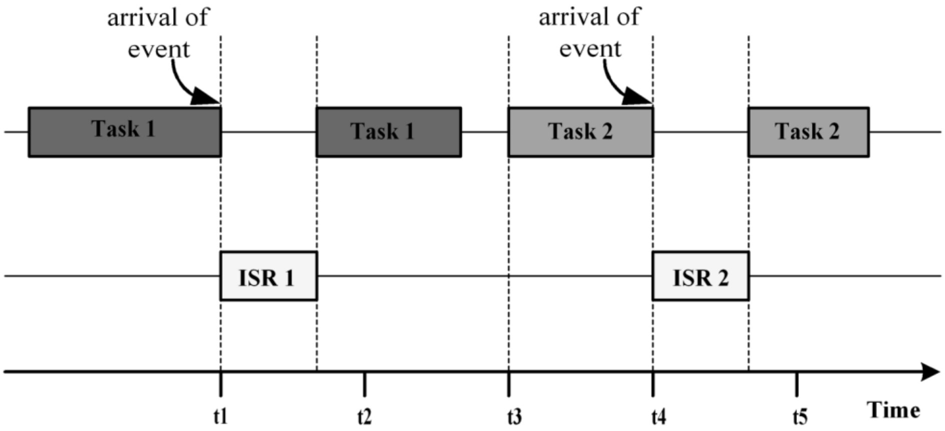


Figure 4-5 Illustrating the interrupt handling mechanism in an embedded application [Adapted from (Pont, 2001)]

Viewed from a high level perspective, interrupts provide a mechanism for creating multi-tasking in applications, i.e. the application is allowed to handle more than one task at a time using a single processor. The nature of ET architecture necessitates the use of pre-emptive/priority-based and online scheduling strategies in systems. Therefore, most of the systems (if not all) designed with ET architectures use pre-emptive/priority based scheduling and so their merits and demerits are associated with those scheduling techniques. ET designs are considered more responsive, flexible and easier to design as no prior knowledge related to task parameters is required at the design time. The main advantage of these systems is their ability to quickly react to asynchronous external events which are not known in advance (Albert and Gerth, 2003) and so they show a better real-time performance. In addition ET systems possess a higher flexibility and allow in many cases the adaptation to the actual demand without a re-design of the complete system (Albert and Bosch GmbH, 2004).

A pure ET design suffers from context switching overheads and unpredictability because of the lack of a priori knowledge of the event's arrival times and the way they are handled in such systems. Also in such systems the arrival of multiple (too many) interrupts at the same instant of time can deteriorate the situation and techniques for minimising this are required. For example, 'Interrupt nesting' is a mechanism which allows further interrupts to occur while a currently running interrupt is in service (Sloss, Symes *et al.*, 2004). A three-level nested interrupt is shown in Figure 4-6.

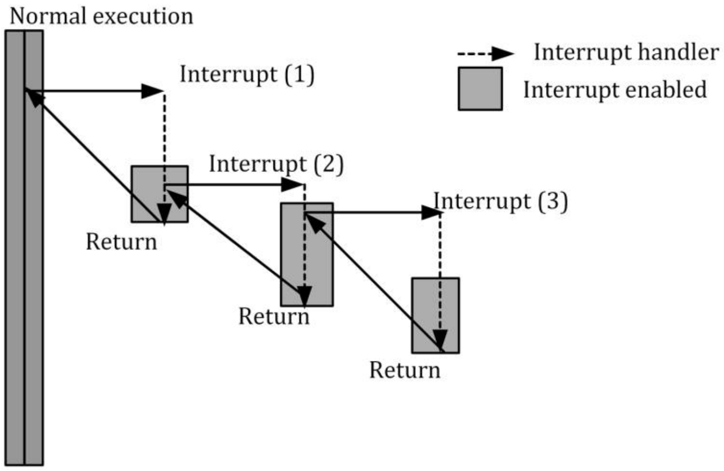


Figure 4-6 Illustrating the process of task switching as various interrupts occur in a nested interrupt based system

However, depending upon the depth or level of nesting this situation could lead to increase in system complexities. In such a case complexity means that designers have to balance efficiency with safety and defensive coding style that assumes problems will occur (Sloss, Symes *et al.*, 2004). There are two ways of looking at this feature and on the positive side this could make the systems more responsive as the higher priority tasks will run immediately as their associated interrupt arrives. On the other hand, this could increase the complexity of the system because of nested pre-emption, context switching overheads, and the possibility of malfunctioning due to stack overflow and register corruption.

4.3.2. Time-triggered (TT) architectures

In systems designed with TT architectures, the triggering signal for tasks is, 'time'. The invocation of tasks is dependent on a single timed signal which is known (or calculated) in advance and is usually a periodic timer interrupt. Thus, TT designs have only one interrupt enabled, all the other inputs to the system are polled. The period of time marked by the timer interrupt is identified as the system tick interval (Pont, 2001). This type of solution suits many control applications where the data messages exchanged in the system are periodic (Kopetz, 1997). In a purely TT designed system tasks are characterised by static or pre-runtime scheduling (Locke, 1992).

TT architectures can be adopted in various ways – for example – a simplest TT design implemented with co-operative scheduling called time-triggered co-operative (TTC) design is based on a cyclic executive model and its implementation is discussed in detail in (Pont, 2001) . In a typical TTC design a timer is set to generate interrupts on a periodic basis (with tick intervals of around 1 ms being typical). In most cases, the tasks will be executed from a "dispatcher" function, invoked after every tick. The dispatcher examines each task in its list and executes any tasks which are due to run in this tick interval. The scheduler then places the processor into an "idle" (power saving) mode where it will remain until the next tick. As an example consider a set of three tasks A, B and C with specifications shown in Table 4-2. Figure 4-7 is portraying the TTC scheduler for the task set of Table 4-2.

Table 4-2 Task specifications for task set required to be scheduled with a co-operative scheduler

Task ID	Start time (ms)	Period (ms)
A	0	2
B	0	1
C	0	1

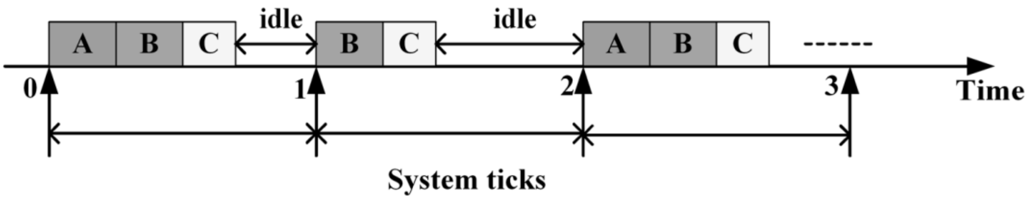


Figure 4-7 TTC schedule for the task set shown in Table 4-2

The TTC scheduler is an attractive option for designers of embedded applications because of its simple design and highly predictable behaviour. Being non pre-emptive in nature, tasks runs to completion once started which eliminates the need to implement any shared resource access control mechanism. Also, it involves low context switching overhead and low task jitter.

However, the main drawbacks of the TTC scheduler are its fragility and lack of flexibility when it becomes necessary to adopt changes in the system requirements. In such a case the entire schedule needs to be reconstructed. Another drawback is the poor responsiveness when the system has to handle any long task as it will block any other tasks until it is finished and consequently this may lead to the blocked tasks missing their deadlines (Locke, 1992; Bate, 1998).

In applications in which TTC does not appear as an appropriate choice, other options are available that allow pre-emption in the system. Under such circumstances, the usual choice is a fully pre-emptive design, however some previous studies have described ways in which support for a single TT pre-emptive task can be added to a TTC scheduler to form a "time-triggered hybrid" or TTH scheduler (Pont, 2001). The TTH scheduler supports:

- A single pre-emptive task which can interrupt co-operative tasks.
- Any number of co-operatively scheduled tasks

To understand the operation of a TTH scheduler consider the task set presented in Table 4-3, for which the timeline is shown in Figure 4-8 (the downward arrows are representing the system ticks).

Table 4-3 Task set specifications required to be scheduled with a hybrid scheduler

Task ID	Task Type	Start time (ms)	Execution time (ms)	Period (ms)
C	Co-operative	0	3	5
P	Pre-emptive	1	0.2	1

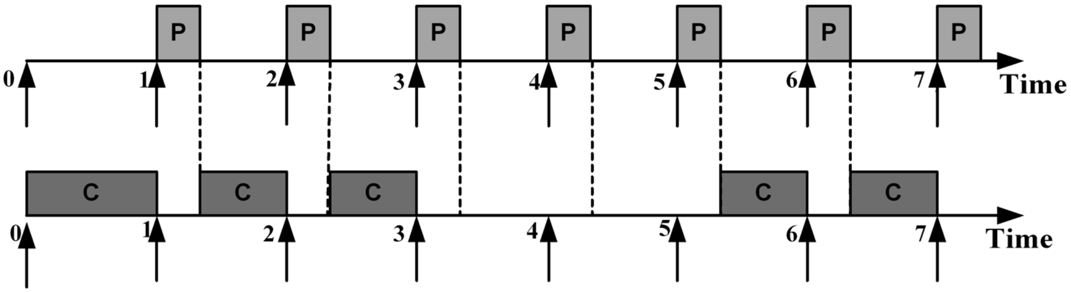


Figure 4-8 Illustrating the operation of a TTH scheduler

In many designs, the pre-emptive task will be used for periodic data acquisition typically through an analogue-to-digital converter (ADC) or similar device. Such a requirement is common in, for example, a wide range of control systems (Buttazzo, 1997).

TTC and TTH designs are not the only options for TT designs but there are other possibilities to implement with fully pre-emptive architecture such as time-triggered rate monotonic (TTRM) and time-triggered deadline monotonic (TTDM) designs discussed in (Maaita, 2008).

The predictable nature of the TT approach makes it the usual choice for many safety related applications such as fly-by-wire aircrafts and drive-by-wire passenger cars (Ayavoo, Pont *et al.*, 2005). As the schedule for such systems is available offline, these systems can be tested exhaustively before their actual implementation. This in-turn contributes to system reliability (Xu and Parnas, 2000). On the other hand, TT systems are considered ‘difficult to design’ or inflexible as adding new tasks in the system (in some cases) will require changes to be made in the entire schedule.

4.4. Event-triggered versus time-triggered architectures

In the published literature, both ET and TT architectures have been widely discussed and debated by opponents and advocates. For example research presented in (Kopetz, 1991; Kopetz, 1993; Albert and Bosch GmbH, 2004; Claesson and Suri, 2004; Obermaisser, 2005; Scarlett and Brennan, 2006; Scheler and Schroder-Preikschat, 2006) has discussed the comparative features of both of these architectures. A combined agreement of all such studies is in favour of ET systems for responsiveness and flexibility in design while TT systems are favoured for their predictable nature.

To summarise – TT concepts potentially provide a higher dependability while ET systems possess a higher flexibility (Albert and Bosch GmbH, 2004). It is impossible to predict the concrete state of an ET system at a given point in time, because only few assumptions on the occurrence of aperiodic and sporadic events can be made, while this is much less problematic in TT systems (Scheler and Schroder-Preikschat, 2006). The TT approach offers considerably less flexibility compared to the ET method. Instead, the TT approach offers highly predictable communication behaviour on account of its static slotted access approach; predictability being a significant factor in its usage for design of safe and reliable systems (Claesson and Suri, 2004). For a distributed system where communication between different nodes is required Scarlett and Brennan voiced their opinion that the TT approach has been favoured in safety-critical systems because it can ensure that no communication delays occur by assigning dedicated communication time windows to specific nodes. The pure ET approaches have been unable to guarantee communication delays and have thus not gained acceptance (Scarlett and Brennan, 2006).

Testing of embedded applications before their actual implementation is an important issue and explored by researchers in the field. TT systems are inherently easier to test than ET systems. This is because, processes in TT systems work in lockstep with time and requires rigid assumptions about the behaviour of the environment while, ET systems react to events in the environment as they occur (Birgisson, Mellin *et al.*, 1999). Since no detailed plans for the intended temporal behaviour of the tasks of an ET system exist, it is not possible to perform “constructive” performance testing at the task level (Kopetz, 1993).

Table 4-4 provides a summarized view of the comparative features of ET and TT architectures discussed in the section above.

Table 4-4 Comparative view of event-triggered and time-triggered architectures

Characteristic	ET Systems	TT Systems
Design	Easier	Difficult
Testing	Difficult	Easier
Responsiveness	High	Low
Flexibility	High	Low
Predictability	Low	High

4.5. Discussion

In this chapter, a detailed discussion on software architectures for embedded applications has been presented. In conjunction with this different scheduling schemes discussed in the literature have also been described. Designing a reliable system is mainly dependent on making the correct choices and taking appropriate decisions about scheduling scheme and software architecture.

4.5.1. Selection of appropriate scheduling scheme

At a glance, it appears there are too many options available for the scheduling of tasks in embedded applications such as co-operative, pre-emptive, offline, online, static priority based scheduling, and dynamic priority based scheduling. The fact is, they are not mutually exclusive and most of them are interrelated. For example, pre-emptive scheduling is actually priority based (static or dynamic) and is online scheduling as most task parameters are revealed and resolved at runtime. Co-operative or non-pre-emptive scheduling is offline or pre-runtime scheduling as it involves designing a complete schedule at the design time. At a higher level, there are two broad categories of scheduling: Co-operative and Pre-emptive.

Commonly used schedulers with offline scheduling technique are based on the cyclic executive model. Schedulers based on the cyclic executive model can be an attractive option when designing real-time systems due to their simplicity and predictable behaviour (Bate, 1998). Also the fact that no shared resource-access control is required as the task runs to completion once started and that there is no pre-emption is an added advantage (Locke, 1992). The most serious concern shown by researchers about the cyclic executive based models is their fragility i.e. small changes to the timing of a particular task can mean that the developer has to make substantial changes (Shaw, 2001). The other main concern reported against co-operative scheduling and the cyclic executive model is task overrun i.e. concerning tasks bearing execution times longer than the length of the minor cycle. If a task is overrun the problem may not even be detected and could seriously affect system behaviour. Buttazzo has mentioned these problems in these words in (Buttazzo, 2005) *"Co-operative scheduling is fragile during overload situations, since a task exceeding its predicted execution time could generate (if not aborted) a domino effect on the subsequent tasks causing their execution to exceed the minor cycle boundary"*

Priority-based or pre-emptive scheduling is considered a better approach because of its ability to handle dynamic situations, but it costs in the form of increasing system complexity. Because pre-emption is allowed shared resource access handling has to be provided in the system. Without proper resource access control, shared data structures could end up in an inconsistent state (Kalinsky, 2001).

Developers and designers of embedded applications use techniques such as semaphores to handle shared resource access but they must be implemented carefully. Such systems are vulnerable to problems known as priority inversion and deadlock. Priority inversion is a phenomenon in which a lower priority task locking a shared resource forces a higher priority task to be blocked as it waits for that resource to be released (Sha, Rajkumar *et al.*, 1990; Buttazzo, 1997; Cottet, Delacroix *et al.*, 2002). A deadlock may occur if a higher priority task is waiting for a resource held by a lower priority task, while lower priority task is simultaneously waiting for a resource held by a higher priority task. In such a case none of the tasks are able to proceed and – as a result – a deadlock is formed. Both priority inversion and deadlock can be solved by using protocols such as priority inheritance protocol (PIP) and priority ceiling protocol (PCP) (Sha, Rajkumar *et al.*, 1990).

Michael Barr a strong critic of the pre-emptive scheme has expressed his views against pre-emption as ‘Perils of pre-emption’ (Barr, 2006). According to him: *“Several workarounds to priority inversion exist, but they always result in wastage. For example, under the Priority Ceiling Protocol each shared* *resource has a priority at least as high as the highest-priority task that ever uses it. Unfortunately, this popular workaround results in another violation of the basic assumption of priority-based pre-emptive scheduling: A medium priority task may NOT use the CPU because a low-priority task is running and using a resource sometimes used by a high-priority task"*

The above discussion highlights the fact that there is no silver bullet for scheduling tasks in real-time applications. There is a price to pay for every benefit so the perfect choice of the scheduling scheme depends on application demands and requirements. Indeed, the choice of a scheduling technique is often the result of a hard compromise between conflicting criteria! (Balarin, Lavagno *et al.*, 1998).

4.5.2. Selection of appropriate software architecture

It is the scheduling strategy which actually defines the underlying software architecture of an embedded application. Two key software architectures widely discussed in the literature are ET and TT. In Section 2.5 a detailed discussion and comparison of ET and TT architectures has been presented. It has been argued that ET architectures are preferred for their responsiveness, flexibility and ease of design, while TT architectures are mainly preferred because of their predictable nature. This feature is a significant advantage for any kind of dependable system (Scheler and Schroder-Preikschat, 2010).

The question still remains that for any application, what is the criterion to choose a perfect architecture? The question of choosing ET or TT architecture has no clear answer and there are no rules defined for the developers of embedded applications which they can follow without more in-depth investigation of the application requirements. The implications of choosing either the TT or ET approaches are not particularly easy to assess, especially within a complex system where the various design considerations span contending trade-offs with flexibility, efficiency, predictability and dependability (Claesson and Suri, 2004). In order to compare the performance of an application both with ET and TT architectures, Kopetz has demonstrated an example of an alarm monitoring system in (Kopetz, 1997). The system has a distributed architecture consisting of ten interface nodes connected to a controlled object and an alarm monitoring node that processes the alarms and displays them to the operator. The system is implemented both with ET and TT architectures. Figure 4-9 compares the performance of the ET solution versus the TT solution.

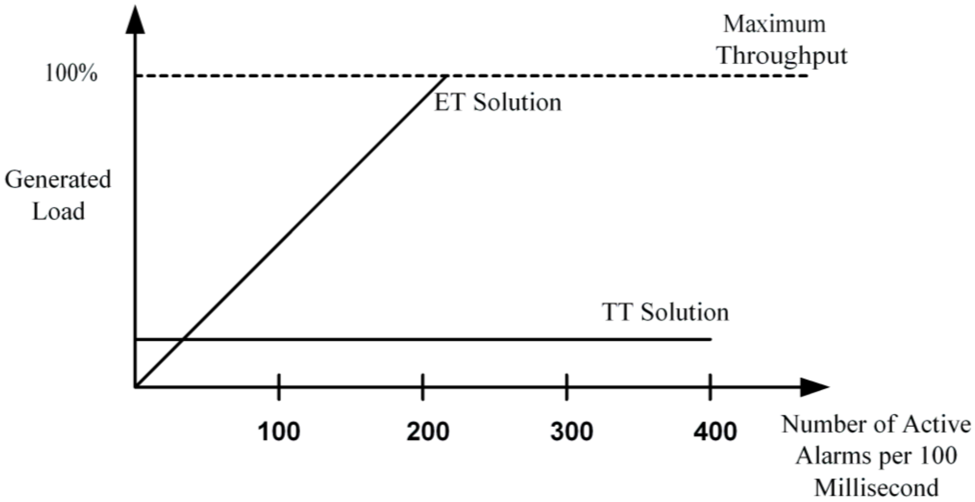


Figure 4-9 Load generated by ET and TT solutions of the alarm monitoring system [adapted from (Kopetz, 1997)]

The break-even point between the two implementations is at about 16 alarms per 100 milliseconds, which is about 4% of the peak load. If less than 16 alarms occur within a time interval of 100 msec then the ET implementation generates less load on the system. If more than 16 alarms occur, then TT implementation is more efficient.

This example clarifies the idea that systems based on ET architecture are not always more responsive than systems based on TT architecture, as in high-generated load situations or in worst-case scenarios the efficiency of ET based system is not verified. Thus, when considering worst-case requirements the TT approach is more resource efficient than the ET one. However, when considering average-case requirements, TT is considerably more resource hungry when compared to ET systems. Consequently, by assessing a system according to its worst-case requirements (e.g. in hard real-time systems) the TT approach tends to be less expensive than the ET one (Almeida, Pedreiras *et al.*, 2002). Thus selecting the appropriate architecture is not that straightforward and involves trade-offs depending upon the situation. As Albert and Bosch noted, "*In general, reality is neither black nor white but rather gray. Thus, it depends on the application whether a time-triggered or an event-triggered behaviour is more suitable*" (Albert and Bosch GmbH, 2004).

4.5.3. Current trends

This Chapter has presented a detailed account of software architectures as discussed in the literature and, it is worth investigating where the current state-of-the-art of software architectures for modern embedded applications lies with respect to system reliability. There isn't a straight forward response to this as modern embedded applications can be found in a huge variety of devices and systems in almost all industrial and business sectors from games and home appliances to transportation and military applications. There is an equally diverse variety of design requirements in these applications but reliability is always a desirable feature for the customer. The tolerable degree of reliability of course varies depending on the application but for safety-critical applications which involve risks to human life, reliability is of paramount importance.

To maintain a benchmark for the functional safety of devices the International Electro-technical Commission (IEC) has introduced a set of standards for the general market called IEC 61508. Because IEC 61508 is very general and as the state of the industry cannot be reflected in a general standard, the specific standards are described for different types of applications such as IEC 60730 for white goods, ISO 26262 (published recently in November 2011) for automotive industry and DO-178 for aerospace. By the use of these standards both systematic and random failures can be reduced and managed to ensure system reliability and functional safety. For example for ensuring functional safety of road vehicles, ISO 26262 has reserved a separate part (Part 6) which is further divided into various sections on guidelines for software architectural design. The standard does not favour or oppose any specific software architecture but has defined guidelines to follow to get safe and reliable systems. Some of the principles defined for the software architectural design are: hierarchical structure of software components, restricted size of software components, restricted coupling between components, appropriate scheduling properties and restricted use of interrupts. Some mechanisms are defined for error detection and error handling at the software architectural level. For example the recommended error detection mechanisms are: detection of data errors, external monitoring facility and control flow monitoring. Mechanisms defined for error handling are: static recovery mechanism and graceful degradation. There is strong emphasis on testing and verification of the software architectural design and methods recommended are informal verification by walkthrough or inspection of the design, semi-formal verification by simulating dynamic parts of the design or by the proto type generation or animation, formal verification which involves rigorous mathematical models, control flow and data flow analysis for the system.

In order to achieve systems which are safety complaint and certified by these standards the competitors in the automotive industry are inclined towards adhering ISO 26262 guidelines. As already mentioned above, there is no single architecture which is recommended or rejected by ISO 26262 it is still to be decided by the software designers and developers which architecture has the ability to follow the guidelines.

4.6. Conclusions

This chapter has explored software architectures and their related background discussed in the literature. It described the real-time task model, scheduling schemes and their relevance with designing software architectures. It has also presented a detailed comparison of ET and TT architectures and highlighted the ambiguities involved in deciding about the appropriate choice of scheduling scheme and software architecture for a particular application. These difficulties sometimes appear as a demand in the change in the current architecture of an application and designers often have to face the challenge of migration in software architectures. The next chapter will discuss and review some real examples of applications quoted in literature that faced the issue of migration due to problems with their existing architectures.

CHAPTER 5. MIGRATION OF ARCHITECTURES IN EMBEDDED APPLICATIONS

5.1. Introduction

As mentioned in Section 1.2, the main objective of this research is to improve the reliability of existing applications by making them more predictable. CHAPTER 4 described how software architectures initially chosen for designing an application play a vital role in system reliability. In spite of taking very careful decisions about choosing the architecture during an application design, designers/developers of embedded applications may have to face unexpected consequences of the application in use afterwards. Software architecture usually demands some modifications at a later stage of its life cycle or in some cases radical alteration becomes crucial because it is not easy to simulate all the conditions of usage of the software at design time. Therefore sometimes an application may have to pass through a process of architecture migration in order to improve reliability. This chapter is dedicated for a discussion on the issues that arise when migration in embedded applications is required. Section 5.2 presents a general discussion on the migration of embedded applications and how this process has been defined by different researchers in the field. In connection with this, Section 5.3 will shadow light on various characteristics and drivers of migration in embedded software development. Section 5.4 presents a discussion on dependencies between different components in embedded software and how change to one part may affect the other part(s). Section 5.5 presents a discussion on few existing techniques found in literature to deal with the migration process and chapter conclusions are presented in Section 5.6.

5.2. Related terminology

In some cases, applications initially designed with a specific architecture even with the consideration of all the issues/trade-offs related to that architecture may prove to be an incorrect choice later on at some point in their life cycle. It is quite possible that application requirements may change with respect to time. In such situations, it becomes crucial to make necessary changes in the existing design or sometimes completely alter the application architecture. To explain this situation, researchers in the field have used different terminologies such as ‘Re-engineering’ (Chikofsky and Cross, 1990; Madisetti, Jung *et al.*, 1999; Park, Ryu *et al.*, 2006), ‘Change and Customisation’ (Eckert, Clarkson *et al.*, 2004) , and ‘Migration’ (Scheler and Schroder-Preikschat, 2010). This section will look at what different researchers mean by their described terminologies and how they are related.

According to the definition provided in Chikofsky and Cross (1990) *“The re-engineering generally includes some form of reverse engineering followed by some form of forward engineering or re-structuring. This may include modifications with respect to new requirements not met by the original system.”* According to Park *et al*. (2006), *“Re-engineering problem is defined as a sequence of activities involving reverse engineering, system alteration, and forward engineering.”* In this process, reverse engineering captures an understanding of the behaviour and structure of the system, alteration modifies the current structure and forward engineering aims to incorporate new functionalities in the system.

Eckert *et al* (2004) has discussed the change and customisation process for complex engineering domains used in aerospace. They have categorised change process as 'Emergent change' and 'Initiated change'. The emergent change caused by the state of the design where problems occurring across the whole design and throughout the product life cycle can lead to changes. The initiated change arises from an outside source typically a new requirement from customers, certification bodies or manufacturers.

In the context of research presented in this thesis 'Migration' specifically refers to the transitioning of existing software architecture of an embedded application from ET design to TT design.

At a higher level, migration issues related to embedded applications have been identified previously in the literature focusing on legacy embedded applications (Madisetti, Jung *et al.*, 1999; Mosley, 2006), complex engineering systems (Eckert, Clarkson *et al.*, 2004), applications related to control systems (Hebert, 2007) and military applications (Oest, 2008). All the previous research work and experts in the field agree that migration brings new challenges for developers and designers of embedded applications. According to Park *et al.* (2006) "*Performance re-engineering problem for an embedded system poses serious challenges to developers.*" Because of the complexities involved in the process migration requires detailed and careful planning beforehand. According to Hebert (2007) "*Breaking a large migration project into smaller and more manageable phases reduces risk and down-time.*" Oest (2008) is of the opinion that "Migrating complex embedded software – particularly in applications requiring real-time response and a high degree of safety criticality – can be a costly, time consuming, and risky process requiring code changes, re-testing, and even re-certifying.”

All the above statements reflect a common agreement that introducing change in the existing design/architecture of embedded applications offer new challenges and this research aimed to explore ways to facilitate the designers in facing the big challenge.

5.3. The need for migration in embedded software

This section will focus on the issues which drive the developers of embedded applications to migrate their applications from the existing platform to a new one.

The previous section illustrated the complexities involved in the process and this can often be a deterrent to developer of embedded software who as a result may wish to avoid the process. However, in cases where reliability and safety in systems are two primary goals, migration is the best solution. Though achieving reliable and safer systems is indirectly linked to many other factors such as system upgradability, desired functionality, system performance in normal and in peak load situations etc. Consideration about these factors needs to be the part of decision-making about whether is it the time to migrate the application. The main drivers for software migration are explained in Sections 5.3.1 to 5.3.4 below.

5.3.1. Hardware modifications

Several hardware modifications in systems can lead to migration in embedded software. This may include either hardware upgrades to avoid obsolescence or hardware enhancement if the application demands improvement in performance.

Mostly the safety-critical embedded systems such as those use in aerospace and defense (missiles etc.) bears longer life cycles. This leaves a possibility for individual components used inside such systems becomes obsolete years before the system itself expires and therefore leads to a migration. An embedded application in its entirety works with the co-ordination of various hardware and software components the upgradability of individual components in the underlying embedded target hardware also calls for migration. This is because the real-time operating system (RTOS) or more primitive runtime support that was available on the old hardware may not match the requirements of the upgraded system. As chief operating officer of DDC-I Inc. Ole N. Oest (2008) observes the possible triggers of migration as: when the host computer becomes obsolete, development tools that were originally used are no longer supported, change in peripherals or changes in bus protocols, augmentation of new system functionality, certification requirements imposed on the system and expertise in the application tools or languages is lost.

Migration is also possible for porting the application to more enhanced hardware. For example, Matassa (2011) has reported porting software from

PowerPC (PPC) to Intel® architecture. The main drivers for this migration are described as established performance of Intel architecture, its ability to meet time-to-market constraints and availability of development tools that help to implement, debug and tune the software performance. The major steps as described in the migration between Power PC and Intel are to make the code compatible to the target hardware and optimise the code for performance to run on Intel® architecture core. Similarly, some applications are recommended to port from 8051 microcontroller to ARM Cortex™-M processors for their higher performance (speed up to 135MHz), more memory, tool support, efficient interrupt handling and better debug facilities (ARM, 2012).

Another aspect that is considered while migrating embedded applications is to move the architecture from single-processor systems to multi-core designs.

5.3.2. Requirements to meet certification standards

With the introduction of standards such as DO-17B/C for airborne systems, and ISO26262 - a functional safety standard in the automotive sectors - many developers are required to re-assess existing designs and begin a process of design migration in order to improve system reliability and thereby meet certification requirements. Such a migration process can present many challenges for an organisation, not least because long-established (and possibly rather informal) working practices can be seen to be under threat.

5.3.3. System expansion in later stages of the life-cycle

Expansion in system functionality is often required at later stages of a system life cycle. If the initially designed architecture is not provided with enough room for future expansions could be a main driving force in migrating embedded applications. The example of F-18 Mission Computer (Shepard and Gagne, 1990) discussed in Section 2.2 leads to a complete change in the underlying software architecture when the system was not able to observe all the timing constraints upon adding new software components.

5.3.4. Incorrect system functionality

Incorrect system functionality or poor system performance observed at later stages in a system life cycle can also lead to changes in the underlying architecture. A practical example is that of a Sony cell processor discussed in Section 2.4 designed to run video games. This showed an extremely slow response in situations of high load (Turley, 2009) and was migrated from interrupt-driven to a software polled design. The image data lost was observed for an image acquisition embedded control application (Kubinger and Humenberger, 2004) discussed in Section 2.3. In both cases change in the software design of the application was ultimately required for the performance improvement.

5.4. Dependencies between components in embedded software

Many components in an embedded system are interlinked and changes to one part propagate changes to the other interlinked parts of the system. At the simplest level a well-structured application code in an embedded application is divided into several parts such as system, scheduler, device drivers and application related tasks. All these parts are tightly interlinked and provide data for each other for proper functionality of the application as shown schematically in Figure 5-1.

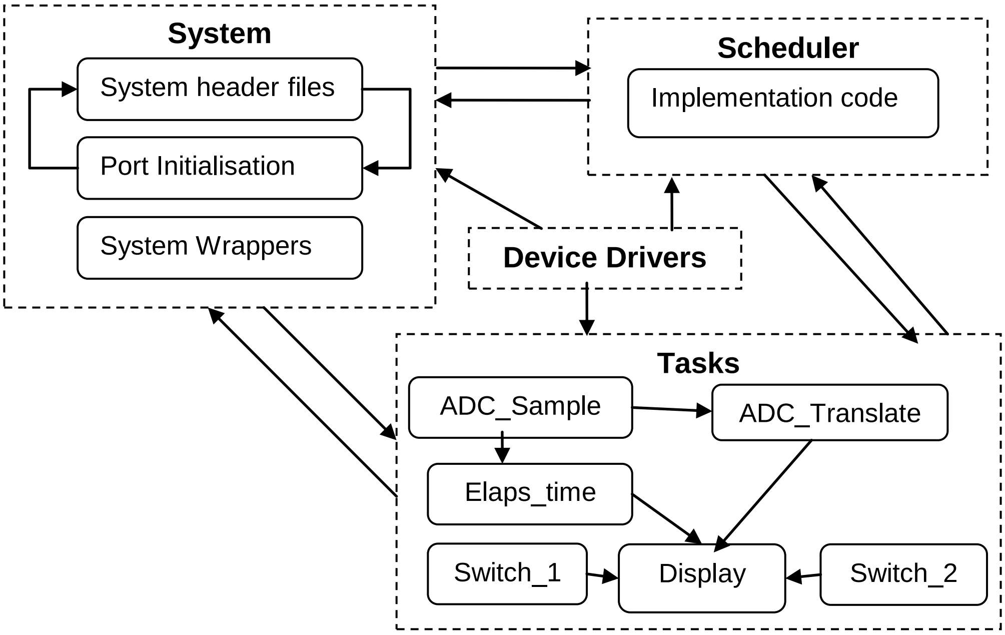


Figure 5-1 An illustration of dependencies between different software components of a simple embedded application

This division into parts is essential to make the code portable and reusable across the different applications which run on the same hardware platform. These include code files for system initialisation such as setting up the oscillator frequency, Phase-Lock Loop (PLL settings), the memory map and interrupt mapping for the system. For a more complex application the system functionality can further be divided into various sub systems as shown in Figure 5-2.

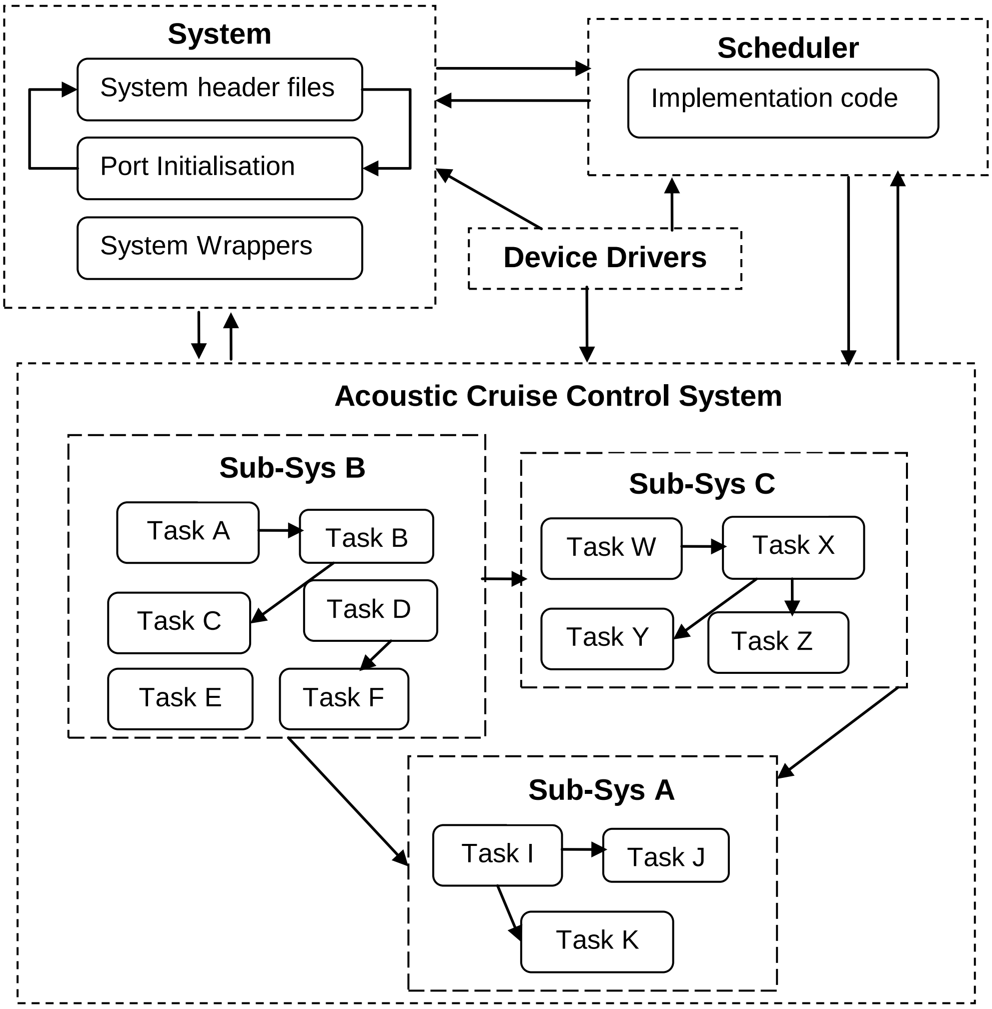


Figure 5-2 Dependency between different components in a complex control system

For example for an Acoustic Cruise Control System (ACCS) which is designed with several sub-systems where different tasks need to communicate to provide a desired functionality. To provide a complete system functionality different sub systems are also required to communicate, for example a sampler calculates the vehicle speed and then send it to the actuator node where a Proportional Integral and Derivative (PID) algorithm is used to calculate the throttle position.

In such a complex system changes made to any sub-system can potentially affect the overall communication between the tasks and other sub-systems which are dependent on the sub-system being changed. It is quite possible that some parts of the system remains unchanged for example ‘Device Drivers’ once designed are always available to use independent of any changes in the underlying software architecture while others have the tendency to absorb or proliferate the change. For example Eckert *et al* (2004) has classified different parts of a system with regards to change propagation as Constants (unaffected by change), Absorbers (absorb more change than cause), Carriers (equally absorb as much as they cause) and Multipliers (generate more change than they absorb).

5.5. Existing techniques in the literature

This section provides brief details of some of the existing techniques found in the literature for meeting the challenges of software migration.

5.5.1. Re-engineering of legacy embedded applications

VP Technologies, Inc. (VPT) a company based in Georgia, USA, deals in the business of re-engineering and has developed a technique for re-engineering legacy embedded applications[8] (Madisetti, Jung *et al.*, 1999). Their technique

[8] Legacy systems are hardware and software systems that require upgrading for reasons such as hardware obsolescence, change in requirements of functionality, better form in terms of size, weight, power and volume and decreased maintenance and life-cycle support costs. Another reason is the availability of superior algorithms, architectures and technologies that meet the system specifications at lower costs (Madisetti *et al,* 1999).

is based on virtual prototyping[9] accompanied by their tools and libraries and simulation/synthesis models. Their approach is based on evaluating the cost and benefits of re-engineering while performing hardware/software co-simulation. Their proposed re-engineering process is divided into three stages (see Figure 5-3).

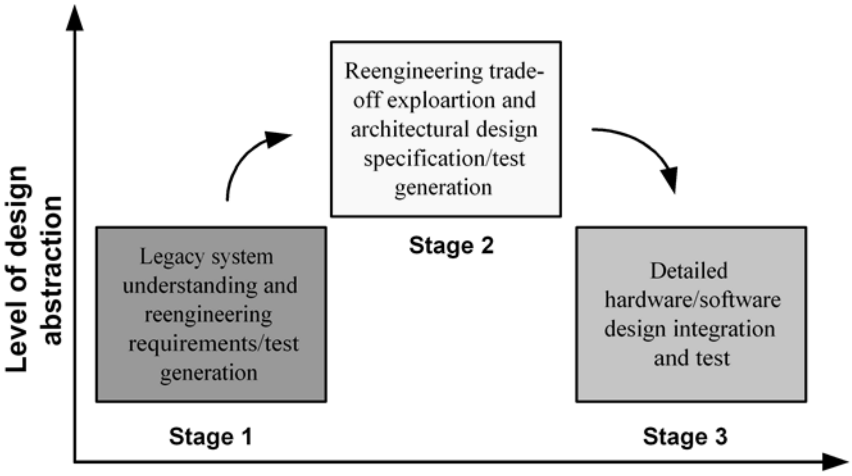

Level of design abstraction
Reengineering trade-off exploartion and architectural design specification/test generation
Stage 2
Legacy system understanding and reengineering requirements/test generation
Stage 1
Detailed hardware/software design integration and test
Stage 3

Figure 5-3 Three stage process for re-engineering legacy embedded applications [adapted from (Madisetti, Jung et al, 1999)]

Stage 1 focuses on design intent abstraction (or reverse engineering) to develop an executable virtual prototype of the legacy system in a language such as VHDL (VHSIC Hardware Description Language) or UML (Unified Modeling Language). Stage 2 involves decision making about the right architectural design and test specifications while Stage 3 completes the detailed software design, system integration and testing.

This technique is useful but has limitations in the sense that being based on virtual prototyping it is highly tool specific and the tool suite is supported for

[9] It is a technique involving Computer Aided Design (CAD) and Computer Aided Engineering (CAE) software to validate a design before committing to make a physical prototype (Source: Wikipedia).

specific hardware only for example, processors Mil-Std-1750, the ADSP Sharc, and the TI-62.

5.5.2. Real-Time System Compiler (RTSC)

The latest research on the migration of software architectures in embedded applications has been undertaken at Friedrich-Alexander University Erlangen-Nuremberg. RTSC (Scheler and Schroder-Preikschat, 2010) is a recent development and is a compiler based tool developed to ease the automated migration from ET systems to TT systems. The design of the RTSC is based on four main components (see Figure 5-4): Front-End, Analyser/Composer, Checker and Back-end.

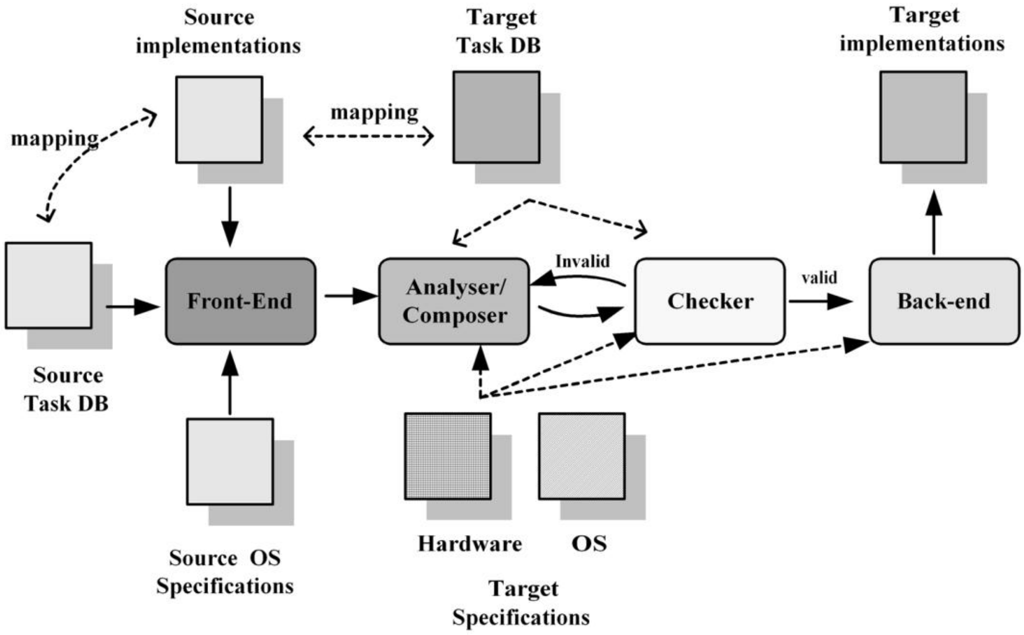


Figure 5-4 Design of the RTSC [Adapted from (Scheler and Schroder-Preikschat, 2010)]

To describe real-time systems Scheler and Schroder (2006) has introduced a representation called the ‘Atomic Basic Block’ (ABB). An ABB is a section of the control flow that ensures the consistency of the data that is affected within this ABB (Scheler and Schroder-Preikschat, 2006). Primarily, ABBs are arranged in three different graphs: a control flow and data flow graph to represent the flow of control and data between ABBs respectively, and a mutual exclusion graph which is undirected and represent mutual exclusion constraints among ABBs.

To transform the system, the basic structure of the source and target real-time systems are stored in task databases (Task DBs). These databases describe the source and the target real-time systems as collection of ABBs. Both source and target databases contain all the specifications for the system to be translated and are provided as inputs to the RTSC. The Front-End is the compiler front-end (which is programming language specific and OS-dependent) that transforms the source implementation into the ABB representation. These ABB graphs are then handed over to the Analyser/Composer component of the tool. This component analyses the requirements of the target and performs necessary steps such as calculation of hyper-period, WCET-analysis and scheduling in order to map the directed non-cyclic ABB graphs provided by the front-end onto a TT execution environment. If required at the end, a Checker verifies that certain temporal constraints recommended in the target Task DB are accomplished. Finally, the Back-end generates the code that can be executed by the targeted RTOS.

- **Limitations of the RTSC**

Although the RTSC is a good attempt to automate the complex migration process, it works only within certain boundaries. Being a compiler based tool it is able to handle only a certain type of real-time systems having a specific structure. In the author's words, *"it is not very promising to support arbitrary applications and we demand for real-time applications having a specific structure"* (Scheler, Mitzlaff *et al.*, 2007). The front-end of the tool is programming language specific and is based on certain assumptions as described by the authors *"our front-end still suffers some restrictions and thus, implicitly relies on some assumptions…Furthermore, we expect the application itself and the OS API to be well-formed"*(Scheler and Schroder-Preikschat, 2010). By 'well-formed' they are referring to their assumptions at the application level (for example, the same instance of a semaphore is not re-used for a different purpose at a different location). On the OS level their assumption is that the same system call is not used for different purposes too. For practical applications, such restrictions are quite difficult to follow especially in a resource-constrained environment which imposes restrictions on memory and power consumption.

5.6. Conclusions

This chapter has aimed to clarify and define the concept of migration and present the views of experts on the challenges related to migration. It has also discussed the reasons for migration and the dependency between different components used to design an embedded application. The chapter has also introduced the work carried out by previous research work on the migration of embedded applications. It was evident in studying the migration by developers of a variety of embedded applications from interrupt-driven to time-driven architectures (examples described in CHAPTER 2), that there was neither the support nor the tools available to enable a unified approach to meet these challenges. There is clearly a case for a 'best-practice' approach to rectify this situation and address the migration issues. This would minimise any mistakes and avoid repeatedly re-inventing methods that could become part of a generic toolkit that would streamline the migration process. The concept of design patterns originated from this need to capture such 'best practice' expertise and the next chapter will provide a detailed account of their development.

CHAPTER 6. DESIGN PATTERNS

6.1. Introduction

The terms “design pattern” and “pattern” are often used interchangeably. According to the definition provided in the Oxford English Dictionary (OED) available online at (OED, 2012) a pattern is *“…Something shaped or designed to serve as a model from which a thing is to be made; a design, an outline, an original”.* This definition highlights the fact that patterns can be of use in many disciplines which involves creativity and design and so a great deal of research has been done on patterns over the last few years. The concept of patterns in this research has its roots in architectural and more widely the use of patterns has received global recognition in software engineering. By now patterns have applications in many diverse disciplines e.g. pedagogy, telecommunication and enterprise development. One of the reasons for their wide appeal is the benefits of “reusability”. In general, patterns are structured documents written by experts to provide tested and proven solutions to commonly occurring problems in a particular context. The power of such documentation is that knowledge and experience is not confined only in the heads of experts but is captured in a way that can be easily accessible and shared. This chapter will present a detailed account on patterns and is organised as follows: Section 6.2 will present an account on the historical background of patterns. The adoption of patterns into diverse disciplines is discussed in Section 6.3 whereas a discussion on pattern applications in embedded software development is presented in Section 6.4. Section 6.5 describes broader aspects of patterns with pattern goals and their limitations with a brief overview of anti-patterns and Section 6.6 is about different forms used to document patterns. Section 6.7 presents an account of pattern languages and Section 6.8 presents the details on the pattern mining process. Section 6.9 discusses perceived inadequacies of patterns with a final section on the conclusions of this chapter in Section 6.10.

6.2. Design patterns in architecture

The concept of abstracting general patterns from a field or discipline in which there is a wide variety of final, apparently differentiated, designs or artefacts, emerged from the work of the Austrian born architect Christopher Alexander. He introduced the concept of patterns during the 1960s and 1970s, when he was a Professor of Architecture at the University of California, Berkley. After obtaining a Bachelor's degree in architecture and a Master's degree in mathematics from Cambridge University, he moved to the United States where he obtained a PhD in architecture from Harvard University. His doctorate thesis, 'Notes on the Synthesis of Form' was published as a book in 1964 (Alexander, 1964).

Alexander and his colleagues (well-known architects, Sarah Ishikawa and Murray Silverstein) published three pioneering texts between 1975 and 1979 (Alexander, Silverstein *et al.*, 1975; Alexander, Ishikawa *et al.*, 1977; Alexander, 1979) that laid the foundation of the use of patterns in the field of architecture. They produced a "pattern language" (Alexander, Ishikawa *et al.*, 1977) to encapsulate practical solutions for designing and building at any scale. The pattern language identified common problems of civil and architectural design, from how cities should be laid out to the location of windows and doors in a room. The aim was to improve the methodology of architecture and urban planning. Additionally he aimed to conserve the knowledge and experience of architects into a collection of ‘patterns’ that Alexander believed could “provide a complete working alternative to present ideas about architecture, building and planning” (Alexander, 1979). In pattern language, Alexander and his colleagues proposed 250 innovative and coherent design patterns for designing and building homes, towns and cities etc. The various patterns in the pattern language can be combined in different ways to build a ‘customised’ solution.

To demonstrate that his proposed patterns works in the real world Alexander outlined a collection of patterns to govern the architecture of a farmhouse (Alexander, Ishikawa *et al.*, 1977). The names of some of the patterns are:

- NORTH SOUTH AXIS
- TWO FLOORS
- WEST FACING ENTRANCE
- BEDROOMS IN FRONT
- GARDEN TO THE SOUTH
- BALCONY TOWARDS THE GARDEN

As one reads through the listed patterns, a visual picture begins to develop in the mind’s eye, creating an image of the farmhouse and the site on which it will rest from nothing more than a written or spoken list (Cloutier and Verma, 2007).

Over the years, Alexander and his associates have applied their pattern language to the design and build of a number of buildings all over the world but his most notable works include: low cost housing in Mexicali in Mexico, University of Oregon in USA, Julian Street Inn (a homeless shelter) in San Jose, California, Eishin School Campus near Tokyo and The West Dean Visitors Centre in Sussex, England.

6.3. Design patterns beyond architecture

It is interesting to observe that over the last several years, Alexander's idea for architecture and designs have had far more impact in fields other than architecture. This includes a diversity of fields from organisational management to poetry, but in particular – and in the context of this research – in the world of computer software design. To quote Richard Gabriel, a prominent advocate of the software pattern approach: *"Chris (Alexander) is a revered cult figure."* (Eakin, 2003).

The adoption of patterns by the software community has been influenced by the need in this community to reuse software. Software developers have a strong tendency to reuse designs that have worked well for them in the past and, as they gain more experience, their repertoire of design experience grows and they become more proficient. However, this design reuse is usually restricted to personal experience and there is usually little sharing of design knowledge among developers (Beck, Coplien *et al.*, 1996). Ganssle (1992) remarks: *"It's ludicrous the way we software people re-invent the wheel* *with every project."* The advent of design patterns offered an opportunity to overcome the inefficiencies and wasted resources of re-invention and to share the collective experience of the software community.

The actual use of patterns in the field of software can be traced back to Kent Beck and Ward Cunningham. In 1987, they introduced a small pattern language (Cunningham, 1987) comprising of five patterns aimed at helping new programmers to design windows-based GUI applications using the Smalltalk programming language. In 1991, Jim Coplien published a catalogue of C++ idioms[10] as a book, "Advanced C++ programming styles and idioms". Later, in 1995 Erich Gamma and his colleagues Richard Helm, Ralph Johnson and John Vlissides now well known in this field as 'The Gang of Four (GoF)' published a set of general-purpose reusable object-oriented design patterns as a book (Gamma, Helm *et al.*, 1995). This is considered the most influential book on software design patterns published to date.

Even though patterns were initially applied mainly in object-oriented software design, they have now been applied in a number of other software engineering fields. Organisational patterns stem from studying recurring structures of relationships within organisations which contribute towards their success. Examples include the pattern language by Cain, Coplien and Harrison (Cain, Coplien *et al.*, 1996) who documented 'best practices' for productive software development, and the collection of patterns for introducing new ideas into an

[10] The authors of the book 'Patter-Oriented Software Architecture (POSA)' (Buschmann, Meunier *et al.*, 1996) have classified patterns into three levels. Idioms are at the lowest level and discuss implementation of certain aspects of the components of a software system using features of a programming language.

organisation by Manns and Rising (Manns and Rising, 2004). Pedagogical patterns capture expert knowledge in the field of teaching and learning and seek to foster best practices in teaching. Some examples of published patterns in pedagogy are (Bergin, 2000) and (Fricke and Volter, 2000). Patterns for telecommunication systems focus on improving the two unique characteristics of software-reliability and human factors. Examples include the works of Adams *et al.* (1996) , Rising (2001) and Hanmer (2007) that covers patterns and pattern languages for use in areas such as telecommunications, distributed systems, middleware etc.

Patterns have also been successfully applied in interaction designs (Borchers, 1999), the software development process (Ambler, 1998), cognition (Gardner, Rush *et al.*, 1998) and software configuration management (Berczuk and Appleton, 2003).

6.4. Patterns and embedded software development

Embedded software development is more challenging compared with desktop applications because they are characterised by resource constraints such as limited memory, limited power consumption and timing constraints. Furthermore, unlike most desktop applications, embedded applications run on specific hardware with special purpose RTOS (real-time operating system), schedulers, programming languages or network protocols such as CAN etc. Another major difference is in the cross development environment. Desktop applications are usually developed on the same platform for which they are designed for, whereas embedded applications are built and tested in simulated environments and the generated executable are then transferred onto the target processor.

As patterns have the ability to capture domain specific information for the benefit of practitioners, they can play a vital role in reducing the complexities involved in embedded software development. A summary of some of the previously introduced pattern collections for embedded software development is as follows:

- A pattern language for designing simple embedded applications was introduced by Mark Bottomley (Bottomley, 1999) and is based on a framework named '*The Carousel'*. The basis of this framework is the famous super loop architecture which can allow designers to design simple applications without the use of any complex control software or operating system to run the system tasks.
- A huge collection of patterns for designing time-triggered embedded systems (called 'PTTES' collection) is developed by Michael Pont (Pont, 2001). This language is intended to support the development of reliable embedded systems and the particular focus of the collection is on systems with time-triggered architectures. Work began on these patterns in 1996 (Pont, Li *et al.*, 1998), and they have since been used in a range of industrial systems (TTE, 2012) numerous university research projects (Kurian and Pont, 2005; Short and Pont, 2005; Phatrapornnant and Pont, 2006; Bautista-Quintero and Pont, 2008; Hughes and Pont, 2008; Gendy and Pont, 2008a) as well as in

undergraduate and postgraduate teaching on many courses offered at The University of Leicester.

- A system of patterns for reliable communication in hard real-time systems called “Triple-T (Time-Triggered Transmission)” is proposed by Wolfgang Herzner and colleagues (Herzner, Kubinger *et al.*, 2005). This pattern collection is focusing on reliable communication with guaranteed transmission times for hard real-time systems. Triple-T is a system of five patterns, which together establish a base for the development of distributed safety-critical real-time systems.
- A pattern language for distributed machine control system by Eloranta and colleagues (Eloranta, Koski *et al.*, 2010) emerged when they found some architectural patterns during their visits to four sites in the Finnish machine industry to find design patterns to this domain. This pattern language included patterns for messaging, fault tolerance, redundancy and system configuration.

Other well-known examples of pattern languages in various domains of interest to control engineers are: patterns for concurrent and networked objects (Schmidt, Stal *et al.*, 2000), communication patterns (Rising, 2001) and patterns for fault tolerant software by Bob Hanmer (Hanmer, 2007).

6.5. Broader aspects of patterns

Alexander’s idea of capturing design experience through patterns has been widely accepted and built upon by the research community especially in the field of software engineering. Different experts provide their own views about patterns and some of these are summarized below.

According to Richard Gabriel, patterns are a means to capture common sense and abstractions that are not easily captured otherwise (Gabriel, 1996). Brad Appleton has a different point of view about patterns, for him, a pattern is a named nugget of instructive information that captures the essential structure and insight of a successful family of proven solutions to a recurring problem that arises within a certain context and system of forces (Appleton, 2000). Authors of the most famous collection of design patterns for object-oriented software design called GoF consider patterns as a way of documenting and sharing 'best practices', solutions that have successfully worked for experienced designers (Gamma, Helm *et al.*, 1995). To Linda Rising, patterns are artefacts that have been discovered in more than one existing system (Rising, 1998).

Patterns emerge from lessons learned in the practice of a particular discipline. Domain experts accumulate these lessons and season them with knowledge earned through a study of the domain's theoretical base. These experts are then able to shape and re-shape patterns which can be re-used in the domain. Together these activities constitute the development of a pattern (Petter, Khazanchi *et al.*, 2010).

The essence of a pattern can be viewed in terms of the existence of a problem and its solution as shown in Figure 6-1.

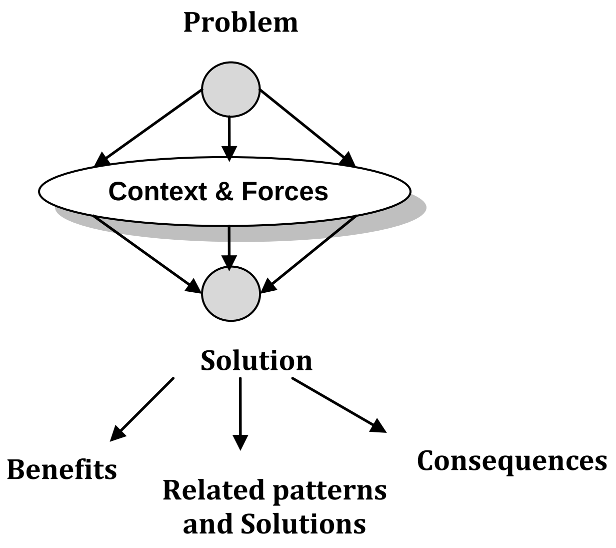


Figure 6-1 Illustrating the concept of a design pattern

The problem is usually elaborated in terms of its context and the applicable design forces[11] which provide a basis for the solution. The role of the solution is to resolve the design forces in such a way that it generates benefits, some consequences and follow on problems which lead to the applicability of other patterns.

It is interesting to compare patterns with other approaches of learning such as algorithms and heuristics. An algorithm provides a method for solving a problem using operations from a given set of basic operations (addition, subtraction, multiplication and division), which produces the answer in a finite number of such operations (Gear, 1973). This definition implies that an algorithm *converges*, that is it always reaches the answer (i.e. the optimal solution) in a finite number of steps. For the purpose of deriving a solution an algorithm may be applied to a set of mathematical relationships or

[11] A force provides a concrete scenario which serves as motivation for the pattern (Appleton, 2000).

mathematical statements that relates to the various components of a system. These relationships are basically the representation of knowledge of how a particular system works while design patterns do not have any such restrictions.

Heuristic rules are those that are developed through intuition, experience and judgment. Heuristics are a representation of guidelines through which a system may be operated and unlike relationships they do not represent the knowledge of the design. Also, heuristics do not necessarily give the best or optimal solution (unlike algorithms). Heuristic rules evolve over years of experience but unlike design patterns they are usually far more private and personal and not available in the public domain.

It is important to note that patterns are not intended to degrade the design individuality but rather support it. It is because a pattern provides a generic solution for a recurring problem: a solution that can be implemented in many ways without necessarily being twice the same (Cool, 1998). The process of adapting or applying the pattern enables customisation at different stages during the software development so it's not a case of 'One size fits all' with patterns.

Software practitioners were quick to learn from the pedagogical interest of patterns i.e. 'to learn from experience'. It is because codifying good design practice helps to distil and to disseminate experience, thereby helping others avoid frequently encountered development traps and pitfalls (Jezequel, Train *et al.*, 2000) .

6.5.1. Limitations of patterns

Patterns do not have the capability for addressing all re-use issues, nor will they single-handedly solve all the crises encountered during software development. Patterns do not also turn novices into instant expert designers, and the following are quotes on patterns and how they should be viewed by some of the experts in this field:

- Patterns are not recipes which say “Do this, and everything will be fine!” (Fricke and Volter, 2000).
- A pattern is not a programming language construct or idiom (Richard Gabriel).
- Patterns will not eliminate the need for intelligence and taste (Paul Chisholm in (Hanmer, 2009)).
- A pattern is not a silver bullet (Rising, 1999).

6.5.2. Anti-patterns

The term anti-pattern was coined by Andrew Koening in 1995. As the name implies, anti-patterns are negative forms of patterns. If patterns are described as best solutions to recurring problems, anti-patterns are bad solutions to the same recurring problems. Koening claimed that anti-patterns may well be more valuable than ‘real’ patterns simply because knowing what doesn’t work (and why) can be incredibly useful (Rising, 1998). Another view of an anti-pattern is that it “describes how to get out of a bad solution, and then how to proceed from there to a good solution” (Appleton, 2000). Jim Coplien explains further: *“Anti-patterns don’t provide a resolution force as patterns do, and they are dangerous as teaching tools: good pedagogy builds on positive examples that students can remember, rather than negative examples. Anti-patterns might be good diagnostic tools to understand system problems”* (Coplien, 2000).

6.6. Pattern forms

Patterns are structured documents, so their layout and constituent components are important in the sense that they are the means to provide information in a way which is easier to grasp and understand by its user. Different authors have their own ways of documenting patterns. However, certain pattern forms have become more established than others. Some well-known pattern forms are presented in Table 6-1.

Table 6-1 Pattern forms

Pattern form	Description
Alexandrian form	This layout is used by Alexander in (Alexander, Ishikawa *et al.*, 1977).
GoF form	Layout used to write the famous Gang of Four patterns (Gamma, Helm *et al.*, 1995).
Coplien form	This layout is followed by James Coplien.
POSA form	This layout is used to write Patterns of Software Architecture (Buschmann, Meunier *et al.*, 1996).
PTTES form	This layout is used by Michael Pont for writing Patterns for Time-Triggered Embedded Systems (PTTES) collection.

Please note that the complete templates for each of these pattern forms are presented in Appendix A.

6.6.1. Elements of a pattern

Even though there are different pattern forms, they all share certain key elements which are listed as follows:

- **Name:** Patterns names are intended to concisely capture the idea behind the problem and solution being addressed by the pattern. For example the pattern HEART BEAT LED is a pattern for checking whether a system is active.
- **Problem-Solution pair:** This constitutes the core of the pattern. A successful pattern is one which conveys the solution effectively and which others can reuse in their designs.
- **Context:** This describes the settings in which the problem is found. This part should answer the question, "When can I apply this pattern?"
- **Forces:** Forces define the problem (Harrison, 2006). A strong forces section in a pattern will enable the reader to judge whether the solution is good and whether it fits the problem statement.
- **Examples:** One or more sample applications of the pattern, supplemented by implementation.
- **Resulting context:** No pattern is perfect. Every pattern has some shortcomings. This section describes the effects of applying the pattern.
- **Related patterns:** This section mentions other patterns that solve similar problems. These may be predecessor patterns whose application leads to this pattern, successor patterns whose application follows this pattern, alternative patterns that describe a different solution to the same problem but under different forces and constraints (Appleton, 2000).

6.7. Pattern languages

Though reflecting different views about patterns, all researchers and experts share a common opinion that patterns should not exist in isolation, they should ideally form a part of a pattern 'collection'. *"No pattern is an island"* (Buschmann, Meunier *et al.*, 1996). In the words of Alexander *et al.* (1977): *"In short, no pattern is an isolated entity. Each pattern can exist in the world, only to the extent that is supported by other patterns: the larger patterns in which it is embedded, the patterns of the same size that surround it, and the smaller patterns which are embedded in it."*

A pattern language is a set of inter-related patterns (see Figure 6-2), where one can use the individual patterns to solve small problems or one can use the language as a whole to solve a much bigger problem. The collection of patterns comprising the 'language' forms a kind of 'vocabulary' for understanding and communicating ideas.

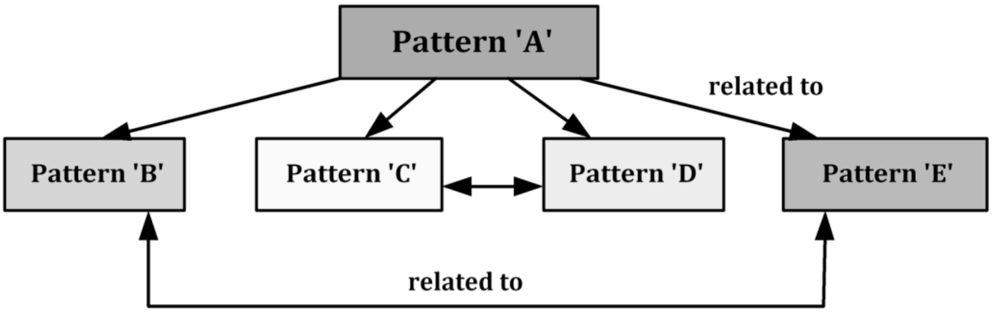


Figure 6-2 Illustrating an example structure of a pattern language

Just as the relationships between words – through meaning and grammar – form a useful 'language', similarly, organising patterns in a structure so that they have logical relationships with each other leads to the formation of a pattern language. Organising patterns in a pattern language gives the designers a ‘map’ to complete the structure they are designing or building. In a pattern language, the patterns are organised such that they guide the reader from large-scale patterns to smaller-scale patterns. The smaller patterns help to complete the larger patterns. The pattern language has the structure of a network that includes various rules and guidelines to explain when and how to apply the constituent patterns to solve a problem that is too large for an individual pattern to solve. According to Bob Hanmer “*Patterns generally adhere to the model of small nuggets of information that work together rather than trying to ‘Resolve the world’ in one pattern, so the solution of one pattern flows into the problem of the next”* (Hanmer, 2007)*.*

6.7.1. Completeness of a pattern language

A pattern language is a group of patterns that completely covers a problem space. A language can be complete in two ways: Functionally and Morphologically (Hanmer, 2003). A pattern language is functionally complete meaning that when a pattern in the language introduces some new force that is un-resolved there is some other pattern within the language that resolves the force. The new force must be resolved within the language rather than some stray pattern that is outside the language. Morphological completeness means that the patterns in the language fit together to form a complete structure without any gaps.

6.7.2. Pattern languages versus text books

Pattern documentation is usually in the form of books that have been specifically written to address a particular domain. For example Alexander et al (1977) for building architecture, Gamma, Helm *et al* (1995) for object-oriented software development, Pont (2001) for designing time-triggered embedded applications and Hanmer (2007) for fault tolerant software design. It is instructive in terms of assessing pattern books to compare them with conventional domain specific text books. Regular text books actually complement the pattern books as they provide a theoretical basis to understand the domain. When the reader is able to implement a basic system then he/she may turn to a pattern book to learn about some proven solutions to the common problems that they have to solve. A reader would not get much theory in a pattern book but only the proven solutions to commonly occurring problems. Conversely a text book does not provide details for how to solve the problems or work through the details.

6.8. Pattern mining and refinement

It is interesting to ask how pattern authors discover a pattern and how they document it so that they can effectively convey their design ideas. There is therefore, a need to look for or discover patterns before documentation. Pattern mining is the process of discovering new patterns prior to documentation. Brad Appleton calls this process reverse-architecting (Appleton, 2000). David DeLano (1998) talks about several metaphors that were proposed such as hunting, fishing, harvesting, paleontology or archaeology and mining to describe the process of discovering and documenting patterns (DeLano, 1998). Fishing, hunting and harvesting 'almost' describe the process of pattern discovery and documentation, but they nevertheless fall short. Fishing and hunting implies too much randomness while harvesting is discarded as patterns are not grown or created but are present in the artefacts that already exist. Paleontology or archaeology no doubt provides a reasonably correct description of the process if patterns are considered as fossils or buried relics. The archaeologists or paleontologists then have to dig through all the mass separating the 'good' from the 'bad', and once discovered, the relics have to be carefully cleaned. Finally, once all the 'pieces' have been retrieved and cleaned; they are re-assembled for 'public viewing'. The primary objections that could be raised to the use of these metaphors are that patterns are meant for everyday use; however, the discoveries of palaeontologists or archaeologists are generally displayed in a museum.

Consequently, the pattern community decided to settle for the mining metaphor when describing the pattern discovery process. DeLano in (Rising, 1998) remarks: *"Mining engineers tend to know where to excavate for the minerals they seek. What they find is not always of high quality, just as patterns vary in usefulness. The mined elements do not need to be removed as gingerly as a fossil or artefact. The elements must be further processed before they become useful. After refinement – cutting, polishing, smelting, moulding – we are left with a useful product. Often the result is one of lasting* *beauty or resilience. As for vocabulary, we are pattern miners participating in pattern mining"*.

Numerous international conferences on the Pattern Languages of Programming 'PLoP' such as EuroPLoP (Europe) , ChilliPLoP (USA) , KoalaPLoP (Australia), SugarLoafPLoP (Brazil), MensorePLoP (Japan) are organised by the patterns community every year as a forum to discuss the latest patterns and pattern languages. As part of the refinement process, pattern languages and individual patterns are critically reviewed by experts at PLoP events. This process is called shepherding. The shepherding process begins when a paper is initially submitted to a PLoP conference. The author improves the paper (generally following the advice of the reviewer called the 'shepherd') and sends the corrected version back to the shepherd. This process of revision between the shepherd and the sheep (the author) is repeated three or four times. The review process is more intensive during the conference. These reviews take place at a *Writer's Workshop* introduced by Richard Gabriel at the first PLoP conference in 1994. In a *Writer's Workshop* a group of people periodically get together and read and critique manuscripts by fellow workshop participants. This feedback allows the participants to improve their patterns and make them more publishable. Thus, from the commencement of an idea that is conceived in the mind a pattern undergoes a rigorous amelioration process that seeks to refine it to a standard of quality that makes it understandable and acceptable by the peer community.

6.9. Inadequacies of patterns

In common with other technologies patterns have certain limitations. The most important question that has been raised by critics about the effectiveness of patterns is related to their standardization. Patterns have no formal standards and different authors write patterns differently. They are semi-formal descriptions that describe a problem and its solution. Depending upon their effectiveness patterns can become classics, remains limited to specialist areas, or just elapsed and forgotten. The success of a pattern is heavily dependent on the name of the pattern and the nature of the information it contained (Schmidt, 1995). Patterns which are too long and flooded with information may lose the focus and core of the solution. On the other hand, keeping a pattern too short might compromise the quality of information provided, and will force the use of other related information sources to understand the design problem (Agerbo and Cornils, 1998; Vokac, 2004).

In projects which involve collaborative team work, for effective use of the patterns it is important to provide an introduction and training to the whole team on the pattern collection of interest (Unger and Tichy, 2000). This is important when patterns are used to enhance the vocabulary of practitioners. Technically intensive communications needs to be supported but it may cost extra time in the training of team members.

Another argument that goes against patterns concerns programming language dependency. Most pattern collections tend to be dependent on the language used such as C, C++ or Java. This dependency sometimes restricts the use of the pattern to certain platforms only. Finally it might not be appropriate to use the pattern in every situation specifically if the solution is obvious. Some critics observe that design patterns are over-hyped (Cline, 1996). To compare patterns with simple solutions, Lutz Prechelt and colleagues have conducted an experiment and concluded that using design patterns can be useful, neutral or harmful depending on the circumstance of use (Prechelt, Unger *et al.*, 2001). Some critics have undermined patterns, for example according to Vokac, "*A pattern is not a rigorous recipe (unlike some modeling standards such as UML) to be followed but it is more than just a loose suggestion*"(Vokac, 2004). This is however difficult to accept, as UML does not provide recipe like instructions but patterns do.

Despite the disadvantages mentioned above, the fact remains that for increasingly complex design problems patterns can provide support by offering a way to reuse proven and tested solutions and thus help to "avoid re-inventing the wheel". The interest in patterns have grown over last few years as evidenced by the number of conferences and meetings held every year all over the world since 1994. Expanding from United States and Europe the recent development of such events are AsianPLoP 2011 in Tokyo and GuruPLoP 2013 in India. It is relatively early days for this novel approach and there has been sufficient successful application of patterns recently in various fields such as formalization techniques (Taibi, 2007), web application design (Millett, 2010) and mobile application design (Neil, 2012), etc. It is therefore reasonable to assume they will grow in importance in the coming years.

6.10. Conclusions

This chapter has aimed to describe the essential concepts related to design patterns. It started with a discussion on the historical background with a brief account of the aims and objectives behind the idea of introducing quality in patterns and some discussion on pattern goals and boundaries. The chapter continued by discussing the adoption and appreciation of patterns in diverse disciplines which are completely different from the field in which pattern concept originated. CHAPTER 6 also presents a brief account of different pattern forms in use, the concept of pattern language and the process of discovering, documenting and refining patterns.

Realising the strength of this established methodology of patterns, it concluded that patterns have the capability to offer a good framework for capturing and sharing practice related to the complex process of the migration of architecture for embedded applications discussed in Chapter 5. Subsequent chapters in the next section will explain how this research has explored and utilised the capacity of patterns for migration between software architectures used in embedded applications.

PART C: DEVELOPMENT OF PATTERNS FOR MIGRATION

This part of the thesis presents the main body of the work carried out during the course of the research period.

The main topics are as follows:

1. Derivation of new design patterns (CHAPTER 7).
2. An overview of the newly documented patterns and formation of a pattern language (CHAPTER 8).
3. Demonstrations of design patterns applied to real design problems (CHAPTER 9).
4. Evaluation of the efficacy of the pattern language through controlled experiments (CHAPTER 10).
5. Evaluation of the patterns in the industrial context (CHAPTER 11).

CHAPTER 7. DERIVATION OF “PATTERNS FOR MIGRATION”

7.1. Introduction

The focus of this chapter is on explicating the process of pattern mining and the derivation of the patterns in the newly proposed pattern language called PMES (Patterns for Migration of Embedded Systems) developed during the research. In the present state there are 23 patterns in the pattern language out of which 13 are completely new patterns that were documented during the research. These patterns were developed and evolved in stages throughout the research. This chapter describes the driving forces behind the creation of new patterns and their associations with existing patterns in the current context. The chapter begins with Section 7.2 which gives the rationale and goals for the patterns to support the migration process. The chapter then proceeds with a discussion on how the patterns were derived. In this regard Section 7.3 to Section 7.11 presents the bases for proposing each of the patterns, and the experimental evidence in some cases demonstrating the usefulness of the pattern. Chapter conclusions are presented in Section 7.12.

7.2. Rationale for patterns to support migration

CHAPTER 2 in this thesis presented the three example applications recorded in the literature that have passed through the process of migration from ET to TT design. The chapter concluded that there is a lack of support in the standard approach of changing the underlying software architecture which could guide the developers in the field. CHAPTER 6 discusses the appealing features of patterns that have led experts in diverse disciplines to introduce patterns in their own field of expertise. In conjunction with this Section 6.4 mentions various examples of pattern collections that were proposed for embedded software development. All of the examples cited in the literature such as (Schmidt, 1995; Bottomley, 1999; Pont, 2001; Rising, 2001; Herzner, Kubinger *et al.*, 2005; Hanmer, 2007; Eloranta, Koski *et al.*, 2010) indicate that in the field of embedded systems, most of the previous research work are focused on documenting patterns for system construction. However, design patterns which could assist embedded system practitioners in the process of migration of architecture has been somewhat neglected. The research presented in this thesis aimed to address this lack of support by introducing a collection of techniques bundled and documented in the form of a pattern collection.

Collectively the underlying goals of the pattern collection are:

- To provide sufficient understanding of the challenges involved in migrating from an event-triggered to a time-triggered design, and to help in decision-making on which of the two designs is appropriate in a particular situation.
- To help the practitioners in choosing the appropriate time-triggered architecture for migration.
- To assist in handling the problems in the achieved time-triggered design such as long task problems in co-operative design and shared resources access in pre-emptive designs.
- To provide techniques for further optimising a completed time-triggered design.

7.2.1. Patterns for new designs versus patterns for migration

It was noted in Section 7.2 that most of the earlier research studies on design patterns in embedded systems are related to system *construction* rather than system *alteration*.

For reference, a comparison between the previously-developed “PTTES” collection (Pont, 2001) and the PMES collection developed in this thesis is given in Table 7-1.

Table 7-1 Comparison between the 'PTTES’ collection and the ‘PMES' collection

PTTES Collection	PMES Collection
PTTES is intended to support time-triggered system designs from scratch.	PMES is intended to support migration of existing systems from E-T to T-T designs.
PTTES describes how to develop new systems using a single interrupt.	PMES describes how to convert existing “multiple interrupt” designs into “single interrupt” systems.
PTTES has addressed system design for single-processor as well as multi-processor designs from scratch.	PMES collection has only addressed the conversion of single-processor designs at this stage.

Please note that the PMES collection has borrowed some patterns from the PTTES collection such as CO-OPERATIVE SCHEDULER, HYBRID SCHEDULER, LOOP TIMEOUT and WATCHDOG. In addition, the PMES collection has also introduced some new patterns such as CHOOSING TASK PARAMETERS, BUFFERED OUTPUT, POLLED INPUT, BALANCED SYSTEM, SINGLE PATH DELAY, TAKE A NAP, SYSTEM MONITORS and TASK GUARDIAN: some of these patterns may also be applicable with the PTTES collection.

7.3. Choosing the appropriate architecture

As mentioned in Chapter 5 migrating embedded applications involved taking decisions at various stages which impact on decisions further downstream in the design. These decisions need to be taken with due consideration because a wrong choice at any stage can cascade further problems later in the design, indeed the fore-most design decision is choosing an alternative architecture against the existing architecture of the application. Pertinent to the focus of this research 'Migration from event-triggered to time-triggered architecture', it was realised that it is vital for the user to know if the time-triggered paradigm is an appropriate choice for their application. If it is, this research has provided grounds for a pattern (TIME FOR TT) which could guide the user about a range of applications for which a TT design can be proven as an appropriate choice. The underlying aim with this pattern was to help the users to gain confidence in their choice of moving to a time-triggered design against an existing event-triggered design. The literature was explored extensively before documenting the pattern to list the general category of embedded applications for which either an event-triggered design or a time-triggered design can be proven as the right choice.

7.4. Transforming an ET design to a TT design

The evolution of the range of design patterns was based on an understanding, informed by real examples, of the sequence of steps a developer would need to follow once he/she had decided to switch to a TT design. After this first step it is important to think about the major architectural changes that are required to effect the overall migration. The emergence of the idea of the pattern EVENTS TO TIME is based on a need for architectural changes required for transforming an interrupt-driven (with multiple interrupts active in a system) to a time-driven (with single interrupt) design. The motivating example of the F-18 aircraft discussed in Section 2.2 mentioned the transformation of the pre-emptive design to a pre-runtime design involving the conversion of interrupt service routines to processes; however it does not mention how the transformation was achieved. It therefore became necessary to explore techniques which could be easily adapted to solve similar problems. In the initial stages of the research the pattern solution was restricted to the provision of a solution of transforming to a co-operative design only as discussed in EVENTS TO TIME (TTC) (Lakhani, Das *et al.*, 2009a; Lakhani and Pont, 2010a). The derivation of the pattern involved analysing various interrupt-driven designs discussed in the literature (Shepard and Gagne, 1990; Pont, 2002; Sloss, Symes *et al.*, 2004; Obermaisser, 2005) and through discussion within the research group among peers by viewing blogs and discussion forums where developers describe and exchange their designs. A generic prototype for both ET and a TT design is then developed and the steps involved in transformation are documented. In the later stages of the research the pattern is made more generic for a variety of TT designs as discussed in (Lakhani, Wang *et al.*, 2011).

7.5. Choosing the appropriate TT architecture

As discussed in Section 4.3 that there are various choices in the design of an application with a TT architecture. It is important therefore for a designer to be aware of the various options available in a TT design, and the details associated with each design so that they can think about various trade-offs involved in making the appropriate choice for their application. The pattern TT Scheduler is derived with this consideration in mind and documentation is provided on the overview and working mechanisms of co-operative and pre-emptive designs, and the strengths and weaknesses associated with each of the designs. This documentation also includes a link to previously documented patterns in (Pont, 2001) as the complete implementation details for Co-Operative Scheduler and Hybrid Scheduler are provided there and the implementation details for the Pre-emptive Scheduler as discussed in (Maaita, 2008). If the users wish they can follow the same implementations as described in Pont (2001) or they can implement their own design with the working mechanisms discussed in TT Scheduler.

Following the decision to move from an ET design to a TT design there are three patterns (as shown in Figure 7-1) provided that discuss all the finer details that a pattern user needs to know when moving from an ET design to an appropriate TT design which could best match their candidate application.

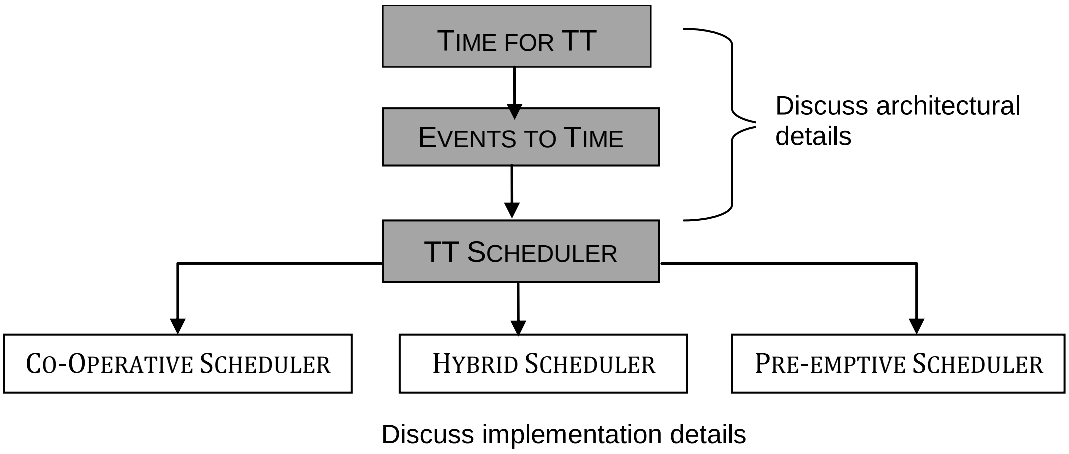


Figure 7-1 Illustrating the sequence of following patterns from the initial decision of migration to choosing an appropriate TT design

7.6. Handling events in TT designs

In the transformed design after migration the basic requirement is to obtain a replica for the handling of events in a TT design such that the same functionality of the system can be achieved with the single interrupt active in the system. For example in the motivating example of the Sony cell processor discussed in Section 2.4 the major problem encountered was frequent interrupts in the system at high data rates, and so a software polling mechanism was chosen to overcome the high overheads in the system. It was therefore apparent that a technique which could describe how this conversion can be achieved would significantly help in handling all internal and external inputs in the system. This realisation led towards the derivation of the pattern POLLED INPUT. As part of the pattern mining process which is required for documenting a new pattern code listings provided in various books for example (Ganssle, 1992; Barr, 1999; Pont, 2002) identified that explained the use of a switch interface in the design. It was realised that most of the experts use the same technique of designing a task that keeps track of the arrival of inputs on a GPIO and describe this as a polling mechanism. However, the periodicity of this task is chosen depending upon the application requirements. It was observed that setting a period which is at least the equivalent of minimum inter-arrival time of the task would make sense for any application, and so the pattern is documented with information gained during the pattern mining process. For the pattern POLLED INPUT a pattern implementation example (PIE) with the name of SWITCH INTERFACE is also provided for the 8051 family of microcontrollers and is described in (Lakhani, Das *et al.*, 2009a). This was introduced to provide platform specific implementation details of the pattern. PIEs are further explained in Section 8.2 in this thesis.

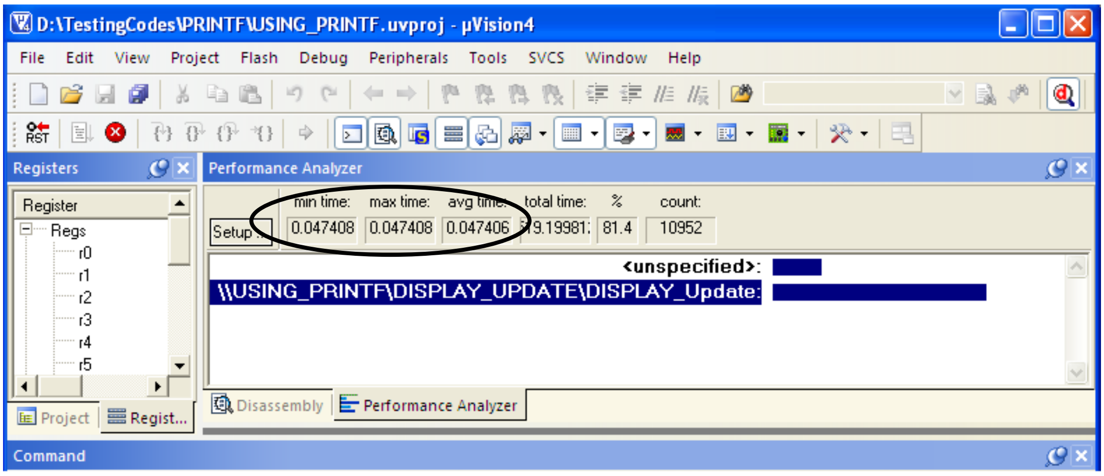
7.7. Handling problems with the co-operative design

Co-operative designs as discussed in CHAPTER 4 are based on cyclic executive model also known as timeline scheduler. One of the drawbacks of co-operative designs is that any long task in the system could incur long time delays as the task pre-emption is not allowed, and the waiting task will be delayed until the long task relinquishes the system resources. Many embedded applications are required to display messages on the user interface for example an LCD screen that displays the change in temperature in say a temperature monitoring application. Another way for displaying messages ay is in the use of the printf() function. Though simple in use printf() calls a library of functions at its back-end and takes approximately 1ms per character of execution time which can be problematic. This is because the display task can take as long as the length of the message (number of characters), for example a display message of 40 characters can take 40ms of execution time which could lead to unnecessary delays in the co-operative based system designs. Instead of discouraging the use of co-operative designs it was desirable to devise a method which could replace the printf() function.

As most of the microcontrollers are provided with a UART (Universal Asynchronous Receiver Transmitter) which can transfer data using the RS-232 protocol it was found useful to make use of a UART buffer to hold data and transfer it in bulk once.

A small testing application was created to compare this technique with the printf() function and a considerable time difference was observed. For example for a simple display task to print a string "Migration from event-triggered to time-triggered" was designed using the use of a buffer technique (details described in CHAPTER 8) and the use of printf() function. The execution times of both the tasks were measured using the Performance Analyser in the Keil µvision simulator for the 8051 microcontroller. The use of the buffer technique showed a reduction of 94% of execution time from 47ms to 3ms.

The results are shown in Figure 7-2 and Figure 7-3.

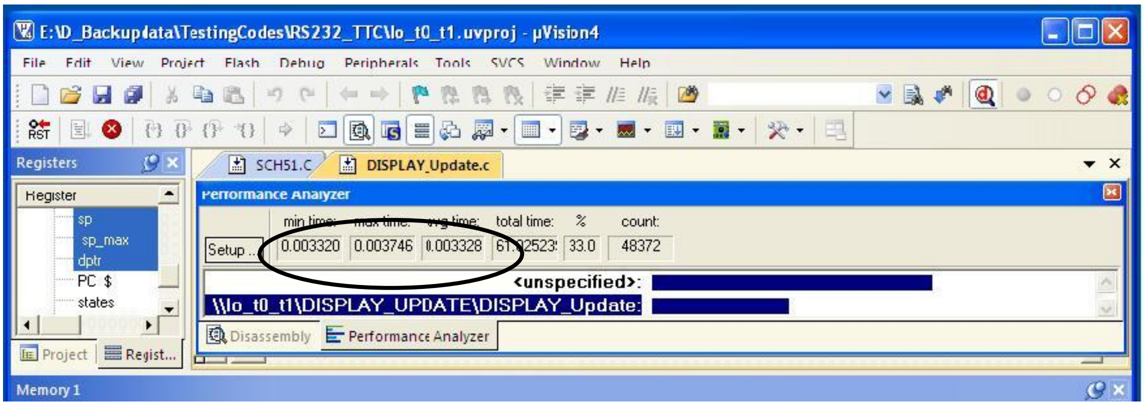

D:\TestingCodes\PRINTF\USING_PRINTF.uvproj - µVision4
File Edit View Project Flash Debug Peripherals Tools SVCS Window Help
Registers
Register
Regs
r0
r1
r2
r3
r4
r5
Project
Regist...
Performance Analyzer
Setup ...
Disassembly
Performance Analyzer
Command

min time:	max time:	avg time:	total time:	%	count:
0.047408	0.047408	0.047406	19.19981:	81.4	10952

<unspecified>:

\\USING_PRINTF\DISPLAY_UPDATE\DISPLAY_Update:

Figure 7-2 Performance Analyzer in the Keil simulator showing the execution time of the Display_Update() task = 47 ms with printf() function



min time:	max time:	avg time:	total time:	%	count:
0.003320	0.003746	0.003328	61.02523:	33.0	48372

<unspecified>:

\\lo_t0_t1\DISPLAY_UPDATE\DISPLAY_Update:

Figure 7-3 Performance Analyzer in the Keil simulator showing the execution time of the Display_Update task = 3ms using Buffered output technique

Other tests with different applications were performed where the number of characters on display may change depending on user input. In that case the buffered technique was found even more useful as it kept the minimum and maximum times nearly same in contrast to printf() which showed high variations depending upon the length of characters in the display message. Since the display task is a commonly required task for a huge range of embedded applications the results provides strong grounds for the pattern

BUFFERED OUTPUT to resolve the long task problem associated with co-operative designs.

7.8. Handling problems with the pre-emptive designs

In moving from ET design, to TT design if the co-operative design does not appear to be a wise choice the developers have two options for selecting a pre-emptive design. These are: a hybrid design with the option of having only a single pre-emptive task in the system, or a fully pre-emptive design. The main problem with the pre-emptive designs is in the handling of shared resources as discussed in Section 4.4. Patterns for handling shared resource access have been explored previously in the Embedded Systems Research Group (ESRG) and published in (Wang, Pont *et al.*, 2007) . Because of their applicability in the current context it was found useful to associate these patterns here in solving the problems that the HYBRID SCHEDULER and the PRE-EMPTIVE SCHEDULER can face. The collection contains one abstract pattern CRITICAL SECTION, and four patterns DISABLE TIMER INTERRUPT, RESOURCE LOCK, PRIORITY CEILING PROTOCOL, and IMPROVED PRIORITY CEILING PROTOCOL.

7.9. Designing tasks for a TT design

Section 4.5 has discussed how TT architectures are more difficult to design than ET designs. This is because choosing task parameters such as ‘offset’ and ‘period’ can affect the overall system performance, for example choosing an incorrect value of offsets may result in missing a deadline and so can compromise the overall system reliability. Similarly selecting the correct task order is also important otherwise un-expected jitter values in the system might be very harmful. This was observed whilst carrying out various experiments (for examples please see the pattern CHOOSING TASK PARAMETERS in Appendix B). It was decided therefore that a pattern which can help in selecting the appropriate task parameters could be useful. The pattern CHOOSING TASK PARAMETERS fulfill this function such that a stable system is achieved with all the tasks meet their deadlines with fixed and lower values of jitter.

7.10. Achieving a 'Balanced' TT design

After a basic framework of patterns was achieved in moving from ET to TT designs (for both the co-operative and the pre-emptive designs) the research focused on making TT designs truly reliable as achieving reliability is the key requirement in the migration process. Whilst carrying out experimental work on several simple embedded applications as part of this research, it was observed that many applications designed using TT architecture showed variations in their task timings. In the example discussed in Section 2.3 it was reported that the TT design was not completely jitter free but has jitter values in the microseconds range compared to milliseconds in ET design. Further investigation and a literature review revealed the fact that these variations may arise due to the hardware features. For example variations in the oscillator frequency (Kirner and Puschner, 2003) or in the software with the use of branch statements (if-else, switch-case etc.) in the code (Puschner and Burns, 2002). These studies resulted in the emergence of the abstract pattern BALANCED SYSTEM which is documented with the aim that developers should be well-informed of all the penalties that may result from the jitter present in the system.

Experimental evidence for jitter variations was obtained using the SANDWICH DELAY technique for the ARM7 platform for two dummy tasks in a system in which an additional timer was used to sandwich a task. Results showed considerable improvement in jitter values for execution times from 8µsec down to 1µsec and for period from 14µsec to 3µsec for task A in the system. This improvement results in precise timings of the period for task B in the system with the reduction of jitter values from 16µsec to 3µsec. A comparison of the results is shown in Table 7-2 and Table 7-3 and these results were also reported in (Lakhani and Pont, 2010b).

Table 7-2 Jitter measurements for the Un-Balanced System

All measurement are in µsec	**Task A**		**Task B**	
	Execution time	**Period**	**Execution time**	**Period**
Minimum	56	99993	25	99992
Maximum	64	100007	26	100007
Difference	8	14	1	16
Average	56	100000	25	100000

Table 7-3 Jitter measurements for the Balanced System

All measurement are in µsec	**Task A**		**Task B**	
	Execution time	**Period**	**Execution time**	**Period**
Minimum	222	99999	25	99999
Maximum	223	100002	26	100000
Difference	1	3	1	3
Average	222	100000	25	100000

7.11. Monitoring the TT design

No design is completely 100% fault-free even if designed with due care and this leaves a possibility of fault occurrence in the system at runtime which could have a dangerous impact on system reliability. Such faults may occur due to hardware problems such as incorrect initialization of the hardware or software, mismanagements such as incorrect initialization of variables, or task overruns in the system. If the possible occurrence of such problems is not considered during the system design it may lead to serious consequences especially in applications such as automotive vehicles or aircrafts and could result loss of invaluable human lives or other assets. It is therefore essential to introduce some monitoring components in such systems to ensure their correct functionality while running. The safety standards introduced by the automotive industry called ISO 26262 for the guaranteed safety of automotive vehicles also emphasises the use of such techniques while designing software. This opened up further requirements in this research for introducing the abstract pattern SYSTEM MONITORS designed to ensure that the system will not ‘hang’ and will keep on functioning despite unfavourable conditions. The relevant patterns found in the PTTES collection are LOOP TIMEOUT and WATCHDOG. A new pattern TASK GUARDIAN was also developed and documented which had the function of avoiding any complications that could happen as a result of task overrun in the system.

7.12. Conclusion

This chapter has described the mining and development process for the new patterns that is one part of this research. The aim in the beginning was to identify problems that occur in changing the underlying architecture from ET design to TT design and what important decisions a designer of the embedded application has to take. To achieve this required a great deal of observational and experimental work that resulted in the development of new patterns and the documentation required to implement these patterns in real systems. It was also realised during the later survey work that the developers' community should also be made aware of the limitations associated with each TT design so that patterns are documented to achieve an optimised TT design after migration.

CHAPTER 8. OVERVIEW OF THE PROPOSED PATTERNS

8.1. Introduction

As briefly discussed in Section 1.2.1 on the goals of the research presented in this thesis, in many embedded applications a change in architecture becomes necessary when initially defined standards are not achieved in practice. CHAPTER 2 introduced examples of real applications such as F-18 MC (Shepard and Gagne, 1990) and Sony cell processor (Turley, 2009) that illustrate this. The process of migration in embedded systems architecture is complex and offers various challenges to the designers and developers of embedded applications. Recognising the lack of availability of a set of guidelines to tackle this situation, the research set out to find ways to meet the challenges of migration in a more robust and coherent way based on the application of design patterns. CHAPTER 6 introduced design patterns and presents the expert's opinion on the potential benefits of using patterns to solve complex design problems. The focus of this chapter is on introducing and explaining the newly proposed pattern collection. Section 8.2 discusses the categories that are used to describe the patterns introduced in the PMES and Section 8.3 has introduced the PMES language. Section 8.4 has briefly introduced each of the patterns individually and conclusions are presented in Section 8.5. Please note that the work described in this chapter have been published in (Lakhani, Das *et al.*, 2009a; Lakhani, Das *et al.*, 2010c; Lakhani, Wang *et al.*, 2011).

8.2. Pattern categories in the PMES collection

The patterns that were derived or associated were categorised as either abstract patterns or concrete patterns. This division of patterns in a pattern language has been introduced previously (Kurian and Pont, 2005; Kurian and Pont 2006) to re-structure the PTTES language. An abstract pattern is meant to identify a class of design problems for which one or more design patterns are available in the collection. Abstract patterns do not directly tell the user how to construct a piece of hardware or software; instead, they assist a developer in deciding whether or not the use of a particular design solution (e.g. a hardware component or a software algorithm) would be an appropriate way of solving a particular design challenge. The information contained in the design pattern outlines the solution that is required to be implemented once the major architectural issues (discussed in the abstract pattern) surrounding the design problem are solved. In other words, by using one of the design patterns, the problem that is tackled by an abstract pattern can be solved. This implies the existence of one-to-many relationship between the abstract pattern and the design pattern as shown in Figure 8-1.

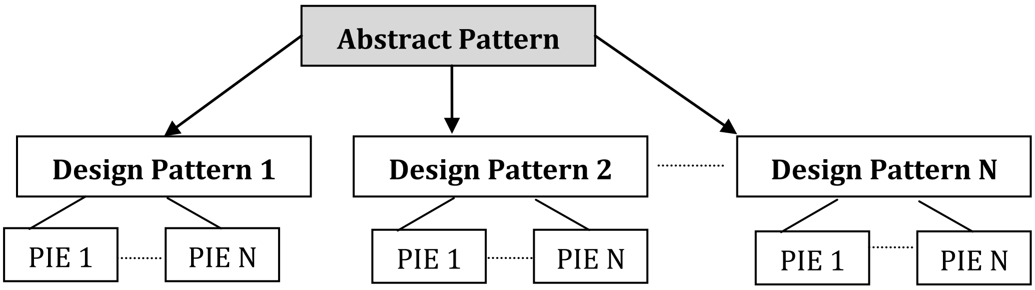


Figure 8-1 One-to-many relationship between abstract pattern, design patterns and PIEs

The third element in the proposed re-structuring is the ‘Pattern Implementation Example’ or PIE. This was introduced to provide platform specific implementation details for a solution presented in a design pattern. This is particularly useful in the field of embedded systems where there are differences in hardware platforms (e.g. 8-bit, 16-bit, 32-bit and 64-bit) and programming languages (e.g. C, C++, Assembly etc) so a single pattern can have multiple PIEs.

8.3. PMES – towards a pattern language for migration

This research presents a new pattern language called PMES (Patterns for Migration of Embedded Systems). A full specification for each individual pattern is included in Appendix B in this thesis. The pattern language is still in the development stage and is not fully complete. In Section 6.7.1 two aspects of the completeness of a pattern language are described as ‘functional’ and ‘morphological’. When patterns are applied they introduce new requirements and so other patterns within the language should be able to meet those requirements for functional completeness. In this regard PMES is functionally complete as patterns are provided to resolve the follow on problems generated by applying preceding patterns. For example the pattern EVENTS TO TIME suggest changes at the architectural level which are required in migrating from ET to TT design, and generate a requirement of choosing the appropriate TT design for a particular problem. This is resolved by the pattern TT SCHEDULER which raises the question of implementing the design, and supported patterns are provided in the language as CO-OPERATIVE SCHEDULER, HYBRID SCHEDULER and PRE-EMPTIVE SCHEDULER. Each of these designs generates different requirements such as in case of co-operative design the main problem is handling of long tasks and so a supportive pattern is provided as BUFFERED OUTPUT. However, the language is incomplete morphologically as there are gaps pertaining to multiprocessor designs and communication protocols. Figure 8-2 shows the pattern association map and Table 8-1 gives the pattern thumbnails.

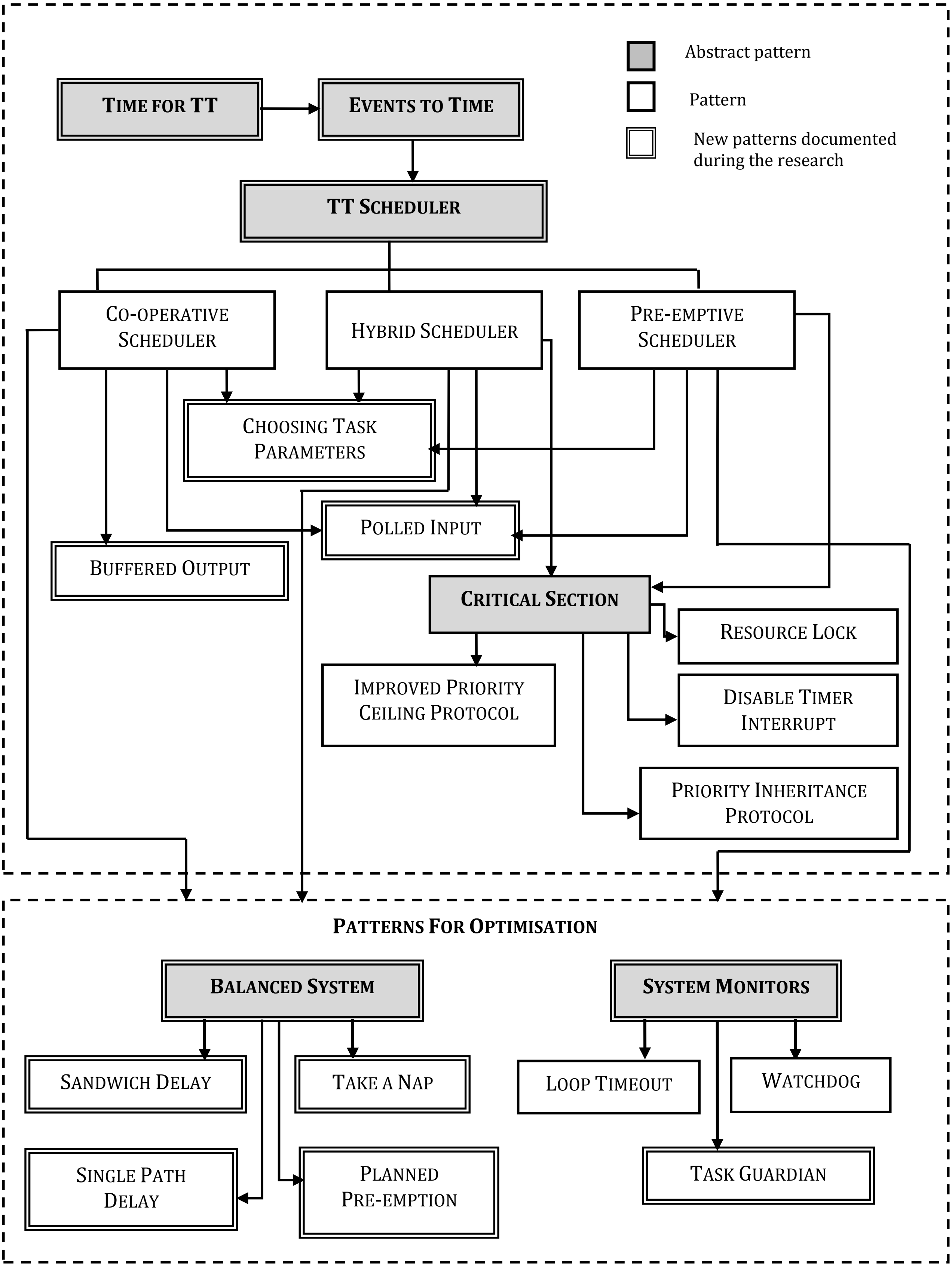


Figure 8-2 Association map for the PMES language

Table 8-1 Thumbnails of patterns in the PMES language

Pattern Name		Description
Patterns for Migration	TIME FOR TT	Discusses when it is appropriate to use TT architecture
	EVENTS TO TIME	Discusses what changes are required to be done at high level when changing the system from ET to TT design.
	TT SCHEDULER	Discusses what a TT scheduler is and its possible types. Also discuss what type of TT scheduler is appropriate in different situations.
	CO-OPERATIVE SCHEDULER	Discusses the implementation of time-triggered co-operative scheduler.
	HYBRID SCHEDULER	Discusses the implementation of time-triggered hybrid scheduler
	PRE-EMPTIVE SCHEDULER	Discusses the implementation of time-triggered pre-emptive scheduler.
	CHOOSING TASK PARAMETERS	Discusses the appropriate choice of task parameters such as offset and order of task execution.
	BUFFERED OUTPUT	Discusses how to deal with long task problem when working with co-operative scheduler.
	POLLED INPUT	Discusses how to deal with external events in time-triggered environment.
	CRITICAL SECTION	Discusses how to avoid conflicts over shared resources during the execution of critical sections.
	RESOURCE LOCK	Discusses the implementation of resource lock in embedded system.
	DISABLE TIMER INTERRUPT	Discusses implementing the simplest way of safe access to shared resources.
	PRIORITY INHERITANCE PROTOCOL	Discusses the implementation of access to shared resources with mutual exclusion and without priority inversion.
	IMPROVED PRIORITY CEILING PROTOCOL	Discusses the implementation of access to shared resources with mutual exclusion avoiding priority inversion, deadlock and blocking chains.
Patterns for Optimisation	BALANCED SYSTEM	Discusses he types of jitter encountered in TT architectures and how to design systems with minimum levels of jitter.
	SANDWICH DELAY	Discusses a technique to reduce period jitter in tasks.
	SINGLE PATH DELAY	Discusses a technique to reduce tasks execution jitter
	TAKE A NAP	Discusses a technique to reduce tasks execution jitter with minimum power consumption.
	PLANNED PRE-EMPTION	Discusses a technique to reduce tick jitter in TTH designs.
	SYSTEM MONITORS	Discusses the idea of implementing monitors in the system which keep track of system state and take necessary actions in case of any errors.
	LOOP TIMEOUT	Discusses the implementation of a software technique to reset systems in case of any unexpected error.
	WATCHDOG	Discusses the implementation of a hardware technique to reset systems in case of any unexpected error that could hang the system.
	TASK GUARDIAN	Discusses the shutdown mechanism of tasks when task overrun happens.

8.4. Patterns for migration

It is important to note that the PMES collection has evolved in stages over time during the course of this research. Various patterns in the collection has been passed through the rigorous review and refinement process for patterns includes *'Shepherding '* and *'Writer's Workshop'* mentioned in Section 6.8. The first part of the collection (containing patterns EVENTS TO TIME, BUFFERED OUTPUT and POLLED INPUT) published in (Lakhani, Das *et al.*, 2009a) has PIEs RS-232 DATA TRANSFER as the implementation example for pattern BUFFERED OUTPUT and the implementation example SWITCH INTERFACE for pattern POLLED INPUT. In the second stage pattern BALANCED SYSTEM and associated patterns SANDWICH DELAY, SINGLE PATH, TAKE A NAP AND PLANNED PRE-EMPTION are published in (Lakhani, Das *et al.*, 2010c).

The patterns CO-OPERATIVE SCHEDULER, HYBRID SCHEDULER, LOOP TIMEOUT and WATCHDOG are part of another set of patterns - the PTTES pattern collection (Pont, 2001). The Patterns including CRITICAL SECTION, RESOURCE LOCK, DISABLE TIMER INTERRUPT, PRIORITY INHERITANCE PROTOCOL and IMPROVED PRIORITY CEILING PROTOCOL have been previously published in (Wang, Pont *et al.*, 2007). Although the details about these patterns are not discussed here they are included in the PMES because of their relevance to the tasks at hand.

The format used to document patterns for the PMES is same as that of PTTES. The format used by the PTTES has already been discussed in Section 6.6 with more details available in Appendix A.

This section will present a brief overview of each individual newly documented pattern in the PMES pattern collection.

8.4.1. Time for TT

This is the first abstract pattern in the collection and its aim to guide developers to decide about whether migration to the TT architecture is an appropriate choice for their application. Being an abstract pattern, it discusses the architectural issues with both ET and TT architecture and provides numerous examples of real applications which are true candidates for TT. This helps the pattern users judge whether TT approach is a perfect match for their application, or they should stick to existing ET design.

8.4.2. Events to Time

The second abstract pattern in the hierarchy is EVENTS TO TIME and this helps in determining an answer to the question: what architectural changes in the ET design will be required? The background section of the pattern describes the structural elements of an ET or TT design. The solution then discusses the steps involved in the migration process. Examples of such steps are shown below:

1. Converting multiple interrupts to a single interrupt. Here it is suggested that the rest of the interrupts should be converted to flags so that they can be polled. The polling mechanism is described in the pattern POLLED INPUT.
2. Conversion of ISRs into tasks.
3. Determination of the appropriate TT architecture (discussed in TT SCHEDULER)

4. Determination of the tick interval (i.e. how frequently the single interrupt in the TT design will be set to occur).

It has also exemplifies a possible ET design and its equivalent proto type TTC design.

8.4.3. TT Scheduler[12]

As the pattern EVENTS TO TIME has discussed the necessary steps in moving from event-triggered design to a time-triggered design, the pattern TT SCHEDULER discusses possible TT solutions and situations in which a particular TT design is appropriate. Being abstract in nature this pattern does not provide implementation details for any of the TT architecture but guides the user which TT architecture (co-operative or pre-emptive) provides a better solution for a particular situation. The abstract pattern then leads to three specific design patterns CO-OPERATIVE SCHEDULER, HYBRID SCHEDULER and PRE-EMPTIVE SCHEDULER. Section 4.3.2 has already discussed the working mechanisms of these TT schedulers. Complete implementation details along with the source code for these patterns can be found in the PTTES collection (Pont, 2001).

8.4.4. Choosing Task Parameters

As already discussed in Section 4.4, it is widely accepted in the embedded systems research community that TT architectures are predictable but difficult to design. One important concern in designing such systems is choosing the

[12] It is worth mentioning here that pattern TT SCHEDULER has been previously introduced in (Pont, Kurian *et al.*, 2008) . While conducting empirical studies during this research (discussed in Chapter 10) it was felt that this pattern is loaded with lot of information and could be decomposed into two different patterns. This resulted in the decomposition of the original TT SCHEDULER pattern into two different patterns for the PMES collection called TIME FOR TT and TT SCHEDULER.

correct task parameters such as the offset and the order of execution so that all the tasks can meet their deadlines. The CHOOSING TASK PARAMETERS pattern provides guidelines along with some examples for choosing the correct task offsets and order of execution.

8.4.5. Buffered Output

A key challenge that developers face with TTC design is keeping task execution times short and predictable. For example transferring large amounts of data from one part of the system to another (e.g. transfer of sensor data in a monitoring system) can take some time. In systems where pre-emption is allowed this activity can be treated as a low-priority task and allowed to run as required. In time-triggered co-operative systems however a long task of this nature will “block” the system (see Figure 8-3). Tasks which are scheduled to run in the next ticks (tick 2 and tick 3) therefore will be missed.

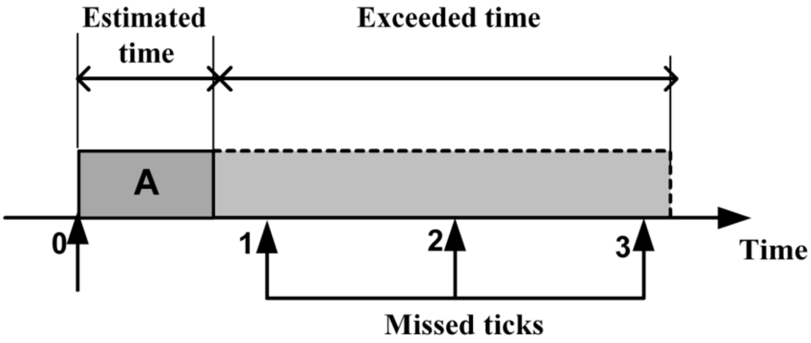


Figure 8-3 Illustrating the problem of a long task in TTC environment

One solution to this problem is described in the BUFFERED OUTPUT pattern. This pattern involves:

- A software buffer.

- A small set of functions used by the tasks in the system to write data to the buffer.
- A periodic task which checks the buffer and sends a block of data to the receiving device when required.

An overview of the use of BUFFERED OUTPUT is shown in Figure 8-4.

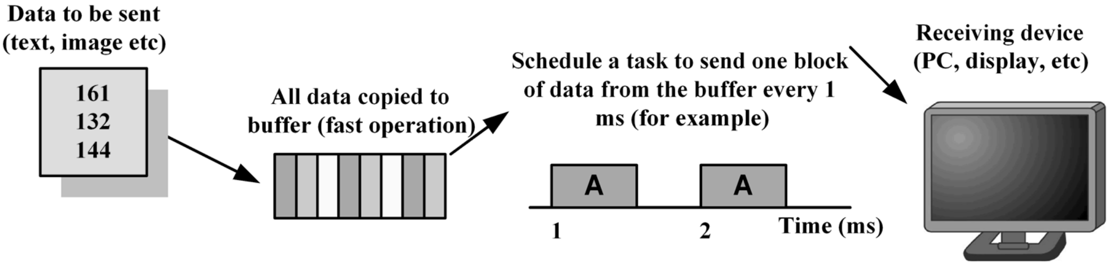


Figure 8-4 An overview of the Buffered Output architecture

The CPU requirements for the pattern BUFFERED OUTPUT are very limited and the technique is generic and highly portable but reasonable care is required to be taken at the design stage to obtain the benefits out of this architecture. For example, if a message takes 0.15ms to transmit, the data transmission task (to check the buffer) should be scheduled at an interval > 0.15ms.

8.4.6. Polled Input

This pattern is introduced to deal with any external inputs that may arise as a result of an event for example pressing a switch, completion of an analogue to digital conversion, arrival of a message on a CAN bus and so on. A POLLED INPUT should meet the following specifications.

- The interrupt associated with the event should be disabled (only one interrupt associated with the timer responsible for generating system "ticks" should be enabled.
- A periodic task which polls for the occurrence of the event (that is, checks the event flag).
- The period of the above task should be set to a value equal to the minimum inter-arrival[13] time of the event to be polled.

This pattern is useful for reading switch inputs and scanning keypad interfaces which are very common requirements in embedded applications.

8.4.7. Balanced System

BALANCED SYSTEM is an abstract pattern and is introduced to achieve optimisation of systems after migrating to time-triggered designs. Though more predictable in nature compared with event-triggered systems, time-triggered systems are still susceptible to jitter. Choosing the correct time-triggered architecture does not fully guarantee system predictability as there are a number of other factors which could make a time-triggered system unpredictable and it is important to know these in advance. These include release time, execution time, finish time and deadline. The prior knowledge of these parameters plays an important role in guaranteeing the overall predictability of the system. However, systems those run in practice generally show considerable variations or jitter. To understand the concept of jitter more clearly, consider the different instances of a task (Task A) in Figure 8-5, where '*r*' is the release time, '*s*' is the start time and '*f*' refers to finish time of the task.

[13] This is previously explained in Section 2.3 for sporadic task as minimum time difference between the release times of any two consecutive instances of the same task.

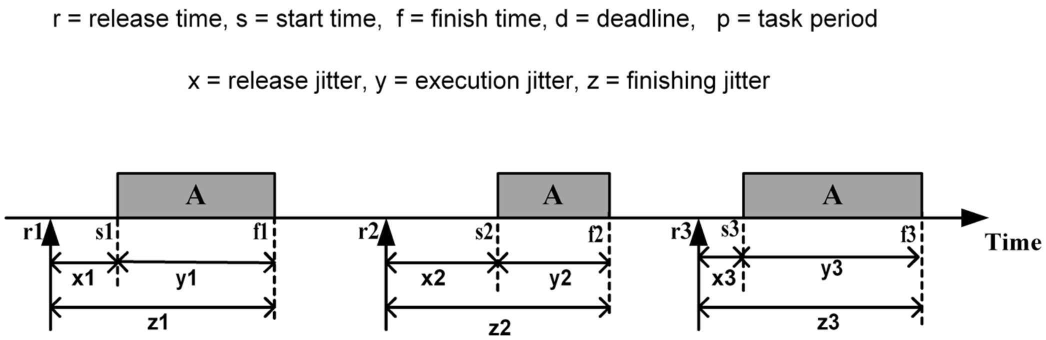

r = release time, s = start time, f = finish time, d = deadline, p = task period
x = release jitter, y = execution jitter, z = finishing jitter
A
A
A
Time
r1
s1
f1
r2
s2
f2
r3
s3
f3
x1
y1
z1
x2
y2
z2
x3
y3
z3

Figure 8-5 Illustrating the presence of release, execution and finishing jitter in different instances of a task

For tasks in TT systems, the release time can be considered as the point at which one would ideally expect a task to start its execution. In actual practice this is delayed due to factors such as the scheduler overhead and the variable interrupt response times (Liu, 2000; Maaita and Pont, 2005). The actual start time of a task is always deviated from its (pre-determined) release time so tasks always suffer from release jitter - see unequal values of *x1,x2* and *x3* in Figure 8-5.

In real-time systems one important parameter is the upper bound of the execution time for a task, previously defined in Section 3.3.1 as the worst-case execution time (WCET). Unfortunately, determining the WCET of tasks is rarely straightforward (Puschner, 2002; Puschner and Burns, 2002; Puschner, 2003). This is because the program code of a task may contain conditional branches and/or loops and each may take different times to execute (Liu, 2000). The decision between one branch and the other during task execution is dependent on the input data. This makes predicting a branch prior to execution a very difficult task. All these factors lead to variable execution time of a task and this is known as execution jitter (see unequal values of $y1$,$y2$ and $y3$ in Figure 8-4). The cascading effects of release and execution jitter will result in the deviation of task finish time, shown as $z1$, $z2$ and $z3$. Ideally, a predictable system should be jitter free. Considering Figure 8-5 once again, it can be stated that in a jitter free system:

$$\boldsymbol{x1 = x2 = x3}$$ **Equation 8-1**

$$\boldsymbol{y1 = y2 = y3}$$ **Equation 8-2**

$$\boldsymbol{z1 = z2 = z3}$$ **Equation 8-3**

Some hardware features such as variations in the frequencies of the oscillator and use of cache memories (Kirner and Puschner, 2003) also contribute to jitter in tasks.

For some applications, such as data, speech or music playback for example these variations may make no measurable difference to the system. However, for applications in real-time control systems which involve sampling, computation and actuation, such delays in operations are very risky for the overall performance of the system. The presence of jitter can have a degrading impact on the performance of real-time systems or can even lead to critical failure (Marti, 2002). A BALANCED SYSTEM is more robust against the presence of various types of jitters in the system and results in more predictable timing behaviour of the system. The various ways of achieving a balanced system are SANDWICH DELAY, SINGLE PATH DELAY, TAKE A NAP and PLANNED PRE-EMPTION.

8.4.8. Sandwich Delay

In embedded applications there is a high possibility that two tasks are required to run one after the other. In such a situation variable execution time of the task could affect the release time of the other task running just after it. Suppose a system is executing two functions periodically using a timer ISR, as outlined in Listing 8-1. Please note that the programming language used in all code listings is ‘C’ language.

```
// ISR invoked by timer every 10ms

void Timer_ISR (void)

{

Do_X();  //WCET approx 4.0ms

Do_Y();  //WCET approx 4.0ms

}
```

Listing 8-1 System executing two functions using timer ISR

According to the Listing 8-1, function Do_X() will be executed every 10ms. Similarly, function Do_Y() will be executed every 10 ms, after Do_X() completes. For many resource-constrained applications (for example, control systems) this architecture may be appropriate. However, in some cases, the risk of jitter in the start times of function Do_Y() may cause problems. Such jitter will arise if there is any variation in the duration of function Do_X(). In Figure 8-6, the jitter is reflected in differences between the values of *ty1* and *ty2* (for example).

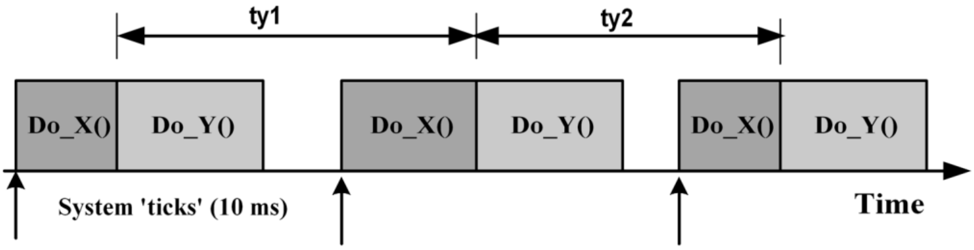


Figure 8-6 Impact of variations in the duration of task Do_X() on the release jitter of task Do_Y()

A SANDWICH DELAY can be used to solve this type of problem. More specifically, it provides a simple but highly effective means of ensuring that a particular piece of code always takes the same period of time to execute: this is done using two timer operations to "sandwich" the activity which it is required to perform. Please refer to code segment in Listing 8-2.

```
// ISR invoked by timer overflow every 10ms
        void Timer_ISR(void)
        {
        Set_Sandwich_Timer_overflow(5);
        Do_X();
        Wait_Sandwich_Timer_Overflow();
        Do_Y();
        }
```

Listing 8-2 Pseudo code for Sandwich Delay

The timer is set to overflow after 5ms (a period slightly longer than the WCET of Do_X()). This timer starts before the execution of Do_X() starts – after the function is complete – Do_Y() will wait for the timer to reach 5ms value. In this way, it can be ensure that as long as Do_X() does not exceed a duration of 5 ms – Do_Y() runs with minimum jitter as shown in Figure 8-7.

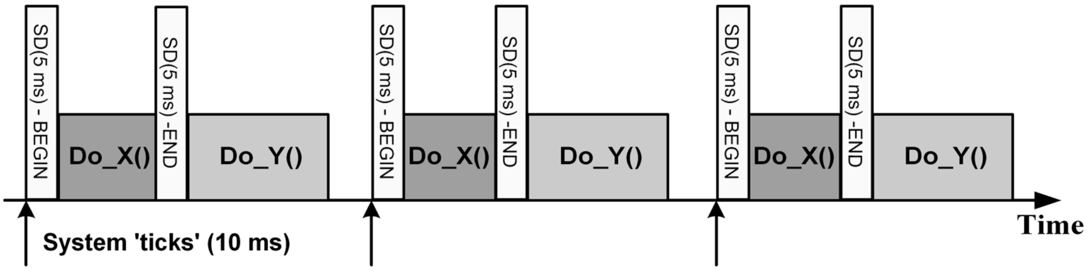


Figure 8-7 Illustrating the use of the Sandwich Delay technique

Sandwich delays are effectively found useful for pre-emptive designs for example TTH designs to control the execution jitter of pre-emptive tasks.

8.4.9. Single Path Delay

Variable execution times of tasks can lead to unpredictable behaviour in systems. To understand this more clearly, consider a system running tasks A, B and C (with equal release time = 0) as shown in Figure 8-8.

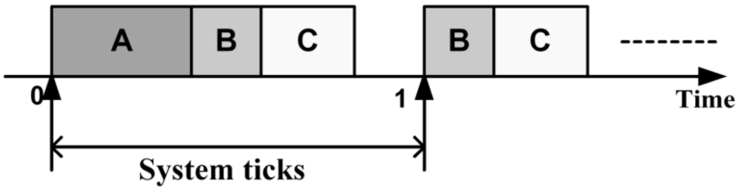


Figure 8-8 Three tasks with same release time scheduled to run in a tick

If for any reason, task A takes a longer time than its estimated WCET, there will be consequences for it. For example: (a) task C will run before task B if it has higher priority than task B (b) task B not being able to finish with the tick could lead to problems in the system. The situation is depicted in Figure 8-9.

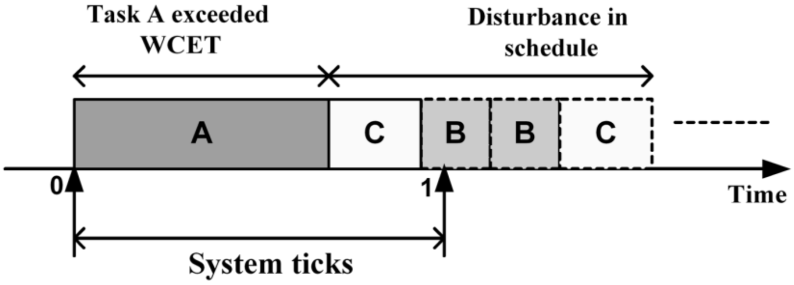


Figure 8-9 Illustrating the change in system behaviour if the execution time of task A takes longer than expected

The point to be noted here is that if task A varies in duration it will affect the overall system behaviour. Tasks involving loops and decision structures (for example, ‘if-else’, ‘switch’, etc.) are more likely to have variable execution times. If such tasks can be balanced, more stable and predictable system behaviour can be achieved.

The single-path programming approach was introduced by Peter Puschner (Puschner, 2003). The aim of this approach is to ensure that blocks of code involving loops or decision structures will have a single execution path and therefore a fixed execution time. SINGLE PATH DELAY can be achieved by replacing input-data dependencies in the control flow by predicated code instead of branch code. Thus, the instructions are associated with predicates and are executed if the predicate evaluates to 'true'. In other case (if the instruction evaluates to 'false'), the microprocessor replaces the instruction with a NOP (no-operation) instruction. This result in a higher but fixed execution time compared to the traditional programming approach.

As an example of a single-path consider the code Listing 8-3 in which Single Path Delay approach is applied to a code snippet involving if-else statement. Temporary variables `temp1` and `temp2` are used to hold results of expressions `expr1` and `expr2` respectively. The conditional move instruction `movt` copies the value of `temp1` to the variable `result` if the test condition evaluates to true, otherwise the processor performs a `No Operation` (NOP) instruction. On the other hand, if the test condition evaluates to false, `movf` will copy the value of `temp2` to result otherwise NOP instruction will be executed. As a result, the translation basically generates a sequential code as shown in Listing 8-3 (the right hand side code segment).

```
if (cond)

{

   result = expr1;

}

else

{

   result = expr2;

}
```

```
temp1 = expr1;

temp2 = expr2;


test cond;


movt result,temp1;

movf result,temp2;
```

Listing 8-3 Sequential code generated from a branching statement using if-else conversion [Adapted from (Puschner,2003)]

The main drawback of this approach is that it is limited to hardware which supports "conditional move" or similar instructions. It is likely to increase the power consumption because the CPU will always execute the single-path code for a fixed (maximum) period. During this time, the processor will be in "full power" mode.

8.4.10. Take a Nap

This pattern is introduced to overcome the limitations posed by the pattern SANDWICH DELAY and the pattern SINGLE PATH DELAY. In systems where power consumption is a concern, neither of the two patterns discussed previously is an attractive solution because in both the cases to achieve BALANCED SYSTEM the CPU operates at "full power" mode at all times. TAKE A NAP provides a solution to achieve BALANCED SYSTEM with reduced power consumption and is based on the technique discussed in (Gendy and Pont, 2007). It works by putting the control flow statement within a SANDWICH DELAY. This will ensure that the particular piece of code will always have constant execution time. For example, consider the code segment in Listing 8-4.

```
for (i = 0; i< x;  i++)
{
    // body of the loop
}
```

Listing 8-4 A simple for loop

The execution time of the loop is dependent on the value of the variable x. Let `MAX` be equal to the maximum number of iterations the loop can execute. Let `Time(x)` be equal to the time spent in executing `x` iterations. The value of `Time(x)` may be measured using hardware timers. Therefore, the time spent in performing (MAX – x) iterations may be calculated using the value of Time(x) as given in Equation 8-4.

$$Time(MAX - x) = (MAX - x) * Time(x)/x \qquad \textbf{Equation 8-4}$$

Once the loop executes x number of times, the processor is put to sleep for a duration equal to Time (MAX – x). A timer interrupt may be generated when the hardware timer count reaches the value Time(MAX – x)and this can be used to awaken the processor. Using this technique, code segment in Listing 5-5 is ensured to always (irrespective of the value of x) have a constant execution time equal to the value of Time(MAX) (i.e. the time spent in executing MAX number of iterations of the for loop). Thus, in addition to enabling a power-saving mode of the processor, the resulting 'balanced' code with the SANDWICH DELAY incorporated, provides an additional layer of predictability to the real-time system. The balanced version of the loop in Listing 8-4 may be written as shown in Listing 8-5.

```
// Start the timer
 Timer_Start();
 for (i = 0; i< x; i++)
 {
 // body of the loop
 }
 // Stop the timer
 Timer_Stop();
 // Store timer count value after x iterations
```

```
Time(x) = Timer_Count_Value;


// Determine value of Time(MAX – x)

Time(MAX – x) = ((MAX-x) * Time(x))/x;


// Reset the timer

Timer_Reset();


// Timer interrupt to occur after Time(MAX–x)

Set_Interrupt(Time(MAX-x)+"safety margin");


// Put processor to sleep

Processor_Sleep();
```

Listing 8-5 Balancing the section of a code with reduced power consumption [Adapted from (Gendy and Pont, 2007)]

It must be noted that the 'for' loop in the code segment above must run at least once for the value of `Time(MAX – x)` to be determined. Furthermore, a small 'safety margin' has been added to the calculated time to ensure that there is sufficient time for the processor to enter sleep mode even when the loop is executed for the maximum number of iterations.

8.4.11. Planned Pre-emption

This pattern is used to achieve a Balanced System specifically with TTH schedulers. A hybrid scheduler provides limited multi-tasking capabilities to the system. Such systems could exhibit unpredictable behaviour for the following two reasons (Maaita and Pont, 2005):

1. Existence of unbalanced code branches in the timer ISR which leads to variable ISR execution times. This in turn leads to unpredictable scheduler behaviour represented by the appearance of task starting jitter.
2. The existence of CPU instructions with different execution times (i.e. in terms of CPU cycles required to execute the instruction).

This leads to variable timer interrupt response times as each of the periodic timer interrupts which take place throughout the life cycle of the application can occur while the CPU is in one of the two different states. The CPU may either be running in sleep (idle) mode, or while it is running an instruction and where the interrupt is only serviced once the currently executing instruction is finished shown in Figure 8-10.

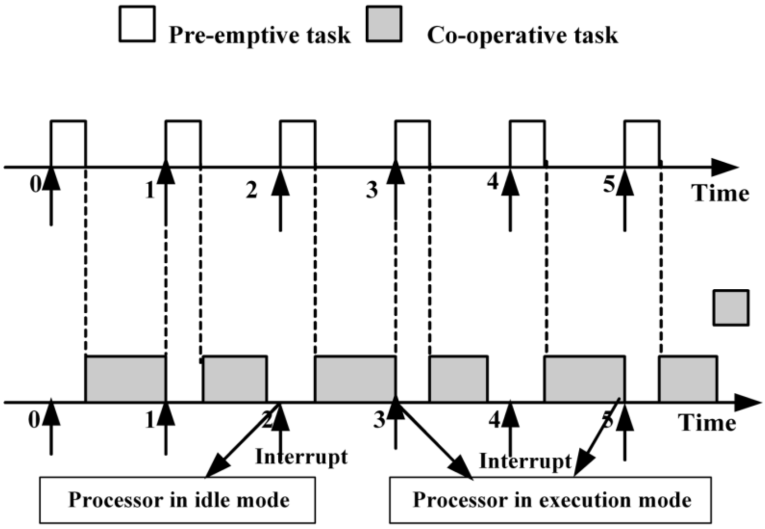


Figure 8-10 Illustrating the Occurrences of timer interrupts when processor is in different modes

The possible occurrences of timer interrupts could lead to variable timer ISR response time which translates to task release jitter. In TTH design this release jitter has the largest impact on tasks which are regularly executed after a timer tick has occurred and is, therefore, referred to as "tick jitter".

Keeping the processor in the same state as all interrupts take place would likely reduce the tick jitter (Maaita and Pont, 2005). PLANNED PRE-EMPTION makes use of another hardware timer to put the processor to power saving mode or 'sleep mode' before the scheduler timer interrupt occurs thus keeping the processor in the same state every time. This will eliminate the jitter as the time required to leave the sleep mode and resume normal execution is a static value (Martin, 2005). Figure 8-11 illustrates the operation of PLANNED PRE-EMPTION.

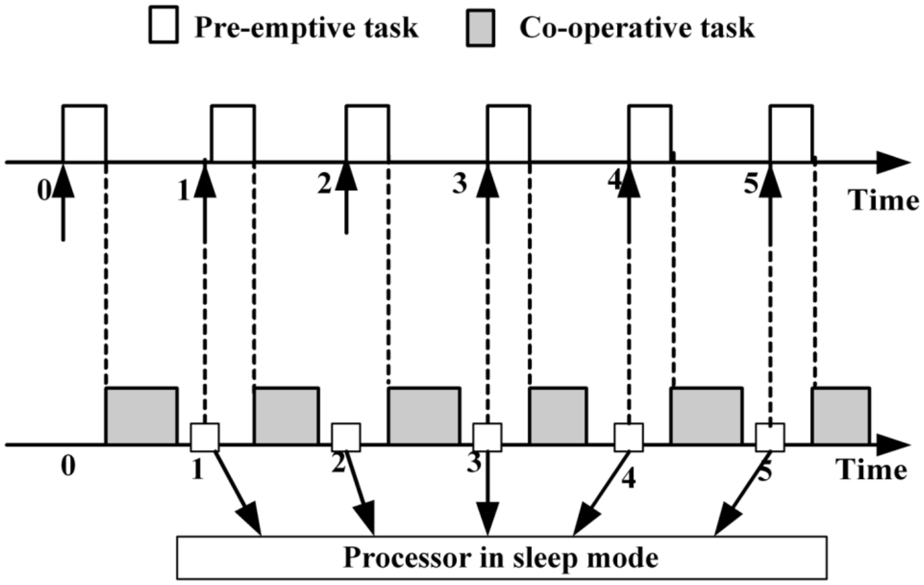


Figure 8-11 Illustrating the operation of 'Planned Pre-emption' where processor is put to sleep mode just before the system tick occurs

Implementation of PLANNED PRE-EMPTION is shown in Listing 8-6. The extra timer used to put the processor to sleep mode is named as "PP-timer" being use for PLANNED PRE-EMPTION. To set the overflow value of the PP-timer it is important to know the WCET in advance so that the processor can have enough time to go to sleep mode before the scheduler timer interrupt occurs. PLANNED PRE-EMPTION will reduce the tick jitter as the time required to leave the sleep mode and pursue normal execution is a static value (Kopetz and Bauer, 2002).

```
while(1)
{
// Dispatch Co-op tasks
C_Dispatch();
}
 void C_Dispatch(void)
 {
 // Go through the task array
 // Execute Co-operative tasks as required.
 // The scheduler may then enter idle mode
  Sleep();
  }
  void P_Dispatch_ISR(void)
  {
  // Start idle timer
  ITimer();

  //Dispatch pre-emptive task
  P_Task();
  }
  // Idle timer ISR
```

```
void Idle_Timer_ISR(void)

{

 Sleep();

 }
```

Listing 8-6 TTH Scheduler with Planned-Pre-emption

8.4.12. System Monitors

SYSTEM MONITORS is an abstract pattern that represents techniques which are intended to keep an eye on system functionality to make sure that system is working normal or as expected. They act like guards for the system. In the event of unexpected occurrences for example, incorrect initialization of some peripheral or variables, the system will not produce the results as expected and the application can possibly hang for an infinite time. In such a situation the implementation of SYSTEM MONITORS can prevent the system from hanging in any unstable states. SYSTEM MONITORS can be implemented with the use of hardware such as timers (see pattern WATCHDOG (Pont, 2001)) or software (see pattern LOOP TIMEOUT (Pont, 2001)). This research has introduced a new pattern called TASK GUARDIAN described next.

8.4.13. Task Guardian

This pattern is based on a technique proposed by Hughes and Pont (2008) to handle task overruns in TTC systems. Despite many advantages, a pure TTC architecture has a failure mode which has the potential to greatly impair system performance: this failure mode relates to the possibility of task overruns (see Figure 8-12 and Figure 8-13).

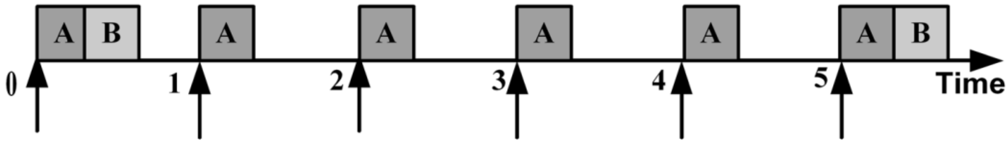


Figure 8-12 Illustrating a TTC based system running in normal conditions

Figure 8-12 illustrates a TTC design running two tasks, A and B. Task A runs every 1ms and Task B runs every 5ms. This system operates as required, since the duration of Task A never exceeds 0.4ms. Now consider Figure 8-13 which illustrates the problems that result when Task A overruns. In this case, it is assumed that the duration of Task A increases to approximately 5.5ms. The co-operative nature of the scheduling in this architecture means that this task overrun has very serious consequences (Hughes and Pont, 2008).

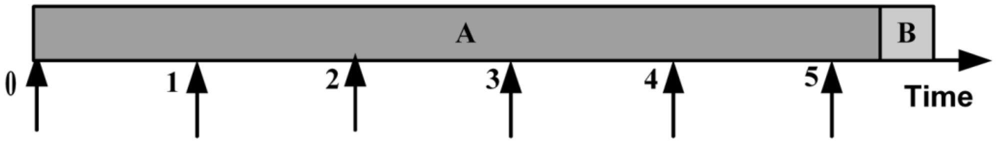


Figure 8-13 Illustrating the TTC system when task A overruns

TASK GUARDIAN suggests the shutdown mechanism for a task which is found to be overrun of execution longer than expected. In summary, the Update ISR in the TTC scheduler detects the task overrun, and returns control to the `End_Task` (a function designed to provide a shutdown mechanism for an overrun task). `End_Task` will be responsible for unwinding the stack, as required, and as far as possible normal program operation will continue.

8.5. Conclusions

This chapter has introduced a new pattern collection intended to support the developers of embedded applications in the migration process from event-triggered design to time-triggered designs. It has discussed the rationale for proposing the pattern collection and described some of the new patterns in the collection. However, there are still a number of questions that need to be addressed before general application and the deployment of the pattern collection with guaranteed performance can be envisaged. These questions concern the efficacy of the patterns and their usefulness, their practical use in real embedded applications and how and why practitioners can benefit from them. The next three chapters of this thesis will explore these questions further and provide answers to them.

CHAPTER 9. APPLYING THE PATTERNS TO REAL APPLICATIONS

9.1. Introduction

The research presented in this thesis was initiated with the belief that for many embedded applications, migration from event-triggered (ET) to time-triggered (TT) architecture improves reliability (see Section 1.2). In support of this belief three motivating examples from the literature are described in CHAPTER 2, and CHAPTER 5 discussed many issues that spawn the migration of embedded applications. The examples quoted in CHAPTER 2 demonstrate that the paramount goal of migration is achieving reliability in systems. High reliability is the most important criterion for safety-related embedded systems operating in unpredictable environments such as military applications, automobiles and air-craft. However, even in more mundane domestic applications, for example an alarm clock that fails to sound on time or a video recorder that operates intermittently, where failure is relatively inconsequential, poor reliability can have other impacts such as reduced sales etc. (Pont, 2001)

The aim of the work described in this chapter is two-fold:

- To investigate the impact of migration (from ET design to TT design) on system reliability with empirical results and comparison of the performance of both the designs.
- To identify the applicability of various patterns in the PMES collection at different stages of the migration process.

To achieve these goals, two examples of non-trivial embedded applications are employed. This chapter aims to demonstrate the application of patterns for real applications in the conversion process from ET design to TT design. Section 9.2 describes the hardware platform specifications and the methodology adopted for various calculations for the experiments described in this chapter. Section 9.3 and 9.4 describes each of the example application in detail along with the process of applying the patterns in the PMES collection to achieve TT designs, and how they helped in improving the reliability. Section 9.5 presents a brief discussion on the results obtained and conclusions are presented in Section 9.6. Please note that the work described in this chapter have been published in (Lakhani and Pont, 2012a).

9.2. Hardware specifications and methodology

For the case studies discussed in this chapter, two different examples of embedded applications with different requirements were chosen. In each case study, the existing design of the application and its flaws are analysed. After migration to the TT architecture a comparison of results between the ET and the TT architectures is presented. Specifications for the hardware platform used for both the case studies are described as follows.

9.2.1. Hardware platform

Both the case studies are conducted on an NXP LPC2378-STK microcontroller mounted on an OLIMEX prototyping board (see Figure 9-1). Such prototyping boards are commonly used during research and development case studies as they provide the neatest and well-managed circuits with processor and all the necessary peripherals connected, well-marked and often labeled. Usually on such boards all the required circuits are self-integrated and well-connected which saves time compared with circuits with unmanageable bulky wire connections. The prototyping boards are designed to facilitate experimental work by engineers/developers as all the required components are held fast and do not roll-off or get disturbed by movement.

The LPC2378 microcontroller is based on 16/32bit ARM7TDMI-S CPU running at up to 72 MHz with 512 KB of flash memory and four on-chip timers. This hardware is chosen because of its wide applicability in industry and is designed for applications commonly used in industrial control, medical systems, protocol conversion and communications. Further details of the hardware can be found in the hardware data sheet available online at (OLIMEX2378, 2012).

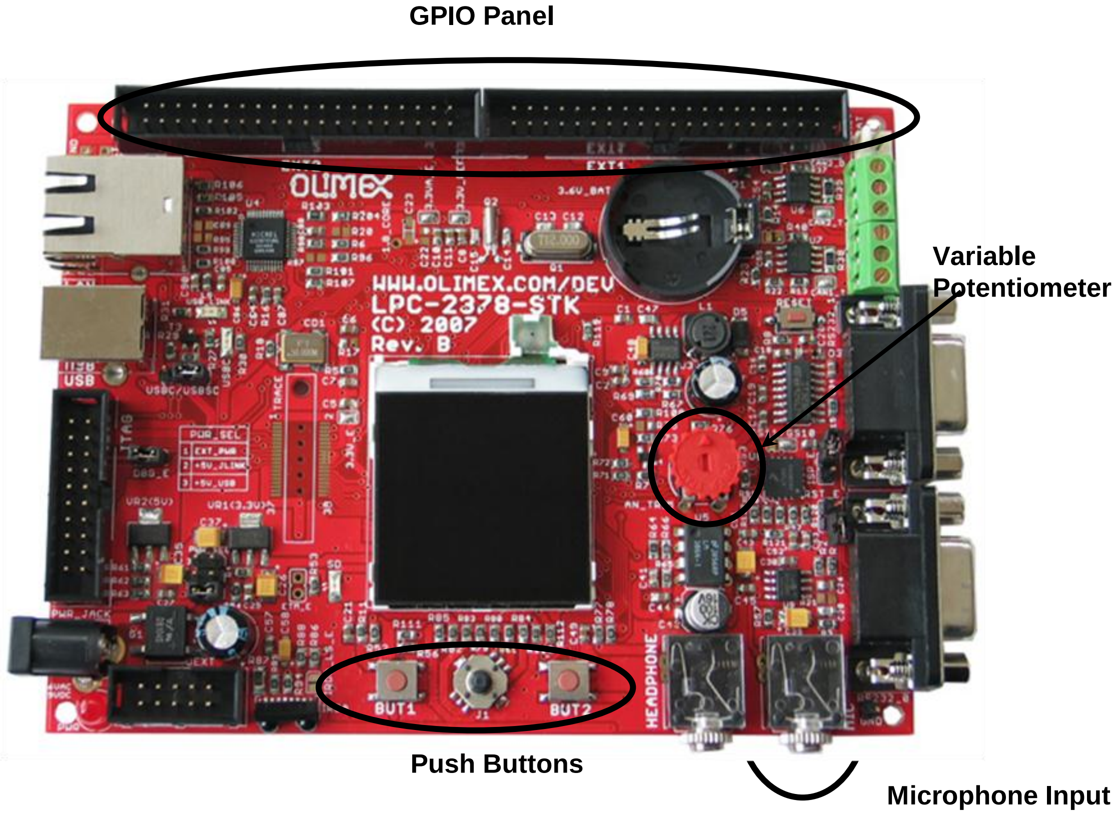


Figure 9-1 Prototype board for ARM 7 LPC2378STK [courtesy: (OLIMEX2378, 2012)]

9.2.2. Measurements approach and methodology

In order to investigate the system performance, measurements were taken of CPU utilization, memory utilization and power consumption. Jitter measurements were also taken to analyse the system reliability.

Memory utilization (Stack and RAM) is measured using the RapidiTTy ®tool set (RapidiTTy, 2009). In RapidiTTy, the code and data memory are simply parsed from the memory map that the linker spits out. RapidiTTy calculates the stack usage by parsing the listing file, finding all the stack allocations in each function and then building up a call tree for all the functions in the application. This information is then combined up the call tree to find the maximum stack usage at every stage.

To measure experimentally, the system active time ‘$\text{CPU}_{\text{active}}$’, the execution time ‘C’ of all the tasks and ISRs is calculated. For this purpose, a general purpose input/output (GPIO) pin on the GPIO panel on board (highlighted in Figure 9-1) is used. The pin is made high at the start of each task/ISR and low at the end. The pulse width of this time span is then measured using a National Instruments data acquisition card ‘NI-PCI-6035E’ in conjunction with LabVIEW 10.0 software (NI, 2012). The sampling rate of the acquisition card used is 100 KHz which is not high enough to get exact and precise results however it is good enough to provide a trend of results. More accurate results can be obtained by using a data acquisition card of higher sampling rate.

Note that the total time spent by the CPU, 'CPU$_{total}$' is the timing window assumed to perform calculations for these experiments (the assumed timing window = 300,000 system ticks where 1system tick = 1ms).

The active time for the processor 'CPU$_{active}$' is the time it spends in executing all the tasks in the system and is calculated using Equation 9-1.

$$CPU_{active} = \sum C_i + Scheduler\ overhead$$ **Equation 9-1**

The scheduler overhead in Equation 9-1 is added to get the precise active time. It is the time span which is spent by the CPU in switching from one task to another. It is measured separately by setting a GPIO pin high when the timer ISR controlling the periodic tasks in the system is called and by setting it to low before it goes to sleep mode.

The time spent by the processor in idle mode 'CPU$_{idle}$' is calculated by subtracting the time for which it stayed active from the assumed timing window and is presented in Equation 9-2.

$$CPU_{idle} = CPU_{total} – CPU_{active}$$ **Equation 9-2**

The percentage of CPU utilization is calculated using Equation 9-3

$$\%CPU\ Utilization\ = (CPU_{active}/CPU_{total})\ *100$$ **Equation 9-3**

For measurements of power consumption, the average current consumption 'I$_{CPU}$' by the ARM processor is calculated using Equation 9-4 below:

$$I_{CPU} = [(CPU_{active}*IDD_{(DCDC)active}) + (CPU_{idle} * IDD_{(DCDC)pd})]/CPU_{total}$$ **Equation 9-4**

In the above equation, $I_{DD(DCDC)act}$ is the active mode DC-to-DC converter supply current (3.3V) with all peripherals enabled and the CCLK operating at 72MHz. $I_{DD(DCDC)pd}$ is the power-down mode DC-to-DC converter supply current (3.3V) for the ARM7 processor. Typical values of these parameters given in the datasheet of hardware are 125mA and 113µA respectively. Using these parameters the average power consumption by the processor 'P_{CPU}' is calculated as shown in Equation 9-5.

$$P_{CPU} = I_{CPU} * V_{CPU} \qquad \textbf{Equation 9-5}$$

where 'V_{CPU}' = 3.3V (voltage across the processor).

In Section 8.4.7 as part of the discussion on the pattern BALANCED SYSTEM, the concept of 'release jitter' has been introduced. It is the deviation of the actual start times of the task from their release times (Tindell, Burns *et al.*, 1994; Bril, Steffens *et al.*, 2004). Release jitter can be expressed in relative or absolute values (Buttazzo, 1997). Relative release jitter is defined as the maximum deviation in the start time of two consecutive instances and absolute release jitter is the maximum deviation of the start time in all instances.

According to (Baurah, Buttazo *et al.*, 1999) the absolute jitter of task T_i can be defined as

$$Absjitter(T_i) \stackrel{\text{def}}{=} = max\ (\ P_i(max)\ -P_i,\ P_i - P_i(min)\ ) \qquad \textbf{Equation 9-6}$$

where:

P_i = the time interval between successive completions (or invocations) of task T_i

P_i(max) = the maximum time interval between successive completions (or invocations) of task T_i

P_i(min) = the minimum time interval between successive completions (or invocations) of task T_i

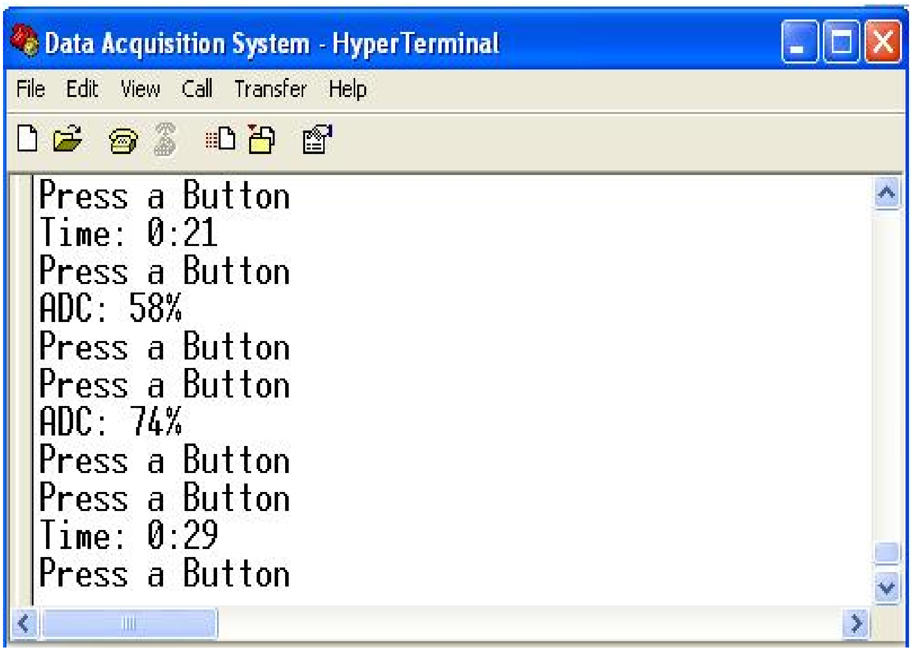
Release jitter is measured experimentally by making a pin high at the beginning of the timer ISR and making it low just before the task starts. A Labview script is then used to measure the maximum and minimum values of release jitter using Equation 9-6.

9.3. Example 1: Data Acquisition System

9.3.1. System functionality

Generally abbreviated with the acronym 'DAQ', the data acquisition system is a widespread control application and is used to measure temperature, pressure, fluid flow and light intensity etc. One of the main activities in such systems is the conversion of analogue signals to digital values. The application discussed here takes data samples from an analogue-to-digital converter (ADC) every 5 milliseconds and translates the sampled value to an appropriate string of characters. By pressing the push button 'BUT1' (user request) available on the board (highlighted in Figure 6-1), the application displays the translated ADC value on the hyper terminal. The user is allowed to change the input to the ADC using the variable potentiometer available on the board (see Figure 9-1) and thereby enabling to view the different values of the sampled data. In addition, the application also displays an elapsed time value since the microcontroller was last reset by pressing the push button 'BUT2'. In normal conditions when there is no user request the application will keep on displaying "Press a Button" message on the windows 'HyperTerminal'.

A sample output of the application is shown in Figure 9-2.


Data Acquisition System - HyperTerminal
File Edit View Call Transfer Help

```
Press a Button
Time: 0:21
Press a Button
ADC: 58%
Press a Button
Press a Button
ADC: 74%
Press a Button
Press a Button
Time: 0:29
Press a Button
```

Figure 9-2 A sample output shown using hyper terminal for the DAQ system

9.3.2. Even-triggered design

The existing design of this application is based on the ET architecture with loose TT operation. Such hybrid designs are common in control applications (Short, Pont *et al.*, 2008). Some of the tasks in the system which require continuous update (for example, ADC data sampling and displaying the message 'Press a Button' continuously on the hyper terminal) are periodic in nature. Task specifications of the periodic tasks in the system are shown in

Table 9-1. Other than the periodic tasks, what follows are the active interrupts in the system.

- TIMER 0, to control all the periodic tasks shown in Table 9-1. The overflow value of this timer interrupt is set as 1ms which is basically the ‘tick interval’ for the periodic tasks.
- EINT3, External interrupt 3 which is invoked when the user presses ‘BUT 1’ to display ADC data on to the hyper terminal.
- EINT0, External interrupt 0 which is invoked when the user presses ‘BUT2’ to display elapsed time value.

Table 9-1 Specifications of the periodic tasks in ET design for DAQ system

Task ID	Task Description	Period (ms)
ADC_Sample	Reads the voltage from a particular ADC channel.	5
ADC_Translate_Update	Translates the ADC data to an appropriate string of characters.	10
Elapse_Time_Update	Calculates the elapsed time since the microcontroller was last reset.	1000
HT_Display_Update	Displays appropriate data on the hyper terminal when the user press a button otherwise it displays the message ‘Press a Button’	1000
Flashing_LED	Periodic flashing of an LED to indicate that system is alive.	1000

In the application discussed in this case study, the interrupt ‘TIMER 0’ is handled by the ‘FIQ’ (Fast Interrupt Request) category, while the external interrupts ‘EINT 0’ and ‘EINT 3’ are handled using the ‘IRQ’ (Interrupt Request) category.

On the ARM processor, interrupts are divided into two levels: FIQ or IRQ. FIQ is a higher priority interrupt than IRQ. While assigning interrupts, a standard practice is, FIQs are normally reserved for a single interrupt source that requires a fast response time while IRQs are normally used to handle general purpose interrupts (Sloss, Symes *et al.*, 2004).

9.3.2.1 Performance measures for the event-triggered design

Performance measures listed in Equation 9-1 to Equation 9-6 are measured for the ET design and are tabularized in Table 9-2.

Table 9-2 Performance measures for the ET design for DAQ system

Parameters	Values
CPU utilization	67.86%
Idle time	32.14%
Jitter (NBP)[14]	0.05µsecs
Jitter (VBP)[15]	0.3,1.25,1.45,1.75µsecs
Power consumption	0.258watts
Stack	3616 bytes
RAM	6554 bytes

Please note that the jitter measurements are taken for the ADC sampling task only, this is because the task ADC_Sample is responsible for acquiring data in the system (runs every 5ms) and the accuracy of the data displayed is dependent on the accuracy of the sampling task. During the experiment, it was interesting to note that the release jitter in the ADC_Sample task showed variations with the increase in number of external interrupt calls. As the number of external interrupts increased in a timed window increasing values of jitter were observed.

[14] Jitter in the absence of any button press call or 'No Button Press'

[15] Jitter with 'Varied Button Press' calls. This reflects the situation in which the user can press the button randomly any number of times.

It is also important to mention here that in reality, calculating the exact CPU utilization for ET system is nearly impossible because of the aperiodicity of events. However, for the present case study equal number of external interrupts are considered in the defined timing window (300,000 ticks) for both (ET and TT) designs to ensure a fair comparison between the two designs.

9.3.2.2 Reasons for variations in the release jitter

ET designs are considered to be more responsive because of the rapid handling of external interrupts as they arrive. This feature sometimes could appear as a problem instead of providing benefits to the system as observed in the experiment discussed in this case study. This can be more clearly explained by considering the situation as depicted in Figure 9-3. As the timer generates the system tick, it runs the FIQ handler and then dispatches the ready tasks. Once all the tasks have finished execution, the system goes into the sleep mode.

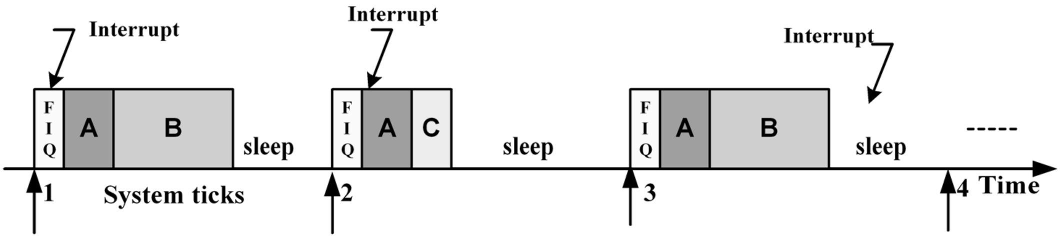


Figure 9-3 Illustrating possible interrupt arrivals during task execution

In the situation when any external event arrives three possible situations are considered here:

1. Interrupt arrives while the FIQ is running.
2. Interrupt arrives when the system is running a task.
3. Interrupt arrives when the system is in sleep mode.

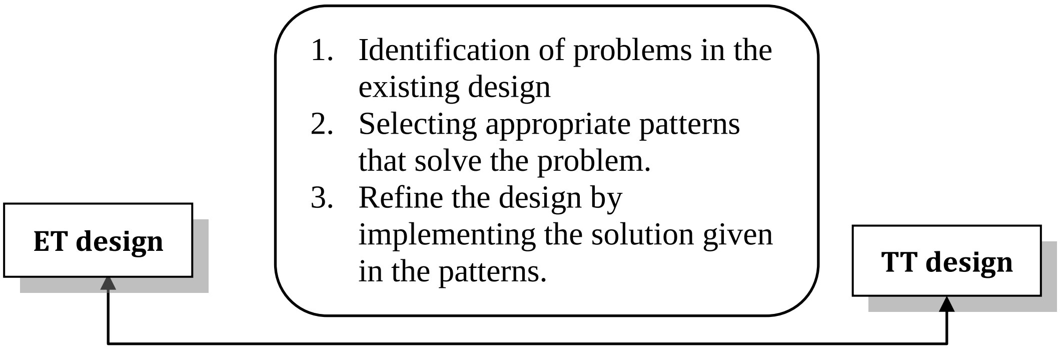
As mentioned in Section 9.3.1, external interrupts are handled using the IRQ therefore in case of situation (1) above the IRQ handler will have to wait for execution until the high priority FIQ finishes execution. In the case of situation (2) the IRQ handler will start executing immediately as it arrives and will interrupt the currently running task. Apparently, scenario (3) looks simple as the processor is in idle mode and the IRQ can start its execution at the same instant. However, there is a possibility that the FIQ happens while the IRQ has not finished execution. As a consequence, the IRQ handler execution will be interrupted by the FIQ handler as the FIQ has a higher priority by default. All the situations explained above lead to variations in task timings and are manifested as variable jitter values for the sampling task. Variations in the release jitter of the ADC_Sample task leads to unpredictability and make the system unreliable. This is because it can lead to jitter during the period of other tasks and inaccurate values of data will then be displayed. This implies that the existing architecture is not reliable and so needs to be migrated to a more predictable and reliable design.

9.3.3. Migration to a time-triggered design

Before converting the DAQ system to a TT design, certain issues are required to be resolved such as: Is it appropriate to convert this system to a pure TT solution with a single interrupt controlling all the tasks in the system? If yes, which TT architecture can best match the application requirements and how are the external inputs going to be handled in a pure TT solution. The next section will describe the process of applying patterns and how the PMES collection can help in this migration process.

9.3.3.1 Applying patterns in the PMES collection

The problems associated with the ET design of the system have been identified and now it is worth looking at the pattern language map and thumbnails in order to select appropriate patterns. A pattern language represents a collection of patterns that work together. It is therefore important for the user to know how they can work together to solve a specific design problem. Generally in a pattern language, each pattern builds upon a pattern that came previously or another way of looking at this that each previous pattern creates the context for the following patterns. The dependency of patterns is shown through a pattern language map to represent how the patterns build upon and relate to each other. The textual description of the language provided and the names and the contents of each individual pattern itself can also help the user to visualize the process embedded in the patterns in order to help in applying them i.e. when, how and in which order? Figure 9-4 illustrates the process of applying patterns in the conversion of ET design to TT design.

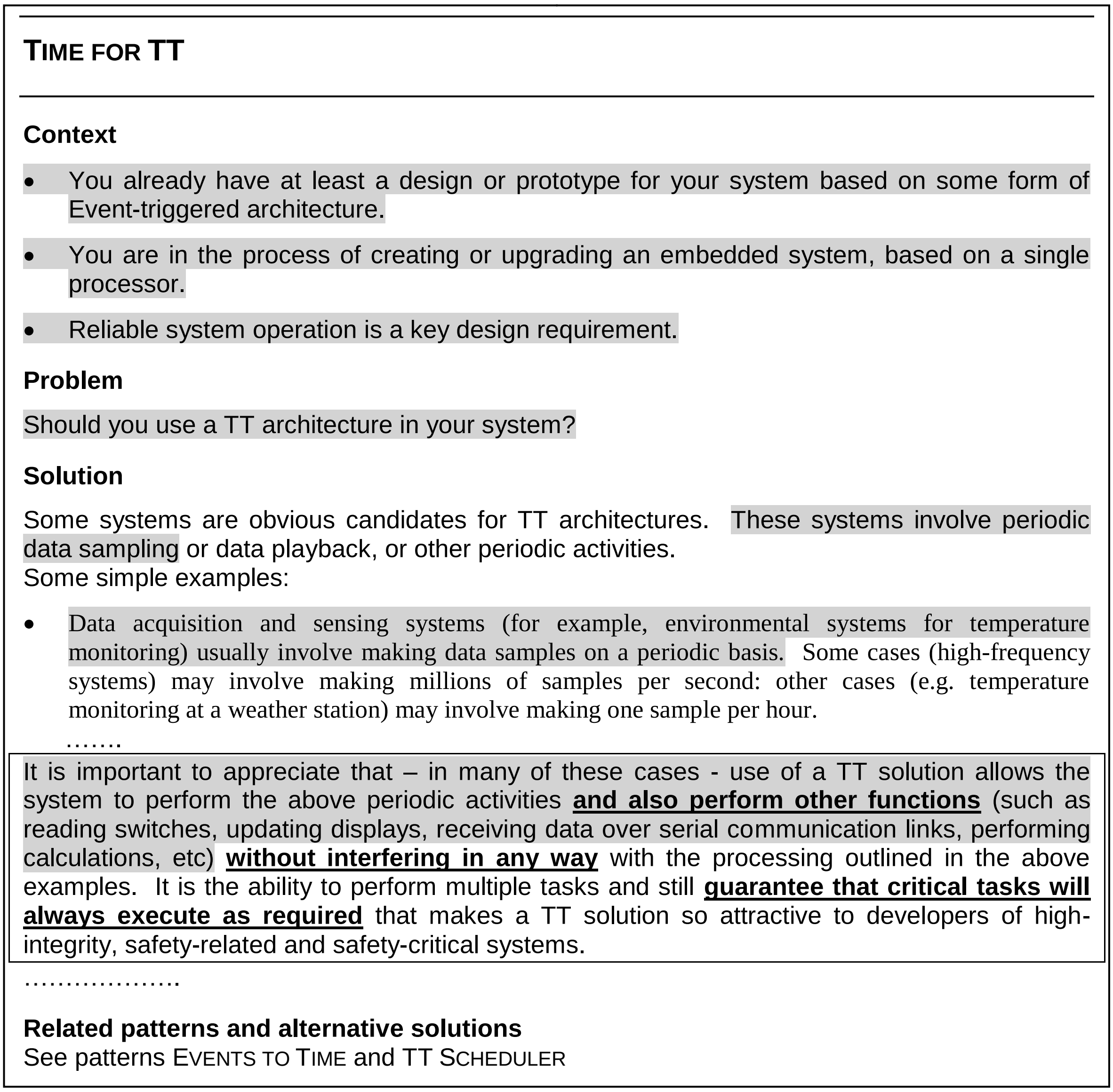


1. Identification of problems in the existing design
2. Selecting appropriate patterns that solve the problem.
3. Refine the design by implementing the solution given in the patterns.

Figure 9-4 Illustrating the process of applying patterns for the conversion of ET design to TT design

The different pattern forms adopted for pattern description can play an important role in pattern selection. For example most pattern forms have a section called ‘context’ which provides:

1. A description of the scope of the pattern in which it is applicable.
2. Gives information on earlier and later patterns that must be implemented before the current pattern can be implemented successfully to solve the problem completely. For example consider an excerpt from pattern TIME FOR TT and its highlighted sections in Figure 9-5.



TIME FOR TT

Context

- You already have at least a design or prototype for your system based on some form of Event-triggered architecture.
- You are in the process of creating or upgrading an embedded system, based on a single processor.
- Reliable system operation is a key design requirement.

Problem

Should you use a TT architecture in your system?

Solution

Some systems are obvious candidates for TT architectures. These systems involve periodic data sampling or data playback, or other periodic activities.
Some simple examples:

- Data acquisition and sensing systems (for example, environmental systems for temperature monitoring) usually involve making data samples on a periodic basis. Some cases (high-frequency systems) may involve making millions of samples per second: other cases (e.g. temperature monitoring at a weather station) may involve making one sample per hour.

…….

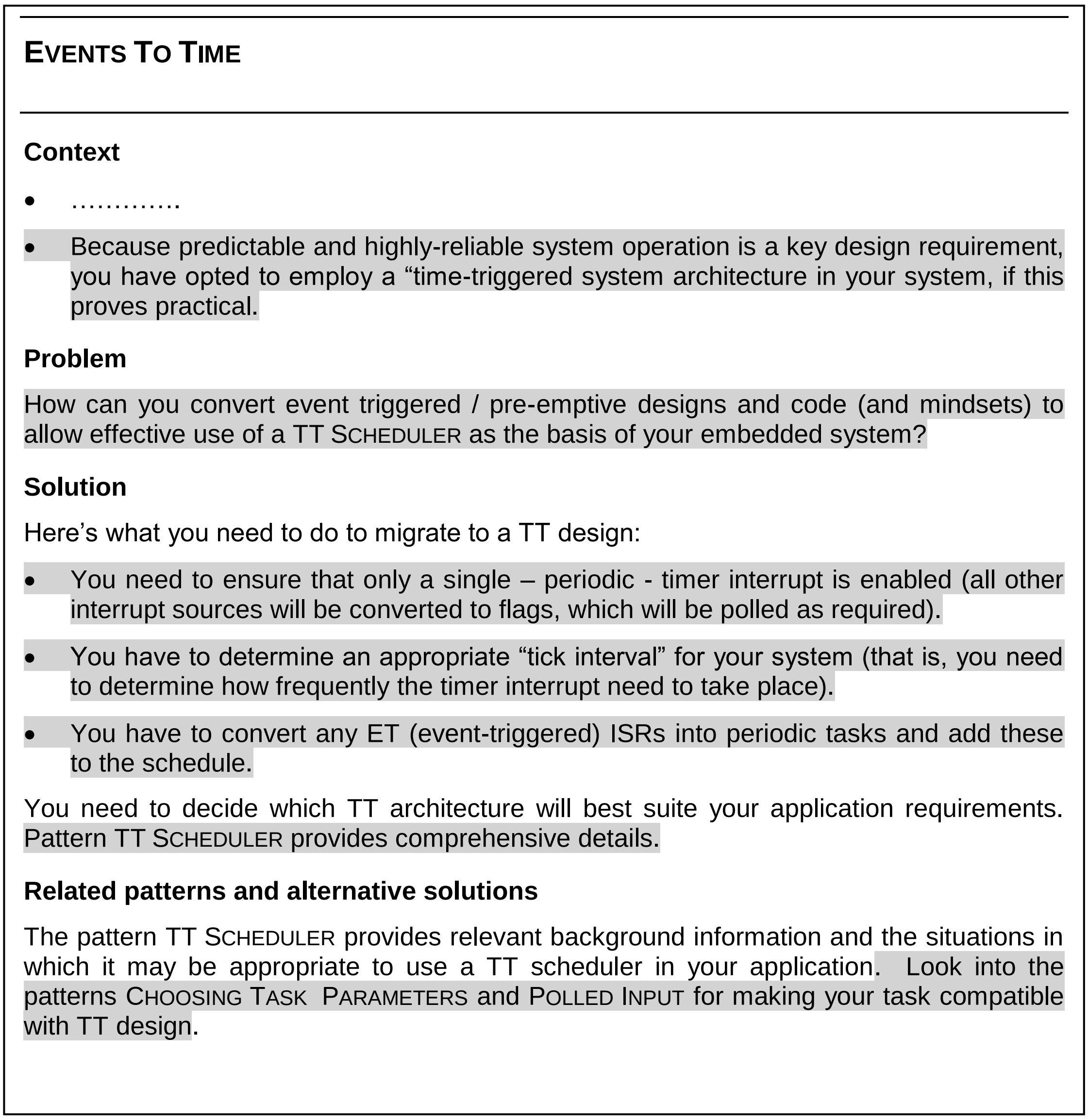
It is important to appreciate that – in many of these cases - use of a TT solution allows the system to perform the above periodic activities **and also perform other functions** (such as reading switches, updating displays, receiving data over serial communication links, performing calculations, etc) **without interfering in any way** with the processing outlined in the above examples. It is the ability to perform multiple tasks and still **guarantee that critical tasks will always execute as required** that makes a TT solution so attractive to developers of high-integrity, safety-related and safety-critical systems.

……………….

Related patterns and alternative solutions
See patterns EVENTS TO TIME and TT SCHEDULER

Figure 9-5 An excerpt from the pattern ‘TIME FOR TT?’ highlighted sections showing how the pattern is applicable in the context of DAQ for migration from ET to TT design

The highlighted sections indicate that as the reader reads through the pattern he/she is guided by how the pattern is applicable in the current context and what patterns are to follow next. As indicated in the Figure 9-5 above the next patterns to consider are EVENTS TO TIME and TT SCHEDULER. Now consider an excerpt from pattern EVENTS TO TIME in Figure 9-6:



EVENTS TO TIME

Context

- ………….
- Because predictable and highly-reliable system operation is a key design requirement, you have opted to employ a “time-triggered system architecture in your system, if this proves practical.

Problem

How can you convert event triggered / pre-emptive designs and code (and mindsets) to allow effective use of a TT SCHEDULER as the basis of your embedded system?

Solution

Here’s what you need to do to migrate to a TT design:

- You need to ensure that only a single – periodic - timer interrupt is enabled (all other interrupt sources will be converted to flags, which will be polled as required).
- You have to determine an appropriate “tick interval” for your system (that is, you need to determine how frequently the timer interrupt need to take place).
- You have to convert any ET (event-triggered) ISRs into periodic tasks and add these to the schedule.

You need to decide which TT architecture will best suite your application requirements. Pattern TT SCHEDULER provides comprehensive details.

Related patterns and alternative solutions

The pattern TT SCHEDULER provides relevant background information and the situations in which it may be appropriate to use a TT scheduler in your application. Look into the patterns CHOOSING TASK PARAMETERS and POLLED INPUT for making your task compatible with TT design.

Figure 9-6 An excerpt from pattern ‘EVENTS TO TIME’ with highlighted sections indicating the relevance of the pattern in the context of getting a TT design for the DAQ

For the application discussed in this case study, CO-OPERATIVE SCHEDULER (discussed previously as TTC scheduler in Section 4.3.2) provides a good TT solution. Before selecting the architecture, it is also important to know about the WCET of the tasks as it helps determining if the system will involve any level of pre-emption? As per the system specifications of DAQ, there is no long task in the system and so there is no requirement of task pre-emption, which makes it easier to decide on CO-OPERATIVE SCHEDULER as the architecture of choice. Therefore one possible sequence of patterns to follow in converting DAQ from ET design to TT design is depicted in Figure 9-7.

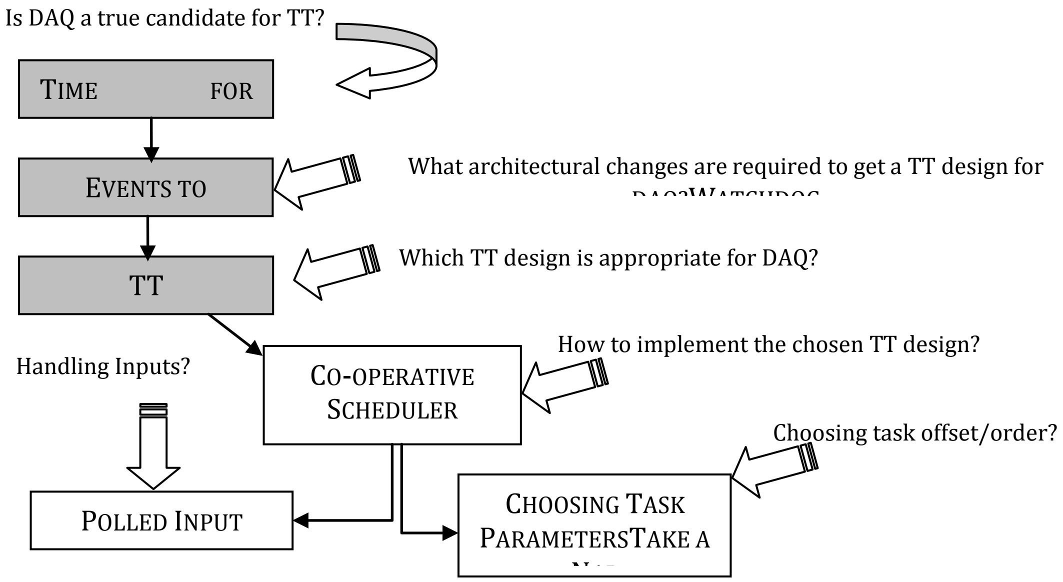

Is DAQ a true candidate for TT?
TIME FOR
EVENTS TO
What architectural changes are required to get a TT design for
Which TT design is appropriate for DAQ?
TT
How to implement the chosen TT design?
Handling Inputs?
CO-OPERATIVE SCHEDULER
Choosing task offset/order?
POLLED INPUT
CHOOSING TASK PARAMETERSTAKE A

Figure 9-7 A possible sequence of patterns to follow in achieving a TT design for the DAQ system

An equivalent time-triggered design is achieved with a single timer interrupt controlling all the tasks. Tasks communicate with each other using global variables. A high level representation of the system at task level is shown in Figure 9-8 and task specifications are shown in Table 9-3.

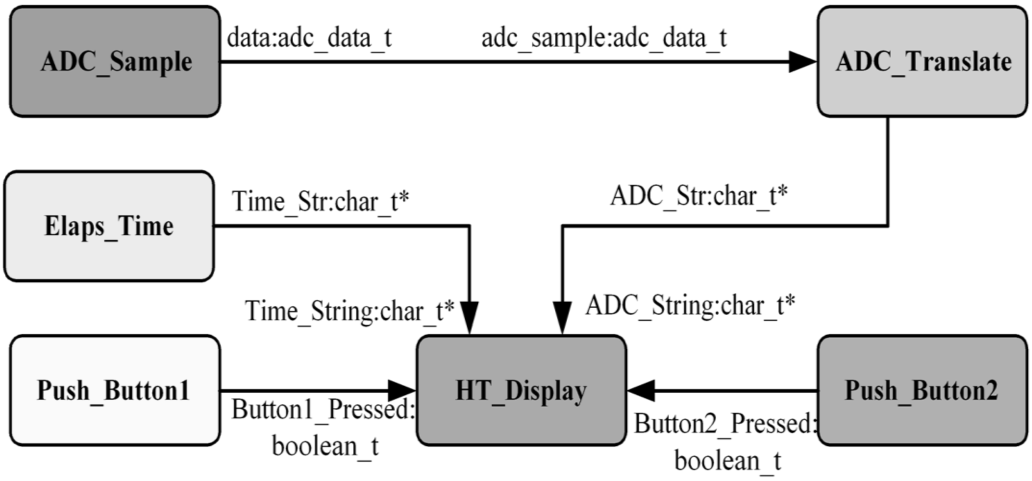


Figure 9-8 High level task representation of TT design for DAQ system

Table 9-3 Task specifications of TTC design for DAQ system

Task ID	Task Description	Delay (ms)	Period (ms)
ADC_Sample	Reads the voltage from a particular ADC channel.	0	5
ADC_Translate	Translates the ADC data to an appropriate string of characters.	2	10
Push_Button1	Polling task for button 1 to display ADC data	2	10
Push_Button2	Polling task for button 2 to display Elapsed time.	2	10
HT_Display	Displays the appropriate data on hyper terminal when the user presses a button otherwise display the message ‘Press a Button’.	6	1000
Elapse_Time	Calculates the elapsed time since the microcontroller was last reset.	8	1000
Flashing_LED	Periodic flashing of an LED to indicate that the system is alive.	10	1000

9.3.4. ET versus TT design

Performance measures listed in Equation 9-1to 9-6 are calculated both for the ET design and the TT design. Results obtained for both ET and TT designs are compared in Table 9-4. A graphical comparison and an analysis of results will be presented in this section.

Table 9-4 Comparison of ET and TT design for DAQ system

Parameters	ET design	TT design
CPU utilization	67.86%	72.12%
Idle time	32.14%	27.88%
Jitter (NBP)	0.05µsecs	0.05µsecs
Jitter (VBP)	0.3,1.25,1.45,1.75µsecs	0.05µsecs
Power consumption	0.258 watts	0.297 watts
Stack	3616 bytes	1544 bytes
RAM	6554 bytes	4260 bytes

- **CPU Utilization comparison:** The TT system showed 4.26% higher CPU utilization than the equivalent ET design. This is due to the inclusion of polling tasks scheduled to run every 10ms for the button press instead of the interrupt service routines which runs only when the external input arrives. These results are consistent with the example of alarm monitoring system discussed in Section 4.5 where it has been argued that the resource utilization of an ET system will be better than that of an equivalent TT system under low or average load conditions. This application was not tested for high load conditions as this was not within the scope of the aims and objectives of the study.

- **Power consumption comparison:** The ultimate result of using extra CPU resources results in more power consumption for the TT design as compared with the ET design. The TT design consumed 0.039 watts more power which is not too high in comparison. However, the results for CPU usage and power consumption for this application are more favourable to ET design.

The results for CPU utilization and power consumption measurements are shown in Figure 9-9.

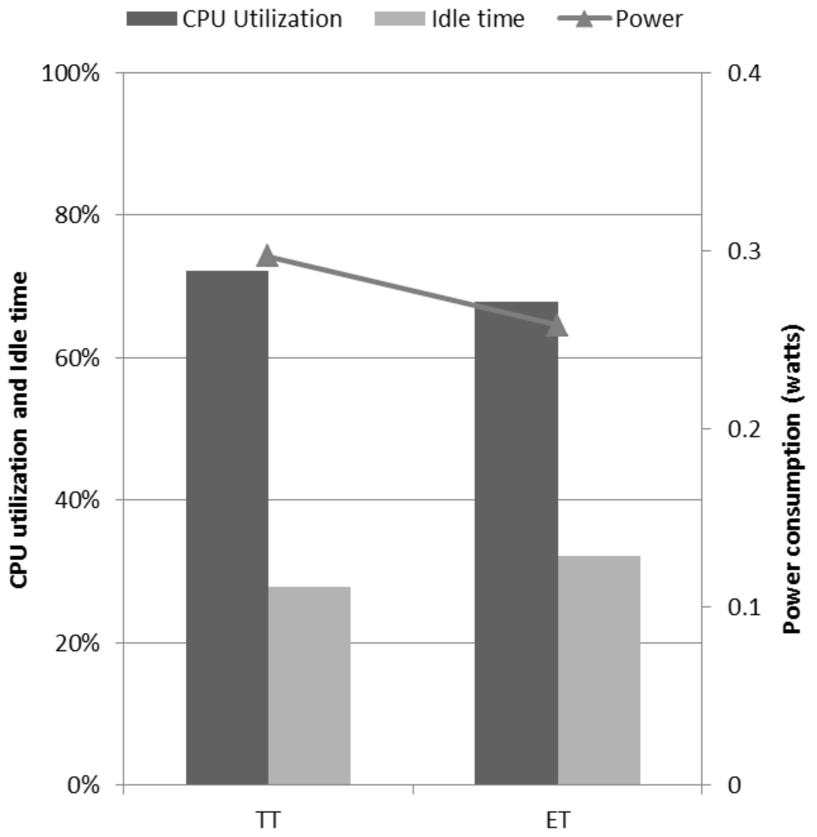


Figure 9-9 Comparison of CPU utilization, idle time and power consumption for ET and TT design for DAQ system

- **Jitter Comparison:** Comparison of variations in the release jitter of the sampling task showed results in favour of the TT design. The value of release jitter for the TT design remains fixed (0.05µsecs) no matter how many number of external inputs there were (see Figure 9-10). This is because the interrupts triggered by button press are replaced with polling tasks for each of the button presses. The periods of the polling tasks were set at the design time and these tasks always run at their pre-determined times unlike the unknown arrival and execution of interrupts discussed in Section 9.3.2. This helped in achieving more stable and predictable behaviour of the system.

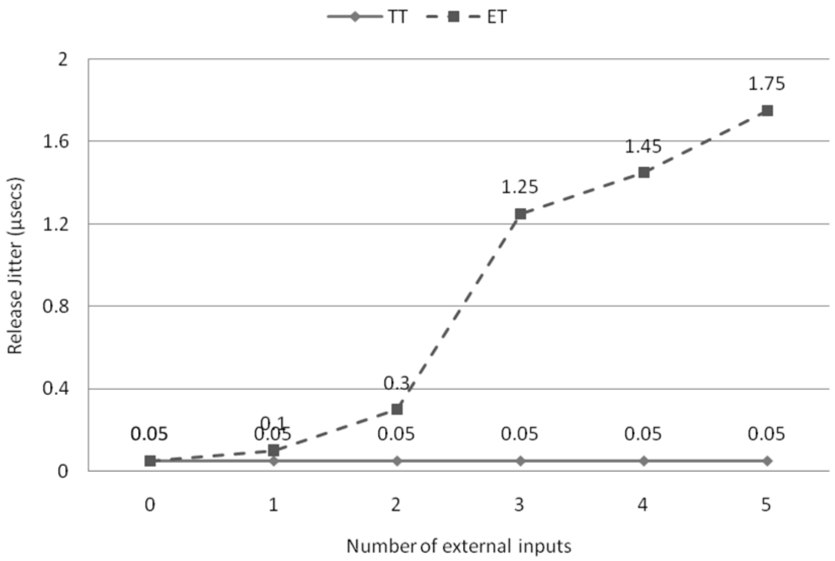


Figure 9-10 Release jitter in the data sampling task of ET and TT designs for DAQ system

- **Memory usage comparison:** Results showed higher memory utilization for the ET design as compared to the TT design shown in Figure 9-11. One of the main overheads associated with interrupt-driven systems is context switching which in turn requires memory space. As in the ET design multiple interrupts were made active, and each interrupt handler maintains its own stack space thereby reserving more memory. For handling interrupts on the ARM processor a vector table (a set of ARM instructions) is maintained that manipulates the program counter (PC). The instructions in the vector table direct the PC to jump to a specific location where the interrupt handler resides. When an interrupt occurs, a value or index of the table is calculated. The content of the table at this index (or offset) reflects the address of a service routine. The PC is initialized with this vector address and the execution begins at this location. Stack space is normally reserved at the design time with due care in order to avoid any stack overflow during the program execution. Reserving a separate stack is a better approach in the sense that if a single task is corrupted it could damage the whole stack.

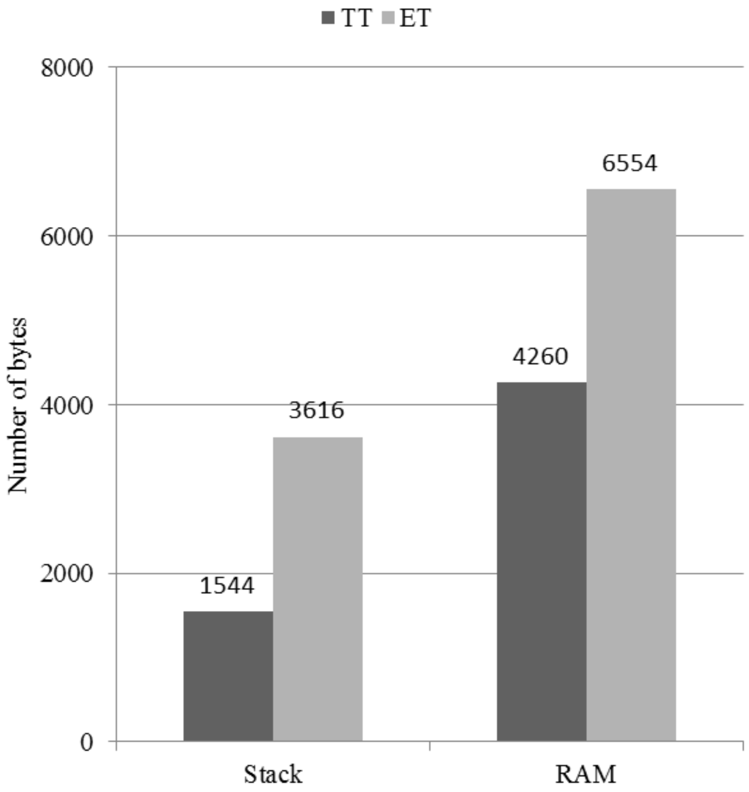


Figure 9-11 Memory and stack utilization of ET and TT design for DAQ system

All these complexities involved in managing interrupts leads to higher memory utilization for interrupt-driven systems.

9.4. Example 2: FFT/ADC framework

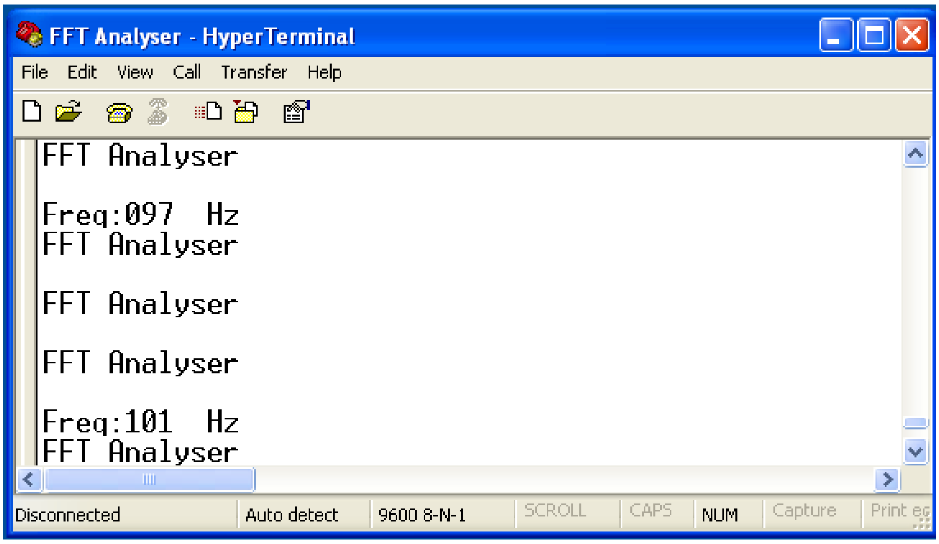
9.4.1. System functionality

The aim of this case study is to demonstrate the use of other TT architectures where TTC does not seem to provide an appropriate solution in migrating from ET to TT designs. The system discussed in this case study is based on two main activities:

- Data sampling using analogue-to-digital converter (ADC)
- Time-frequency conversion using the Fast Fourier Transform (FFT).

These two activities are commonly found in many embedded applications used for fault diagnosis and condition-monitoring. One of the requirements of such applications is to perform data sampling at high frequency (for example in some cases the system may require sampling data every 1 millisecond). The FFT processing is a long task typically requiring 256 or 512 data samples. This takes several milliseconds to complete it is though required to run less frequently (i.e. calling this task every 50 ticks for this system would be enough to perform FFT on a bunch of data samples).

The ‘FFT Analyser’ samples the generated signal, carries out a Fast Fourier Transform on the sampled data and finally outputs the first harmonic frequency to the hyper terminal when a user presses a push button available on the board. The rest of the time, the system keeps on displaying a message ‘FFT Analyser’. A sample output is shown in Figure 9-12.


FFT Analyser - HyperTerminal
File Edit View Call Transfer Help
Disconnected
Auto detect
9600 8-N-1
SCROLL
CAPS
NUM
Capture
Print ec

```
FFT Analyser

Freq:097  Hz
FFT Analyser

FFT Analyser

FFT Analyser

Freq:101  Hz
FFT Analyser
```

Figure 9-12 A sample output shown using hyper terminal from the FFT/ADC framework

9.4.2. Experimental setup

In the present case study the generic ADC/FFT framework is implemented as shown in Figure 9-13. The main application is running on an ARM7 NXP LPC2378-STK board and in addition to this, another Altera DE2-70 board is used which is pre-programmed as a function generator to generate frequencies in the range of 1 KHz.

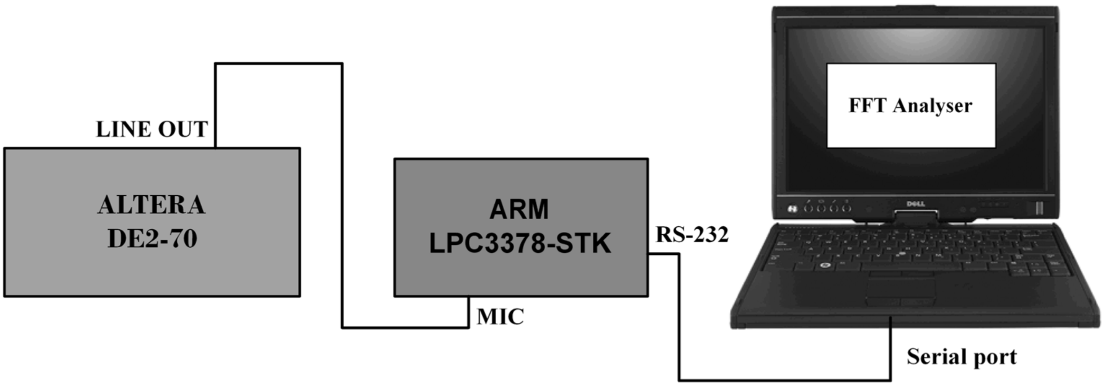


Figure 9-13 The 'FFT Analyser' frame work

The Altera DE2-70 development and education board is an excellent vehicle for learning about digital logic using FPGAs. However, for this study it is only used as a replacement of a function generator to generate signals of variable frequencies and for ease of use. More information about this development board is available online at (ALTERA, 2012).

One of the output lines ‘LINE OUT’ on the Altera board is connected with the microphone input (highlighted in Figure 8-1) of the LPC2378 board. The sampling task can access the microphone input by making use of the standard ADC driver[16]. On the LPC2378 development board the microphone input is connected to the ADC device 0, channel 2 and can therefore be accessed with the ADC device channel. A serial cable is connected between the PC and the ARM board to transmit data via an RS-232 link. Such a framework is commonly used in medical devices. An example of this is in an ECG machine for monitoring electrocardiograms in patients as they perform a series of exercises. In an industrial setting this framework is used for condition-monitoring of machinery in a factory (Schlindwein, Smith *et al.*, 1988).

9.4.3. Event-triggered pre-emptive design

The existing design of this system is based on a fully pre-emptive architecture (unlike the previous case study) in which all the tasks are pre-emptive in nature and are assigned with a priority at the design time. A task with higher priority which is ready to run can pre-empt a currently running task with lower priority. The tasks are periodic and controlled by a pre-emptive scheduler. The details on task specifications are shown in Table 9-5. Please note that

[16] RapidiTTy® set of tools (RapidiTTy, 2009) are used to develop implementation for these case studies. The standard driver for ADC is built-in the tool.

task priorities are defined in descending order with 4 being highest and 1 is the lowest in the present case.

In addition to the pre-emptive tasks, an external interrupt (EINT3) triggers on the button press ‘BUT 1’ (available on LPC2378 board) to display the first harmonic frequency.

Table 9-5 Task specifications of the periodic task in the ET design for FFT Analyser

Task ID	Description	Priority	Period (ms)
ADC_Sample	Sampling task which samples data.	4	1
FFT_Converter	Processing task running every 50ms which perform FFT of the sampled data to select the first harmonic frequency.	3	50
HRT_Display	Display task which prints first harmonic frequency on hyper terminal.	2	500
Flashing_LED	Periodic flashing of an LED to indicate that system is alive.	1	500

Problems associated with external interrupts have already been discussed in the previous case study. Generally speaking because pre-emptive architectures require task context switching, fully pre-emptive designs generally have both larger CPU overheads and RAM/ROM requirements than “equivalent” co-operative schedulers (Pont, 2001; Short, Pont *et al.*, 2008). As a consequence, it has also been argued that the timing properties of software code in non-pre-emptive designs are both easier to inspect and verify than the pre-emptive code. In addition, the increase in both CPU and inter-task communication overheads in a pre-emptive design will typically result in an increase in CPU utilization when a given system specification is implemented.

In the light of the above facts, there are strong grounds to migrate the ‘FFT/ADC framework to a suitable TT architecture.

9.4.4. Obtaining time-triggered design using patterns

It was previously stated in this thesis that the ideal choice is the TTC design when the decision has been made to switch to the TT architecture, but in certain circumstances it might not be possible to achieve the system requirements using the TTC design. As Jean Labrosse (2000) commented: *“The main drawback of this strategy is its latency to responding to important events: a higher priority task will have to wait until the currently running task finishes its execution”*(Labrosse, 2000). The current application is an example of this case as the sampling task is required to run every 1ms.

For the present case study it is clear that the FFT task (a long task) cannot be neatly decomposed into a sequence of shorter tasks and therefore it is not possible to employ a pure TTC architecture. The reason is that the FFT task will block the high priority ADC sampling task which has to run every 1 ms. The next immediate choice is a time-triggered hybrid (TTH) design which is guided by the patterns EVENTS TO TIME and TT SCHEDULER. The beauty of the TTH design is that it maintains the core co-operative design and allows the addition of a limited degree of pre-emption that is essential to meet the requirements of the application. This architecture is easy to implement and can operate with high reliability along with a controlled degree of pre-emption.

In the time-triggered implementation of this application, a single pre-emptive task is the ADC data sampling task. All the other tasks are made co-operative. The event triggered/interrupt driven button press call is translated into a periodic task using the pattern 'POLLED INPUT'. The order of patterns which can lead to a TT design of the FFT Analyser is shown in Figure 9-14. The tasks are then designed accordingly shown in Table 9-6.

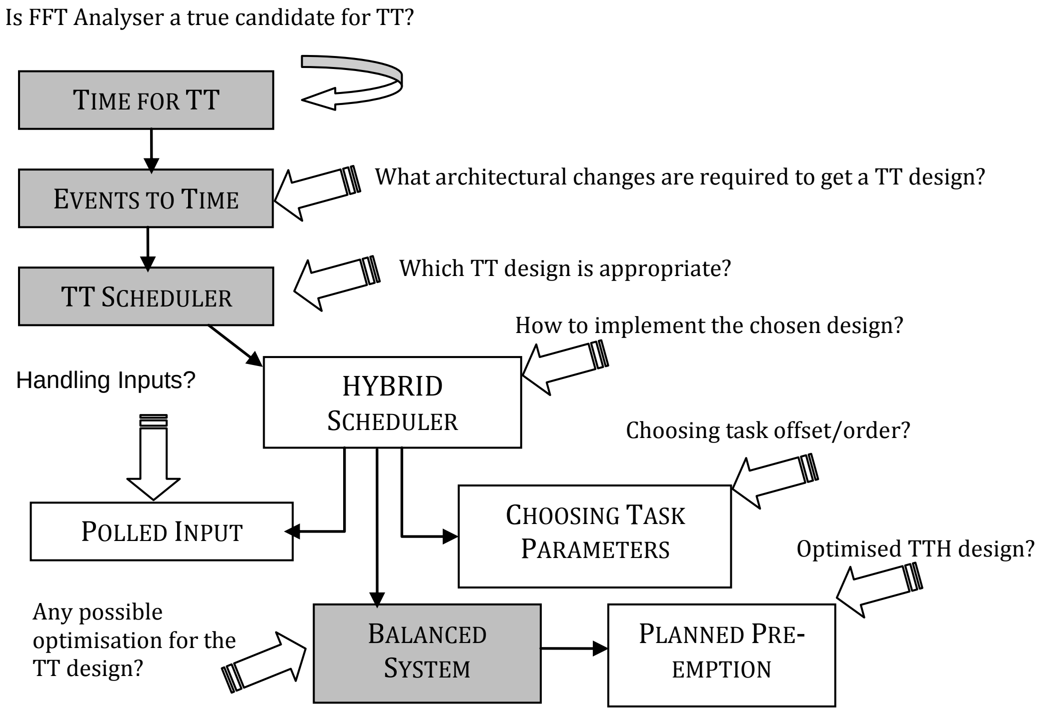

Is FFT Analyser a true candidate for TT?
TIME FOR TT
EVENTS TO TIME
What architectural changes are required to get a TT design?
TT SCHEDULER
Which TT design is appropriate?
How to implement the chosen design?
Handling Inputs?
HYBRID SCHEDULER
Choosing task offset/order?
POLLED INPUT
CHOOSING TASK PARAMETERS
Optimised TTH design?
Any possible optimisation for the TT design?
BALANCED SYSTEM
PLANNED PRE-EMPTION

Figure 9-14 One possible order of patterns to be followed which can lead to a TT design for the FFT Analyser

Table 9-6 Task specifications of TT design for FFT Analyser

Task ID	Task Description	Task type	Period (ms)
ADC_Sample	The sampling task which samples data at a rate of 1KHz.	Pre-emptive	1
FFT_Converter	The processing task running every 50ms which performs FFT of the sampled data to select the first harmonic frequency.	Co-operative	50
HRT_Display	The display task which prints the first harmonic frequency on the hyper terminal	Co-operative	500
Push_Button1	The polling task for the push button	Co-operative	25
Flashing_LED	Periodic flashing of an LED to indicate that the system is alive.	Co-operative	500

9.4.5. Comparative analysis of ET and TT designs

Measurements for CPU utilization, power and memory consumption are taken for both the designs and are tabulated in Table 9-7.

Table 9-7 Performance measures of ET and TT design for FFT Analyser

Parameters	ET design	TT design
CPU utilization (%)	73.85	65.32
Power consumption (watts)	0.3047	0.2695
RAM (bytes)	6881	3616
Stack (bytes)	4260	1056

It is important to mention again that to make a fair comparison equal numbers of external inputs are considered for both ET and TT designs. Unlike the previous case study, the ET design showed more CPU utilization because of the fully pre-emptive design.

The frequent pre-emption in the system incurs frequent context switching and so more CPU cycles were involved. In the TT design the level of pre-emption is reduced a great deal by making only a single task pre-emptive resulting in improved CPU utilization. Results for the power consumption clearly indicate less power usage in the TT design and these results are presented in Figure 9-15.

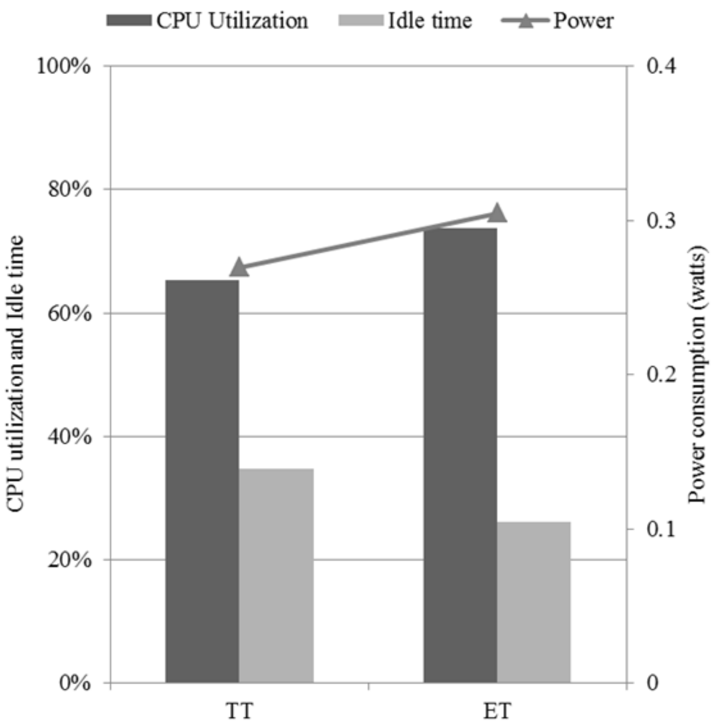


Figure 9-15 Comparison of CPU utilization, idle time and power consumption by ET and TT designs for the FFT Analyser

Also the TT design proved to be a better choice from the memory utilization point of view as shown in Figure 9-16.

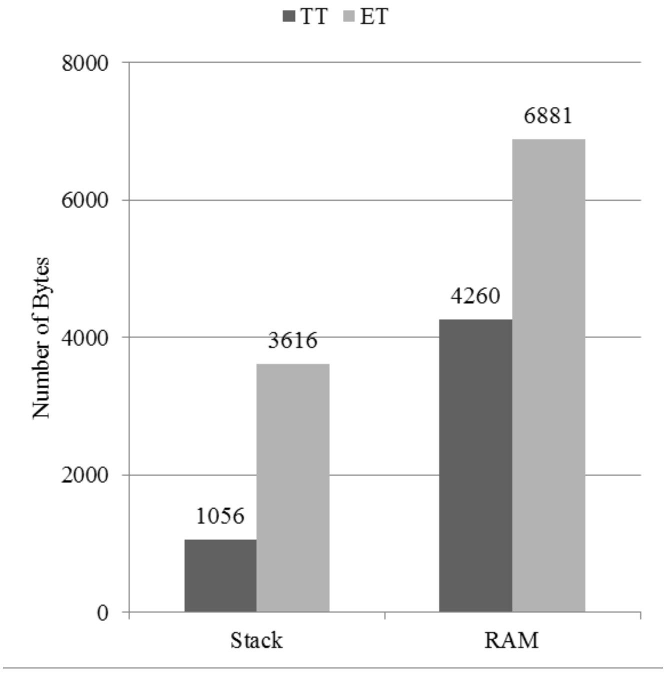


Figure 9-16 Comparison of memory utilization by the ET and TTH designs of the FFT Analyser

9.4.6. Optimisation of the TTH design

The results presented in the previous section is an indication that the TTH architecture may serve as a cost effective replacement for a fully pre-emptive design, however this architecture is still susceptible to jitter (Cottet and David, 1999). Similarly Jerri discussed the detrimental impact of jitter on applications such as spectrum analysis and filtering (Jerri, 1977). In control systems, jitter can greatly degrade the performance by varying the sampling period which leads to inaccuracy of results and data in the system.

In the PMES pattern collection, the pattern BALANCED SYSTEM provides ways to achieve optimised TT designs by reducing the levels of jitter wherever possible and the pattern PLANNED PRE-EMPTION has specifically addressed this issue in TTH based designs. In order to achieve the optimised TTH design for the FFT Analyser, PLANNED PRE-EMPTION technique is implemented as described in Section 8.4.11 and Listing 8-6 by using another timer called the ‘idle timer’ which places the processor into sleep mode just before the pre-emptive task starts (as shown in Figure 8-11). The optimised TTH design is named as TTH-PP (Time-Triggered Hybrid with Planned Pre-emption). Jitter values are measured for the sampling task in the TTH and the TTH-PP designs. Considerable improvements were observed in the TTH-PP design the results of which are shown in Table 9-8.

Table 9-8 Jitter comparison for TTH and TTH-PP designs

	TTH	TTH-PP
Jitter(µsecs)	1.5	0.1

9.5. Discussion

This chapter has demonstrated the effect of migrating existing applications based on the ET and/or pre-emptive designs to the TT designs. The demonstration process comprises two case studies of non-trivial embedded applications which are in common use. The examples demonstrate which aspects of the migration to the TT architecture can be of benefit to these applications. Example 1 showed the detrimental impact of migration to a purely time-driven application resulting in more CPU utilization and power consumption compared with the ET design. In contrast, the TT architecture provides more stability in the system by keeping the jitter values constant in all situations as compared to the ET design in which the number of external inputs causes variations in jitter. By looking at the difference of CPU utilization and power consumption between ET and TT designs it appeared that the TT design lead to 4.26% more of CPU utilization and 0.039 watts more power consumption. In other words these values represent the trade-off for a more predictable system.

Example 2 however showed a completely different scenario in which a completely pre-emptive architecture which is based on an ET design was migrated to a TT design by limiting the pre-emption to a single task only. The TTH design employed in this case not only reduced the level of pre-emption but also resulted in a reduction of CPU power and memory consumption. Further optimisation of this was evident in reduced release jitter for the sampling task in the system.

Both the case studies demonstrate that the PMES collection introduced in Chapter 8 can help in the migration process.

9.6. Conclusions

In conclusion the empirical results presented in the case studies in this chapter are indicating that for some applications migration to the TT architecture can help in improving the system reliability. As mentioned in the research premise in Section 1.2.1 this research aims to evaluate the effectiveness of patterns in the migration process and the case studies presented in this chapter are part of this evaluation process. These studies were conducted by the author herself and user trials involving a target group of embedded application developers/designers are required to effectively evaluate the proposed pattern collection. To this end the next chapter will present the details of empirical studies that were conducted with the involvement of subjects to investigate whether the PMES collection has the potential to help the practitioners during the process of migrating architectures for embedded applications.

CHAPTER 10. ASSESSING THE PATTERNS: EMPIRICAL STUDIES

10.1. Introduction

In CHAPTER 9, an effort was made to demonstrate the application of the proposed 'PMES' pattern collection on real embedded applications where migration to time-triggered designs makes sense. The results showed a considerable improvement in the performance and reliability of the tested applications after migrating to the time-triggered architecture. However, there is still a need to obtain feedback on these patterns from the target users for whom these patterns are documented – i.e. the developers and designers of embedded applications. This chapter will describe the investigations aimed at exploring whether or not the proposed pattern collection is providing the intended support to the developers of embedded applications.

This chapter is organized as follows: Section 10.2 and Section 10.3 present a literature review on different evaluation techniques adopted for design patterns. In Section 10.4 an overview of the preliminary pattern evaluation process employed in this project is presented. In Section 10.5 and 10.6 information is presented about the empirical studies conducted to evaluate the patterns developed during the research project. Two different experiments were designed and Section 10.7-10.8 describes each of these in detail. Section 10.9 presents a discussion on the results obtained and the chapter conclusions are presented in Section 10.10. Please note that the work described in this chapter have been published in (Lakhani and Pont, 2012b).

10.2. Evaluation of design patterns – an overview

Evaluation research can be described as an attempt to assess the worth or value of some innovation, service or approach (Robson, 2002). Evaluation or assessment is the means by which both quantitative and qualitative judgments can be made on something that aims to provide improvement over existing methods. There are many different approaches to evaluation, just as there are many different types of artefact, yet the purpose of evaluation remains consistent. It is almost meaningless to evaluate certain products without a purpose or without criteria for assessment. The goal of the evaluation is to answer questions such as, "Does the artefact or theory work?" and how useful is the artefact or theory?" The most important benefit of evaluation activity is that it offers feedback to the researcher in order to identify if the problem is well understood, if the assumptions are appropriate, if the quality of the design process is appropriate, and if further refinements are needed for the artefact (Hevner, March *et al.*, 2004).

New and existing patterns that have not been rigorously evaluated can raise many questions (Petter, Khazanchi *et al.*, 2010) about the value of the designs such as – is there a measureable time and resource saving? Are the patterns truly generic? Are they addressing the real problems faced by the practitioners working in a certain domain? Are they representing the knowledge gained by expert(s) in a way which is easily understandable and applicable?

10.2.1. Validation of patterns

A pattern describes a solution to a commonly recurring problem and one of its characteristic is that it has stood the test of time in a system that has seen frequent revisions. This implies that code written in 1970's that is still in use in a large system that has had lots of revisions can be considered a "proven solution". There is an issue however on how may argue that how to validate a "proven solution" as a pattern?

In the patterns community, it is an accepted 'requirement' that a pattern can be truly called a pattern only if it has been applied to a real-world solution at least three times. Brad Appleton (2000) calls this the "Patternity" test. To quote Linda Rising: *"A pattern is based on experience – it's not just a 'good idea' that someone thought of in the shower. Ideally there should be a section in the pattern that describes 'Known Uses' – instances of actual use. Ideally there should be three separate instances*"(Brandberg, 2005)*.*

This number three may well be arbitrary as some members of the pattern community propose a 'Rule of Two': in the words of John Vlissides, "We had one inviolable rule as we developed Design Patterns: we had to find two existing examples of a problem and its solution before we would write a pattern for it. This was a particularly important rule for us to follow, because we were exploring unfamiliar territory, and we wanted to make sure what we wrote was grounded in reality. We didn't want to end up with a set of solutions to problems no one had" (Vlissides, 1996). Clearly, testing a pattern in more than one example of a problem is necessary, and the more examples tested the better. The Writer's Workshop at PLoP conferences (discussed previously in Section 6.8) is a testing ground for an early stage of pattern validation, because during a workshop the readers of patterns cross-examine the pattern authors by asking for example "Are you sure this is a pattern? What are your examples?" or "This appears not to work well as it's written. What forces are missing?" Therefore to make a pattern valid the author has to make sure they do a thorough job of understanding the context, where the solution is valid, and the trade-offs involved in the solution. A well tested and well explained pattern can help the users to decide if the solution described in the pattern is valid and appropriate for their use? This is one way of assessing pattern validity though to some extent the pattern user has to have some faith in the pattern author.

All published patterns pass through a preliminary process of validation through the process of 'Shepherding' and the 'Writer's Workshop' and through the pattern author's own testing and experimentation process. For more rigorous validation empirical studies involving the target subjects need to be conducted. The next section will report on some of the studies of pattern validation found in the literature.

10.3. Related work

There has been lot of work carried out on the creation of design patterns in a wide range of diverse fields as discussed previously in Section 4.4 in this thesis. There has been limited emphasis however on developing guidelines for evaluating the validity of patterns (Khazanchi, Murphy *et al.*, 2008). This section will discuss the earlier work on the evaluation of design patterns and pattern languages.

Patterns for designing object-oriented software by GoF (Gamma, Helm *et al.*, 1995) have been the center of focus for many researchers since they were introduced in the mid 90's. Several studies have also been reported in the literature on the evaluation of these patterns. For example, Lutz Prechelt and Barbara Unger reported a series of experiments (Prechelt and Unger, 1998) to test certain claims such as 'design patterns improve communication from developers to maintainers by carefully documenting pattern usage in the code'. They also suggested that the unsuitable application of patterns may be harmful and warned against using them in a random 'cook book' fashion; that is to say that they need to be applied thoughtfully by applying common sense. Again Lutz Prechelt along with other colleagues also reported other experiments that concern the assessment of the usefulness of pattern documentation in program maintenance (Prechelt, Unger *et al.*, 2002). In their study, seventy-four student participants were involved in testing the question: 'Does it helps the maintainer if the design patterns in the program code are documented explicitly compared to a well-commented program without explicit reference to design patterns?' These explicit comments are some additional lines of comment called PCL (pattern comment lines) that describe the pattern usage where applicable. They found that PCL in a program considerably reduced the time required for a program change and helped in improving the quality of the change.

Chung and colleagues described the evaluation of a pattern language to help designers of ubiquitous computing[17] (Chung, Hong *et al.*, 2004). Their evaluation study was conducted in several rounds. In the first round, eighteen designers divided into nine pairs were involved including professionals as well as graduate students. All participants were given design tasks to solve problems related to ubiquitous computing such as the design of a location enhanced service to help customers in a shopping mall. Participants involved in the study were divided into two major groups. One group was given the proposed pattern collection to help them solve the exercises and the other group was asked to solve the exercises based on their professional experience and without making use of patterns. The exercises were assessed by Human-Computer Interface (HCI) graduate students and the results indicated that:

- Patterns helped novice designers.
- Patterns helped designers to solve problems in the domains for which they were unfamiliar.
- Patterns helped designers communicate design ideas amongst themselves.
- Patterns helped designers avoid some design problems.
- Patterns did not help with the privacy related issues for websites.

Based on their observations of the first round of evaluation the authors edited the contents of their proposed patterns and conducted a second round of evaluation. However in their second round they reported the participants still

[17] Ubiquitous computing is the use of technologies to accomplish both simple and complex tasks throughout our work and personal lives. Ubiquitous computing systems are embedded in the environment or carried on the body and adapt to the natural interactions of people. Ubiquitous computing is not intended as a description of the ubiquity of computer devices themselves; it's a description of how computing fits in our lives ubiquitously (Begole, 2011).

failed to take advantage of the privacy patterns because the privacy issue was not emphasised in the higher level patterns sufficiently.

Another example of a pattern evaluation study is by Aras (2005) based on a project for a scientific application for three-dimensional modeling. This application called COMPUCELL3D (Aras, 2005) is a software framework for a three-dimensional simulation of morphogenesis[18]. The evaluation methodology adopted for this application did not involve any participants. It consisted of a comparison of performance measures (speed and memory consumption) and maintainability for different versions of COMPUCELL3D designed with the use of a combination of design patterns described as FACTORY, STRATEGY and SINGLETON (Gamma, Helm *et al.*, 1995) and without patterns (simple C Structures). Altogether, there were seven versions designed, four with the use of patterns and three without design patterns (Aras, Cicovski *et al.*, 2005). The results obtained demonstrated that as the application was re-factored to support additional functionality without using design patterns there was performance degradation such as extra memory utilization. The reason for this is that implementations with design patterns avoid the need for some additional code (by instantiating the appropriate algorithm at run time, which can be referenced throughout the simulation unlike C structures).

Another interesting evaluation study reported by Ayavoo (Ayavoo, 2006) related to a tool called 'PTTES builder' (Mwelwa, Pont *et al.*, 2006) designed

[18] Morphogenesis is the structural development of an organism and its organs involving cell differentiation growth and migration, bulk changes in tissue shape, secretion, and the diffusion of extra cellular materials, for example proteins (Aras, 2005).

to support an automated pattern-based code generation technique. An example from this work is one of the series of studies conducted to test a technique called ‘SGM’ (Small Group Methodology). SGM is used to conduct empirical studies of software engineering with a small number of volunteer participants with similar skills and well- matched academic backgrounds. This study involved a comparison between two different development approaches of a non-trivial embedded application called ‘CCS’ (Cruise Control System). One used the PTTES builder and the other used a manual approach (i.e. using a pattern collection in a reference book). Altogether eight participants (split into four groups) were involved in the study. The results showed that the groups that used the PTTES builder worked more efficiently and required less effort to complete the design compared with groups that did not use the PTTES builder tool.

Another recent research study concerning the evaluation of design patterns for multi-core embedded systems is reported by Strebelow and Prehofer (Strebelow and Prehofer, 2011). A multi-core processor is a single computing component with two or more independent actual processors (called “cores”), which are the units that read and execute program instructions (Source: Wikipedia). The aim of this study was to evaluate four software design patterns: Half-Sync/Half-Async, Leader/Follower, Proactor and Reactor (Schmidt, Stal *et al.*, 2000) in the design of efficient message processing for a large number of input streams on several cores. The authors included a basic multi-threading solution in the evaluation to compare patterns with simple but commonly used solutions. For this study, they used a specific hardware – the

Cavium Octeon cn5650 multicore system[19] running applications designed with pattern implementations. Each pattern implementation was run and the throughput (in terms of messages per second) and average latency was measured and compared with the designs implemented without the use of patterns. Based on the results obtained they made the following conclusions: The basic multi-threading solution performed best followed by Leader/Followers and Half-Sync/Half-Async. The Proactor pattern showed inefficiency in systems because of massive thread creation. The system implemented using the patterns Half-Sync/Half-Async systems suffered increased latency due to the inefficient event de-multiplexing. The main conclusion of their evaluation study was 'The use of design patterns does not always provide the best solutions.'

One important observation from the example studies quoted above is that there is no universally accepted evaluation criteria for design patterns. Aware of the complex issues of evaluation, Khazanchi and colleagues have suggested some general guidelines for evaluating patterns in the Information Systems (IS) domain (Khazanchi, Murphy *et al.*, 2008). They supported Alexander's idea of 'Quality Without A Name' (QWAN) to make patterns useful and beneficial for the intended users – however they believe that – QWAN is difficult to describe. They define a few general qualities associated with patterns that could help to achieve QWAN and could provide guidance in evaluating patterns. They name these qualities as plausibility, feasibility,

[19] This system is designated for embedded telecommunication devices as found in base stations, routers etc. The Octeon provides 12cnMIPS cores operating at 800MHz. Each core has 32/16 KB of L1 cache and all cores share a 2 MB L2 cache. More information is available on www.caviumnetworks.com

effectiveness, pragmatic, empirical and predictive. Their research was further expanded by Petter *et al* (2010) and resulted in a framework for the evaluation of design patterns based on 'Design science[20]' . This proposed framework is based on the belief that the evaluation of patterns should continue throughout the life-cycle as a continuous activity which brings improvement in the patterns from time to time as depicted in Figure 10-1.

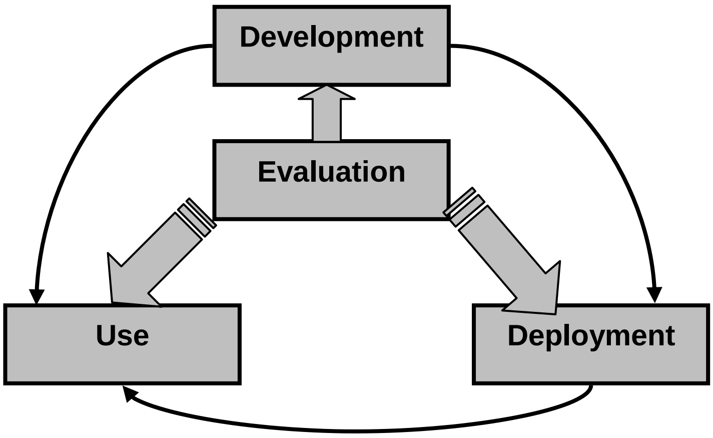


Figure 10-1 Pattern life cycle [Adapted from (Petter et al, 2010)]

According to these authors, writing patterns of any form is susceptible to human fallibility and biases. The evaluation of patterns however provides an opportunity to test if a pattern is actually applicable and useful for the intended users. The evaluation comprised deploying the pattern in the specific domain for which it exists, and then carefully analyzing the results against intended effects described in the pattern definition itself. The qualities previously defined by Khazanchi and colleagues to achieve QWAN are now formally

[20] Design science research in information systems creates and evaluates IT (Information Technology) artefacts intended to solve identified organisational problems (Hevner, March *et al.*, 2004).

presented with their traditional definitions, and adapted definitions for patterns are shown in Table 10-1.

Table 10-1 Criteria for evaluating patterns [Adapted from (Petter et al, 2010)]

Evaluation Criteria	Traditional Definition	Adapted Definition for Patterns
Plausible	The degree to which a concept is more than just a belief (Sproull, 1995; Khazanchi, 1996).	The pattern is sensible considering the current understanding of the domain (Alexander, 1979; Brown, Malveau *et al.*, 1998; Khazanchi and Zigrus, 2007).
Effective	The degree to which a concept describes the phenomenon under study parsimoniously and stimulates inquiry (Khazanchi, 1996).	The pattern is described in language that is understandable. Root causes of the problem are identified and addressed by the recommended solutions (Appleton, 2000).
Feasible	The degree to which a concept is workable or operational (Khazanchi, 1996).	The pattern can be operational or implemented as described.
Predictive	The degree to which a concept is capable of predicting outcomes for given conditions (Sproull, 1995; Khazanchi, 1996).	The pattern produces the expected result or produces a result in the intended direction (Coplien, 2007).
Reliable	The degree to which a concept is certifiable by different researchers using different methods (Straub, 1989; Khazanchi, 1996).	The pattern produces similar results regardless of the implementer or technique.

These guidelines are intended for pattern authors to write quality patterns and are obviously useful; nevertheless the authors are unable to provide a strong framework and methodology for conducting the evaluation process itself.

Thus, the lessons learned from this review of evaluation studies are:

- Evaluation studies are essential and incur additional cost and time.
- Evaluation studies are highly dependent on the characteristics of the domain for which the patterns are designed.

- Evaluation studies can be conducted with or without the involvement of participants.
- Useful results can be obtained from a small number of participants.
- Participants involved in such studies can work individually or in groups.
- The task designed to be performed by participants may or may not involve programming exercises.
- An evaluation study can span several rounds or be completed in a single sitting.

In the light of these observations, evaluation studies for the PMES collection were designed and these are discussed in the next section.

10.4. Preliminary evaluation of the PMES collection

There is a common understanding within the pattern community that feedback on the use of the patterns in different applications is necessary to improve and validate the patterns. Feedback can be provided by those that have implemented the patterns or by experts that have read the patterns (Brown, Malveau *et al.*, 1998). On this latter point, it was previously mentioned in Section 10.4 that most of the patterns in the proposed collection have already passed through the rigorous process of *'Shepherding'* and the *'Writer's Workshop'* at various PLoP conferences. The pattern collection published during the early stage of this research including the patterns EVENTS TO TIME, BUFFERED OUTPUT and POLLED INPUT (Lakhani, Das *et al.*, 2009a) has been shepherded by Robert Hanmer for the conference EuroPLoP 2009. Robert Hanmer is a renowned name in the pattern community and is the author of a book on patterns (Hanmer, 2007) and has published numerous papers on this topic. The pattern collection published at a later stage including the patterns BALANCED SYSTEM, SINGLE PATH DELAY, SANDWICH DELAY, TAKE A NAP and PLANNED PRE-EMPTION (Lakhani, Das *et al.*, 2010c) were shepherded by Jorge L. Ortega-Arjona. Jorge authored two recent books on patterns (Ortega-Arjona, 2009; Ortega-Arjona, 2010) and again is widely published in this field. Patterns documented by fellow colleagues in the Embedded Systems Research Group (ESRG) which are included in the PMES collection include the patterns CRITICAL SECTION, RESOURCE LOCK, DISABLE TIMER INTERRUPT, PRIORITY CEILING PROTOCOL, IMPROVED PRIORITY CEILING PROTOCOL (Wang, Pont *et al.*, 2007) and the pattern TT SCHEDULER (Pont, Kurian *et al.*, 2008) has also passed the same pattern refining process. Therefore, most of the patterns have been assessed by experts, though a few still need further testing such as the patterns CHOOSING TASK PARAMETERS, SYSTEM MONITORS and TASK GUARDIAN. A major part of the evaluation still remains and that is the feedback from a target user group. In order to fill this gap, empirical studies with the involvement of end users are vital to prove the efficacy of the proposed pattern collection. This chapter will provide the details of these studies conducted during the research.

10.5. Empirical studies for the evaluation of the PMES collection

10.5.1. Methodological considerations and practical constraints

In order to obtain the maximum possible benefits from the evaluation process for the PMES collection the following practical considerations were necessary:

- Available funds to conduct the studies
- Available time to conduct the research
- Availability of the appropriate resources

The key ingredients in designing empirical studies for design patterns are: the principal research approach (controlled experiment versus field study), the background of the subjects (students versus professionals), the type of software work investigated (design, development, individual or team work etc.) and the methodology of the study (Prechelt and Unger, 1998). The proposed collection of patterns for this study as stated earlier is intended to support developers and designers of TT and ET based embedded applications. To evaluate the patterns ideally they should be tested ‘in the field’ in real embedded software development environments where real practitioners are facing the migration challenge with some ‘live’ projects. However, because of the practical considerations related to funds, time and resources (mentioned above) under which these studies were designed such an ideal situation was not practical within this study.

Given the relatively easy access to students in this research environment, MSc (Engineering) and MSc (Computer Science) students at the University of

Leicester were selected as subjects. Fortunately, as mentioned in Section 10.3 the 'SGM' technique of conducting empirical studies is quite effective with small number of volunteer participants with well-matched backgrounds (Ayavoo, 2006; Mwelwa, 2006). Therefore 'SGM' has been adapted for the studies presented in this chapter. The core of the SGM has the following main stages:

1. Study preparation
2. Experimental investigation
3. Analysis of results

Further actions within each stage are illustrated in Figure 10-2. The study preparation includes the selection and management of subjects and the preparation of the exercise used in the study to evaluate the artefact. During the actual experiment, a test/assessment is conducted based on the exercise. After the experiment the subjects are given questionnaires to obtain feedback or they can be interviewed on an individual basis if required. In the analysis stage, exercise sheets solved by the subjects are assessed and finally results are analysed.

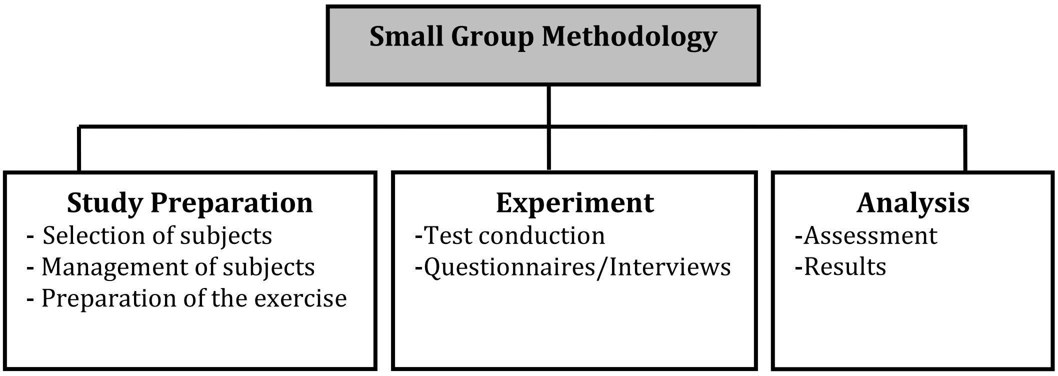


Figure 10-2 An overview of the SGM technique

One way to improve the SGM would be to employ a “blind” approach (Kitchenham, Pfleeger *et al.*, 2002) to analyse the results. In this case, the individual who assess/analyses the results from the study is kept unaware as to which set of results have been subjected to a treatment. The results obtained this way are fair and unbiased.

The SGM was employed in these studies to evaluate the PMES collection of patterns described in CHAPTER 8. Some amendments were essential to make the studies more robust and useful and these will be explained in the experiments described in the following sections.

10.6. Experiment planning

10.6.1. Goals

The goal of the studies described here is to explore two specific hypotheses, by means of empirical studies. These hypotheses are as follows:

1. Patterns in the PMES collection help experienced developers to choose appropriate TT solutions during the migration process.
2. The PMES form helps the pattern users to apply the information presented in a more effective way than “traditional” information sources.

10.6.2. Design of experiments

The key ingredients in designing empirical studies for design patterns are: the principal research approach (controlled experiment versus field study), the background of the subjects (students versus professionals), the type of software work investigated (design, development, individual/team work etc) and the technical conduct of the study (Prechelt and Unger, 1998). The proposed collection of patterns for this study as stated earlier is intended to support developers and designers of TT and ET based embedded applications. However, because of the practical considerations (mentioned in Section 10.5.1) under which these studies were designed an ideal situation where patterns can be tested with a ‘live’ industrial project was avoided as industrial projects usually takes years to complete and companies have their own reservations about sharing any information while a project is in process. Given these limitations the next available option was to arrange for some controlled laboratory experiments which could be managed conveniently and these were chosen for the studies discussed here. However, the experiments conducted were designed and planned according to the recommendations for designing empirical studies in software engineering such as those described in (Juristo and Moreno, 2001) which also provides useful input for further research in the industrial context.

10.6.3. Selection of subjects

The next important thing was to decide on the subjects i.e. the class of users for which such a collection can provide support. Again, ideally this should include subjects from a broad cross-section of professionals from the embedded software community such as software architects, system developers and designers and programmers but the same issue of funding and time constraints obviated this. An alternative is to use students as subjects and studies for example (Carver, Jaccheri *et al.*, 2003; Arisholm and Sjoberg, 2004) highlighted the issues of using students as subjects rather than professionals working in the industry. These issues relate to the benefits that researchers gain from empirical studies with students such as obtaining preliminary evidence to confirm or refute hypotheses (why are students better in this respect) and benefits to students such as getting better insights on specific industrial problems. Interestingly, the studies encourage the use of students as subjects particularly when some of the students had the same level of skill as the professionals. Therefore many studies published in the literature such as (Unger and Tichy, 2000; Prechelt, Unger *et al.*, 2002; Chung, Hong *et al.*, 2004) used students as subjects for pattern evaluation.

Within the limited funding constraints it was not possible to involve a huge number of participants in the study as it is customary to pay participants for their time in being subjects for research and the accepted rate was £20.00 per participant. Therefore, for the studies reported in this chapter, it was necessary to select a small “balanced” group of subjects with equal background and capabilities. This is also recommended in previous studies such as Jazequel, Train *et al* (2000) and Gear (1973). To do this, an account was taken of the subject’s prior experience and the courses they had undertaken. Moreover, given the relatively easy access to students in this research environment, MSc (Engineering) and MSc (Computer Science) students at the University of Leicester were selected as subjects.

10.6.4. Task design and assessment methodology

For empirical studies of software engineering there are several dimensions in which the task performed by the experimental subjects may differ: it may be a design or implementation task, from scratch or in maintenance, may be done alone or by a team, it may target programs of different size and from different domains, and it may employ different types of design patterns (Prechelt and

Unger, 1998). For the studies reported here tasks were designed such that the usability of patterns can be assessed to their maximum. Two aspects were considered in the study:

1. The uniqueness of the design problem that pattern has addressed
2. The pattern form.

It was decided not to use any tasks involved coding, rather the exercises designed were completely analytical and the aim was to test how patterns in the PMES collection can help designers in taking the appropriate decisions during the migration process.

For a fair and unbiased assessment of the exercises the “blind” methodology discussed in (Kitchenham, Pfleeger *et al.*, 2002) was adopted. In this case a neutral person or a third party which has no direct involvement in the research is hired for the assessment. In this case the third party who analyses the results is kept unaware as to which sets of results have been subjected to a treatment. For the studies discussed in this paper, some senior members of the research group helped in this regard by offering their time for grading the exercises solved by the subjects. The assessors were acting as a third party and were not told about the aims of the study, neither were they told about which exercise sheets were solved with or without using patterns. After the results had been obtained, the data analysis was done by the experimenter himself.

The next two sections provide full details about the experiments.

10.7. Experiment 1: Patterns for experienced developers

10.7.1. Aims and motives

Experienced developers of embedded systems are usually adept in designing applications from scratch. They are also usually confident about the systems which they have designed with the architectures and tools at their disposal. However, it can be quite a challenge for experienced developers to transform an existing architecture to another especially when they are not too familiar with the architecture that they wish to transform. This first experiment was designed with this aforementioned situation in mind and investigated the following hypothesis:

Hypothesis H1: Patterns in the PMES collection can help experienced developers to choose appropriate TT solutions during the migration process.

10.7.2. Preparation of the exercise

An exercise was prepared based on the details of three different embedded applications that were designed in the ET architecture and needed to be migrated to the TT architecture. The exercise was completely analytical with no coding requirements and the applications given in the exercise are summarised below:

- **Traffic Control System (TCS):** An application based on the controller for the traffic lights and pedestrian crossing lights used at a typical crossroads in the UK.

- **Fast Fourier Transform on ADC samples (FFT/ADC):** This application is based on two main activities: data sampling using ADC and time-frequency conversion using a Fast-Fourier Transform (FFT).
- **Data Acquisition System (DAQ):** A control application used to sample data from an analogue-to-digital converter (ADC). It translates the sampled value to an appropriate string of characters and displays this value and the elapsed time since the microcontroller was last reset, on to the screen on user request.

The author herself has already worked on the design and development of all the above applications and was fully aware of the design constraints involved in each application. All the applications differed in their design; difficulty level and complexity. Here complexity is measured in terms of the number of lines of code (LOC) in each system (see Table 10-2). The reason for choosing systems of varied complexity is to test the usability of patterns more broadly.

Table 10-2 Difficulty levels of systems used in exercise for experiment 1

Application	Lines of Code	Complexity
TCS	3400	high
ADC/FFT	1509	Medium
DAQ	600	Low

10.7.3. Management of subjects

For this experiment, it was decided to involve MSc(Engineering) students who had already taken a 10 week, one-semester module in Programming Embedded Systems (PES-I), and were doing the extended module PES-II during this study. Therefore, the subjects involved in this experiment had sound intermediary knowledge and prior experience of design and programming embedded applications. An email was circulated to the group of students enrolled on the PES-II module inviting them to participate in the research study. The subjects were not told about the actual theme and purpose of the study to keep the results unbiased, and this also helped to eliminate the possibility of the subjects' behaviour being influenced by knowledge of the experimenter's expectation. This is known as the Hawthorne Effect in the literature (Kitchenham, Pfleeger *et al.*, 2002; Berry and Tichy, 2003). The test subjects then were only informed of the study's objectives and motives at the end of the experiment.

From a list of fifteen volunteers, eight subjects were selected for the study who's mark history in PES-I was in 90-70% range. The participants were organised as PG (Pattern Group i.e. the group members were given the PMES pattern collection during the exercise) and NPG (Non-Pattern Group i.e. the group members were not given with the PMES pattern collection during the exercise). The top scoring members for the PES-1 module were put in pairs PG1 and NPG1 and the four who scored around the 70% mark were also put in pairs PG2 and NPG2 as shown in Table 10-3.

Table 10-3 Group structure for Experiment 1

Group ID	Members	Patterns	Marks in PES-I
PG1	2	√	in 90% range
NPG1	2	×	
PG2	2	√	in 70% range
NPG2	2	×	

The NPG were expected to use their prior experience in embedded system design to solve the exercises during the study and the PG groups were provided with the PMES collection of patterns.

The experiment was conducted in the Electrical Teaching Laboratory (ETL) of the Engineering Department in the University of Leicester. All of the groups were called to attend the study on the same day and were seated far apart from each other (to reduce the possibility of interaction among groups during the study). During the study, all the groups were constantly under observation (by the author herself) to make sure that they were completely focused on the study and not wasting time as the measurement time to complete the task was vital data for this study. This observation was as unobtrusive as possible so that those being observed were not distracted from the task at hand as noted by Seaman (1999) in her paper on empirical studies of Software Engineering. Each of the paired groups was asked to take strict notice of the start time and the finish time they spent on each system. There was no time limit restriction to finish the exercise and the groups were free to spend as much time as they wanted. The exercise sheets were collected back when all groups had completed the tasks and were forwarded to the assessors.

10.7.4. Data analysis

For the purpose of data analysis, two variables were important for this study: the performance of the subjects and the time spent on finishing the exercise.

After the assessment was done the performance of each group was calculated using Equation 10-1.

$$Performance = (marks\ obtained/maximum\ marks) * 100 \qquad \textbf{Equation 10-1}$$

The results obtained for each individual task in the exercise for each of the group are summarised in Table 10-4 and plotted in Figure 10-3.

Table 10-4 Performance of groups (Experiment 1)

Groups	TCS	DAQ	FFT/ADC
PG1	66%	100%	75%
NPG1	40%	100%	75%
PG2	90%	100%	95%
NPG2	50%	75%	60%

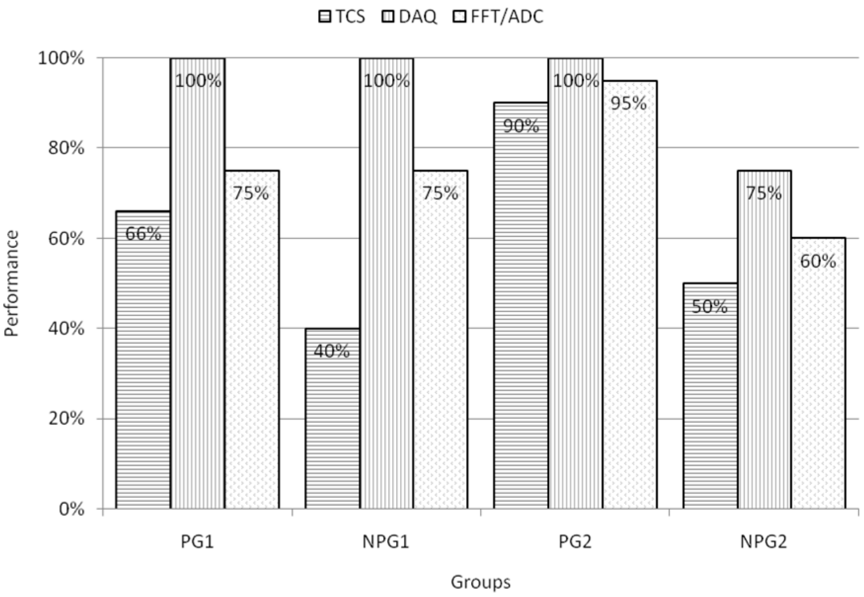


Figure 10-3 Performance of groups (Experiment 1)

The results reveal some interesting facts. For example, in the case of the DAQ application there is not much difference in the performance between groups PG and NPG except for NPG2 and even they obtained a high score of more than 70%. This suggests that patterns did not have a significant impact in helping the subjects working on systems with lower complexity (see Table 10-4). On the other hand there is a considerable difference in the results for PG and NPG groups for the TCS system which is at the highest level of complexity, indicating that patterns helped the subjects in understanding more complex architectures. Further the overall average performance of each group in the exercise is calculated and is shown in Table 10-5 in column 'Average'. To further compare the mean or average performance are again calculated overall for PG groups and NPG groups and is shown in column 'Overall Average'.

Table 10-5 Overall average performance of groups in the exercise

Groups	TCS	DAQ	FFT/ADC	Average	Overall Average
PG1	66%	100%	75%	80.33%	87.66%
PG2	90%	100%	95%	95%	
NPG1	40%	100%	75%	71.66%	66.66%
NPG2	50%	75%	60%	61.66%	

Results shown in Table 10-5 indicate the performance benefits of using patterns in both PG groups. Overall the pattern user groups performed better than the groups that worked without patterns.

Regarding the validity of the statistical hypotheses in software engineering experiments Juristo and Moreno (2001) has described some decision rules.

According to them the differences between the mean of the sample data can help in accepting or rejecting a hypothesis and analysing the significance of it. A hypothesis is normally rejected if the differences between the mean (for two alternatives) is either zero or a value less than 5%. Alternatively, higher values of the difference between the mean indicates that the hypothesis is acceptable and proves the validity and significance of the stated hypothesis. For the validity of the hypothesis H1 the mean of the performance of PG and NPG groups and the difference in mean is calculated and is shown in Table 10-6.

Table 10-6 Calculations for the difference in mean of PG and NPG groups

	PG	**NPG**
Performance of groups(%)	80	72
	95	62
Sum	∑PG = 175	∑NPG = 134
Mean	PG’ = 87.5	NPG’ = 67
Difference of mean	PG’ – NPG’ = 20.5	

The high value of the difference of the mean indicated that hypothesis H1 is acceptable for the experiment described above.

In order to further analyse the results the overall time spent (in minutes) is plotted graphically and shown in Figure 10-4.

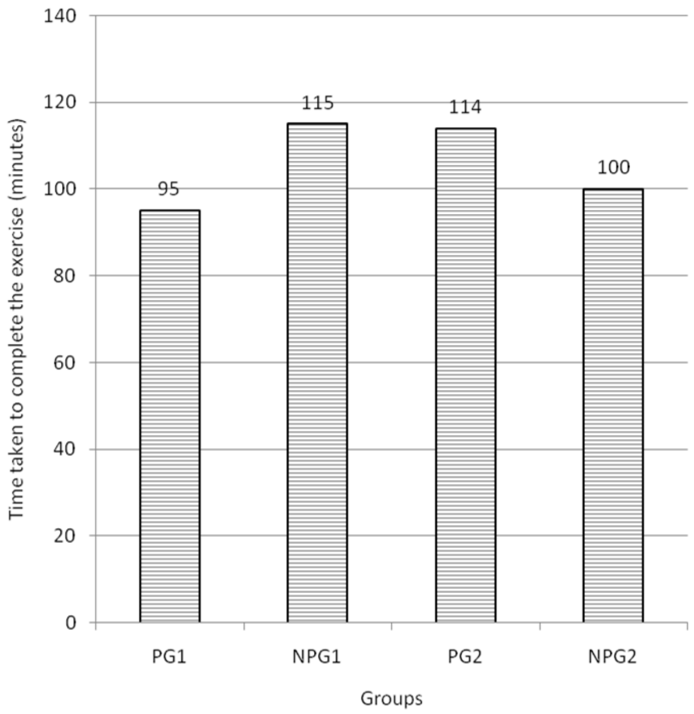


Figure 10-4 Time taken by each group to complete the exercise (Experiment 1)

For the time measurements, the total time spent by the groups on the whole exercise was determined from the recorded start and finishing times spent on each application.

It is quite interesting to note that the PG2 group overall spent more time than PG1. This was because this group needed to spend more time familiarising themselves with the information provided in the patterns (as noted by the observer during the experiment). Interestingly the performance of PG1 (the 90% scorers in the PES-1 module) and PG2 (the 70% scorers in the PES-1 module) PG2 performed extremely well in the exercise as shown in Table 10-4. This implies that expert knowledge and availability of patterns is not enough, but lesser expertise plus a more thorough study of patterns can be very productive. Further work would be needed with more groups however to verify this and identify if there were other variables present.

10.8. Experiment 2: Patterns versus alternative resources

10.8.1. Aims and motives

At one level, a pattern is simply a structured document which formalises the relationship between a non-trivial problem and a non-obvious solution. It is implicit in much of the work on design patterns that this “pattern form” offers advantages over traditional ways of representing this type of information (for example, in a textbook). Different authors have their own ways of documenting patterns. However, certain pattern forms have become more established than others. For example the *Alexandrian form* used by Alexander in (Alexander, Ishikawa *et al.*, 1977), the *GoF form* use to write software patterns for object-oriented software by (Gamma, Helm *et al.*, 1995), the *POSA form* use to write Patterns for Software Architecture (Buschmann, Meunier *et al.*, 1996) and the PTTES format use by Michael Pont to write Patterns for Time-triggered Embedded Systems (Pont, 2001). For the PMES collection we have followed the PTTES form because of its relevancy to the context of the experimental investigations. All the patterns in the PMES collection have information organised in the layout shown in Figure 10-5.

PATTERN NAME

Context
This describes the situation for which the pattern is applicable.

Problem
This is the statement and description of the problem for which the pattern is documented.

Solution
This describes the core of the solution to the problem.

Related patterns and alternative solutions
This section contains information regarding related patterns that may be affected by the application of the solution described in the pattern and other possible alternative solutions.

Reliability and safety issues
This gives reliability information specific to the pattern.

Examples
This includes relevant code examples.

Overall strengths and weaknesses

- ☺ Positive effects
- ☹ Negative effects

References

Figure 10-5 Structural layout used for documenting patterns in the PMES collection

For this experiment the specific hypothesis under test was the following:

Hypothesis H2: The PMES form helps the pattern user to apply the information presented in a more effective way than “traditional” information sources.

10.8.2. Management of subjects

One of the potential users of the proposed pattern collection are new graduates of Computer Science who are about to start their career in the area of embedded systems development. Graduates in Computer Science have already taken essential courses in software development/programming and have a general set of skills which are attractive to companies working in the business of embedded systems. In applying and extending their skills to new engineering problems in a professional environment, such graduates would rely on resources such as books, manuals, research papers, and other on-line material to complement their learnt knowledge and tacit skills to help solve the problems.

For the selection of subjects in this experiment, an email was sent to the group of the MSc (Computer Science) students of The University of Leicester. Out of the list of volunteers 10 high performing students were chosen whose academic performance was comparable and most of them were distinction holders in their previous semesters. In order to fill in any knowledge gaps the participants may have had on embedded systems, an introductory tutorial was prepared (by the author herself) and emailed to all the participants a week before the study. The introductory tutorial covered the essential concepts of embedded systems and the terms that would be used in the exercise.

For this experiment participants worked individually instead of in groups. The reason for this choice was to avoid differences in the ways the supplementary material would have been used and digested. Out of the ten candidates, five individuals were randomly selected to work with patterns and were given the labels PP1 to PP5 (PP refers to pattern-participant), and the five that worked without patterns were given the labels NPP1 to NPP5 (NPP refers to non-pattern participant). This distribution is shown in Table 10-7.

Table 10-7 Participants in the study

Pattern Participants	PP1	PP2	PP3	PP4	PP5
Non-Pattern Participants	NPP1	NPP2	NPP3	NPP4	NPP5

10.8.3. Preparation of the exercise

For this study an exercise was prepared based on smaller tasks with the aim of testing the hypothesis H2. There were 4 different tasks in the exercise and each was designed to test a different pattern with a consideration that the relevant information must be available through other supplementary resources provided to the non-pattern participants to compare results. For example, Task 2 in the given exercise was related to a programming technique which can allow programmers to make the worst-case execution time (WCET) of a task equal to the best-case execution time (BCET) in order to improve code predictability. In the newly proposed PMES collection, the pattern SINGLE PATH DELAY under the umbrella of the abstract pattern BALANCED SYSTEM has discussed this technique. Originally, this technique is proposed by Peter Puschner and is discussed in some of his papers (Puschner and Burns, 2002; Puschner, 2003) . More details about each task are presented in

Table 10-8.

Table 10-8 Details of the tasks given in the exercise (Experiment 2)

	Relevant Patterns	Relevant resources
Task 1	EVENTS TO TIME	(Kopetz, 1991; Kopetz, 1997; Albert and Bosch GmbH, 2004; Scheler and Schroder-Preikschat, 2006)
Task 2	SINGLE PATH DELAY TAKE A NAP	(Puschner, 2002; Puschner and Burns, 2002; Puschner, 2003; Gendy and Pont,

		2007)
Task 3	Loop Timeout	(Pont, 2002)
Task 4	Choosing Task Parameters	(Gendy and Pont, 2008a; Gendy and Pont, 2008b)

Please note that a copy of the complete exercise used in this study is included in Appendix C.

10.8.4. Procedure

At the beginning of the experiment all the participants were given the exercise and resources to help them to solve the exercise. The pattern-participants (PP1 to PP5) were given the patterns in the PMES collection along with other supplementary material i.e. the relevant research papers and books. The non-pattern-participants (NPP1 to NPP5) were provided with the same relevant research papers and books only (books and papers were the same as those given to the pattern-participants). A pre-test assessment was carried out to make sure that all subjects had gone through the introductory tutorial provided before the study, and have a basic understanding of the embedded system concepts required to solve the exercise. To ensure this all the subjects were asked to write a short paragraph about their understanding on ET and TT architectures.

After the pre-test, participants were given the actual exercise and relevant research papers. Participants PP1 to PP5 were also given copies of the PMES pattern collection and they were given the option to use them if they wished. However, it was interesting to note during the study that all participants who were provided with the pattern collection preferred to use them against papers/books as their first choice. An obvious reason is the clearer and precise impression of a pattern as a training document compared with its more general description in research papers and books. During the study, as part of capturing the essential data in this study, all the participants were asked to note down the start time and finish time for each task. On completion of the exercise, subjects were asked to complete questionnaires designed to obtain their feedback on the exercise. Different questionnaires were designed for the PP and the NPP subjects.

After the pre-test, participants were given the actual exercise and relevant research papers. Participants PP1 to PP5 were also given copies of the PMES pattern collection and they were given the option to use them if they wished. However, it was interesting to note during the study that all participants who were provided with the pattern collection preferred to use them rather than papers/books as their first choice. An obvious reason is the clearer and precise impression of a pattern as a training document compared with its more general description in research papers and books. During the study, as part of capturing the essential data in this study, all the participants were asked to note down the start time and finish time for each task. On completion of the exercise, subjects were asked to complete questionnaires designed to obtain their feedback on the exercise. Different questionnaires were designed for the PP and the NPP subjects. Copies of questionnaires given to both PP and NPP participants are included in appendix B.2.

10.8.5. Data analysis

For the data analysis, individual performance of the participants in the pre-test, which is based on the tutorial and in the actual exercise, is calculated using Equation 10-1 and the results are shown in

Table 10-9 and Figure 10-6.

Table 10-9 Performance of students in Experiment 2

	PP1	PP2	PP3	PP4	PP5	NPP1	NPP2	NPP3	NPP4	NPP5
Pre-test	60%	80%	60%	70%	80%	60%	80%	60%	60%	80%
Actual	67.5%	90%	70%	65%	70%	37.5%	27.5%	55%	17.5%	30%

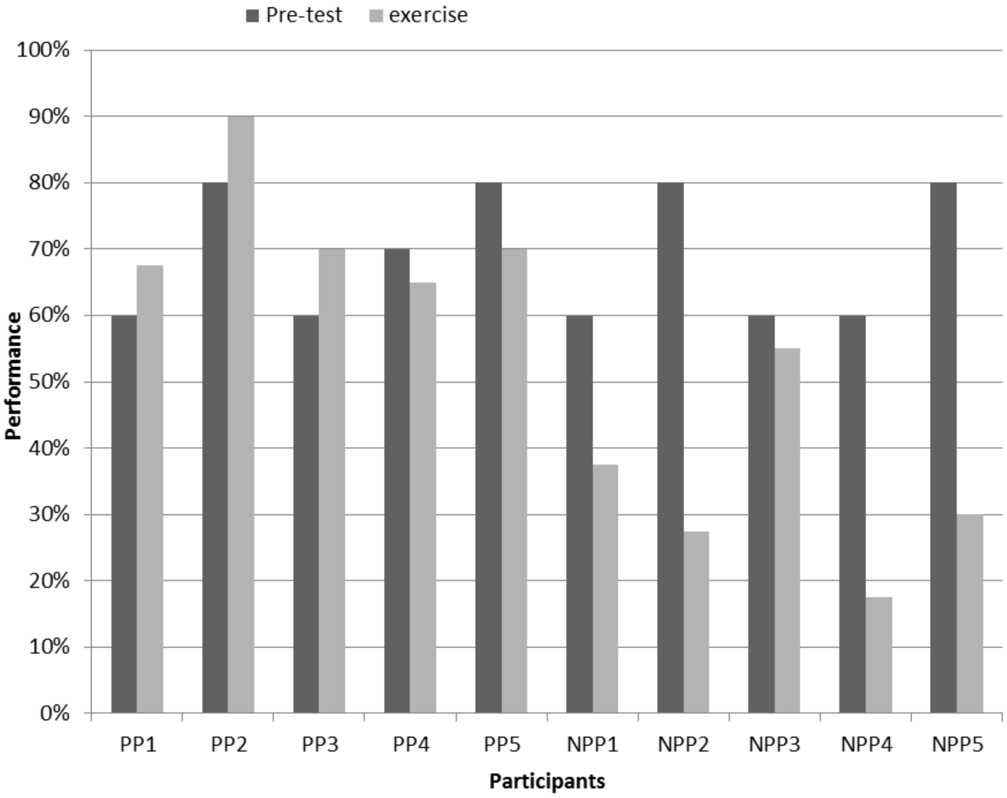


Figure 10-6 Performance of participants (Experiment 2)

In the pre-test, participants scored between 60% and 80%. This score was random between PP and NPP participants indicating the average knowledge of basic embedded systems concepts among all the participants was nearly equal. However, the results of the actual exercise show a clear difference in performance of the participants who worked with patterns compared with those who worked with other supplementary materials provided for them. The results shown in Figure 10-6 clearly indicate the difference of performance for PP and NPP participants.

For further data analysis, the same methodology is adopted that was discussed in Section 10.7.4 and the values of the overall average performance of the participants worked with and without patterns are calculated and is shown as ‘Average’ Table 10-10.

Table 10-10 Overall average performance of PP and NPP (Experiment 2)

	PP1	PP2	PP3	PP4	PP5	NPP1	NPP2	NPP3	NPP4	NPP5
	67.5%	90%	70%	65%	70%	37.5%	27.5%	55%	17.5%	30%
Average	72.5%					33.5%				

The performance of all the PPs is higher than the mean of the overall result. The total time taken by each participant to solve the complete exercise was calculated and is shown in Figure 10-7.

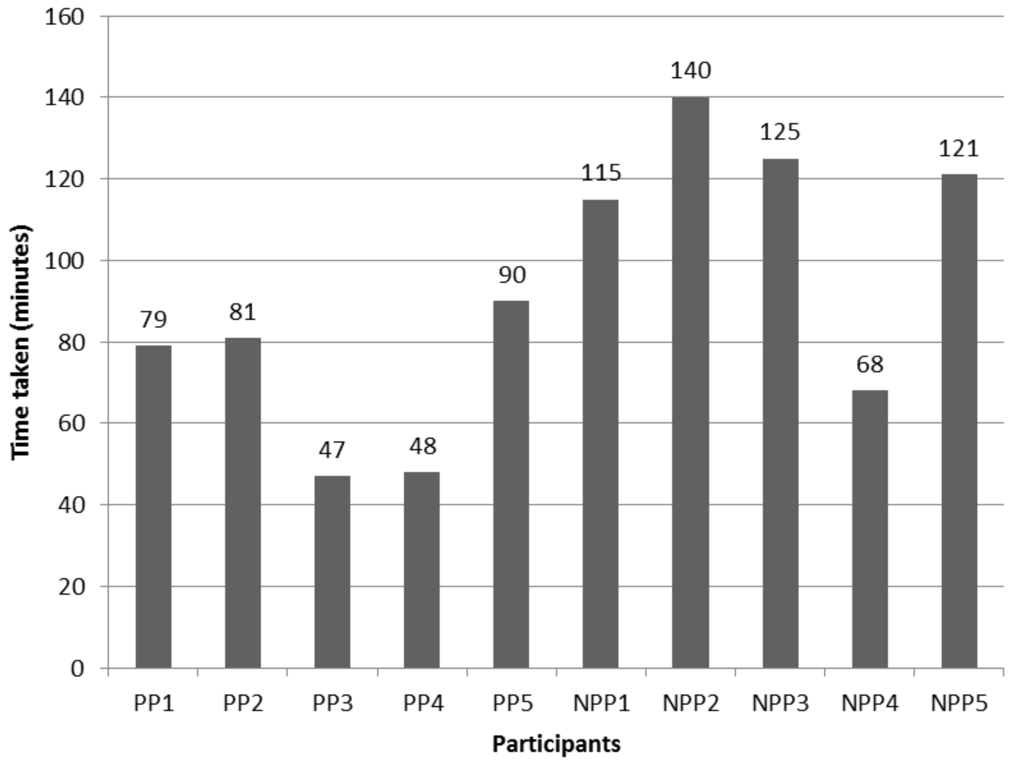


Figure 10-7 Time taken by the participants to complete the exercise (Experiment 2)

The graph clearly indicates the difference in the time taken by the PP and NPP participants to complete the exercise. Results indicated that the subjects provided with the pattern based representation of the material generally managed to absorb the knowledge and apply it more quickly than those who have got the equivalent material in the form of research papers and books. This implies that patterns also provide a way of representing information in a way which is easier to digest and understand.

To investigate the acceptance level and significance of the hypothesis H2 the differences of the mean are calculated as shown in Table 10-11.

Table 10-11 Calculations for the differences of mean for experiment 2

	PP	**NPP**
Performance of participants in (%)	67.5	37.5
	90	27.5
	70	55
	65	17.5
	70	30
Sum	∑PP = 362.5	∑NPP = 167.5
Mean	PP’ = 72.5	NPP’ = 33.5
Difference of mean	PP’ – NPP’ = 39	

The difference of mean is a high value indicated that the hypothesis H2 is acceptable.

10.8.6. Feedback from the subjects

Some important results were also extracted from the questionnaires given to all the participants at the end of the study. This section will discuss the results obtained.

- **Difficulty level of the exercise**

  In one of the questions the participants were asked: “How difficult did the exercise appear to you given there was a requirement to give extra effort to find the relevant information from the additional resource material provided?”. A difficulty level (based on a Likert-type scale[21]) was defined in the questionnaire going from 1 to 4 where 1 = Too Easy, 2 = Easy, 3 = Difficult and 4 = Too Difficult. The results are depicted in Figure 10-8.

[21] A Likert scale is a psychometric scale commonly involved in research that employs questionnaires. It is the most widely used approach to scaling responses in survey research. The scale is named after its inventor, psychologist Rensis Likert (Source: Wikipedia).

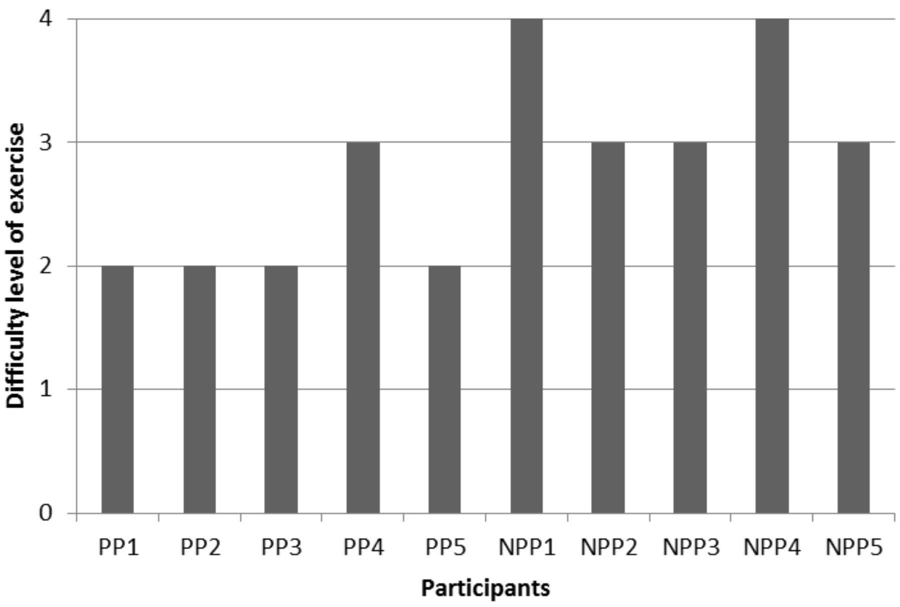


Figure 10-8 Feedback from the participants regarding the difficulty level of the exercise

The results showed that for PP the overall exercise was easy to solve compared with NPP who were struggling to find the relevant information from books and research papers.

- **Value of the material provided**

To obtain feedback about the documents provided another question was asked in the questionnaire: “Did you find the given documents helpful to solve the tasks given in the exercise?” The results are shown below in Figure 10-9. The participants were again given the following four options (1 to 4) where 1 = Not helpful, 2 = A little bit helpful, 3 = Helpful and 4 = Very helpful, and invited to select one.

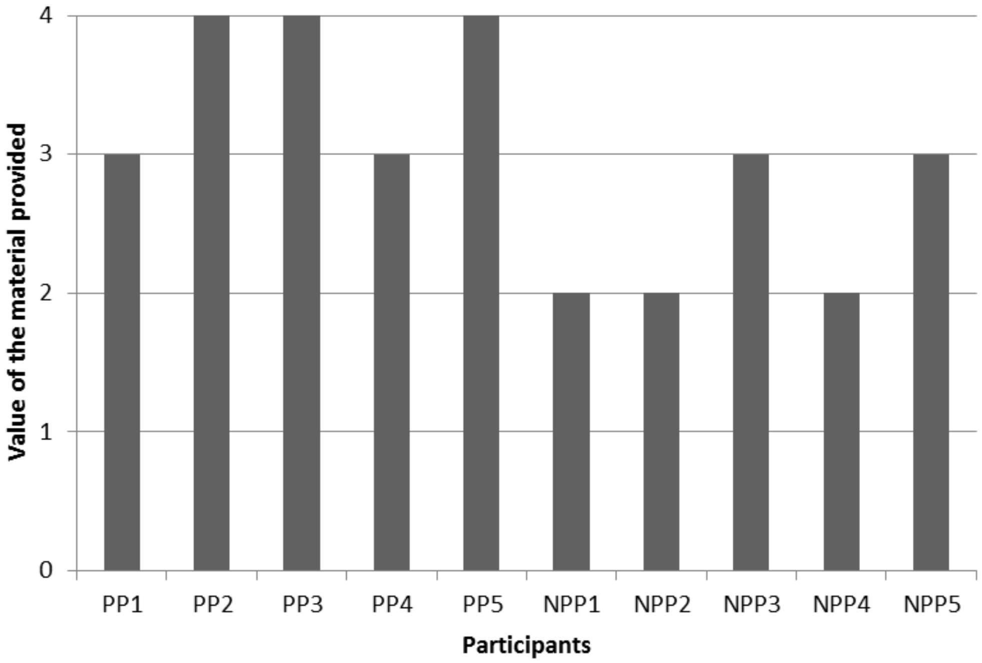


Figure 10-9 Feedback of the participants about the value of the material provided

Participants who worked with patterns found these documents very helpful (average rating = 3.6) while participants provided with other documents felt they were only a little bit helpful (average rating = 2.4). These results also reflected the significance of patterns as documents because of their structural simplicity, and the distribution of information into clear sections of problem, context and solution.

- **Feedback about patterns**

There are certain elements in a pattern which play an important role in the success of a pattern indeed even the name of the pattern is an important consideration. If the name encodes the patterns' meaning well, a designer can more easily find a suitable pattern in an unfamiliar pattern language (Coplien, 2000). In the PMES collection most of the pattern names are quite simple such as TIME FOR TT, EVENTS TO TIME and TT SCHEDULER and they give a pretty clear indication of the purpose for which they are designed. Some pattern names however are chosen to describe their function by analogy such as Sandwich Delay, Single Path Delay and Take a Nap and these names can be problematic to inexperienced designers who may be unfamiliar with the domain. Also, at the heart of a pattern is the solution described in it. It was felt important to obtain feedback from individuals about the clarity of different sections of each individual pattern in the given pattern collection used during the study. Therefore, one of the questions asked in the questionnaire aimed to rate the individual sections of each pattern. The participants were invited to rate the names of the patterns and the solution described in terms of the clarity they gave in representing the actual function of the patterns. The clarity was quantified using a five point Likert scale from 1 to 5 where 1 = Lowest and 5 = Highest.

The pattern Choosing Task Parameters received the highest rating in terms of the clarity of the name whilst Take a Nap is least understandable because it requires one to unpack its analogical meaning.

For the clarity of the solution, the pattern Events to Time appeared as the most highly rated pattern (average 4.4) while Choosing Task Parameters as lowest rated (average 3.2). This information provides useful feedback for making improvements to patterns.

The results obtained are shown in Figure 10-10 gained from the participants' feedback.

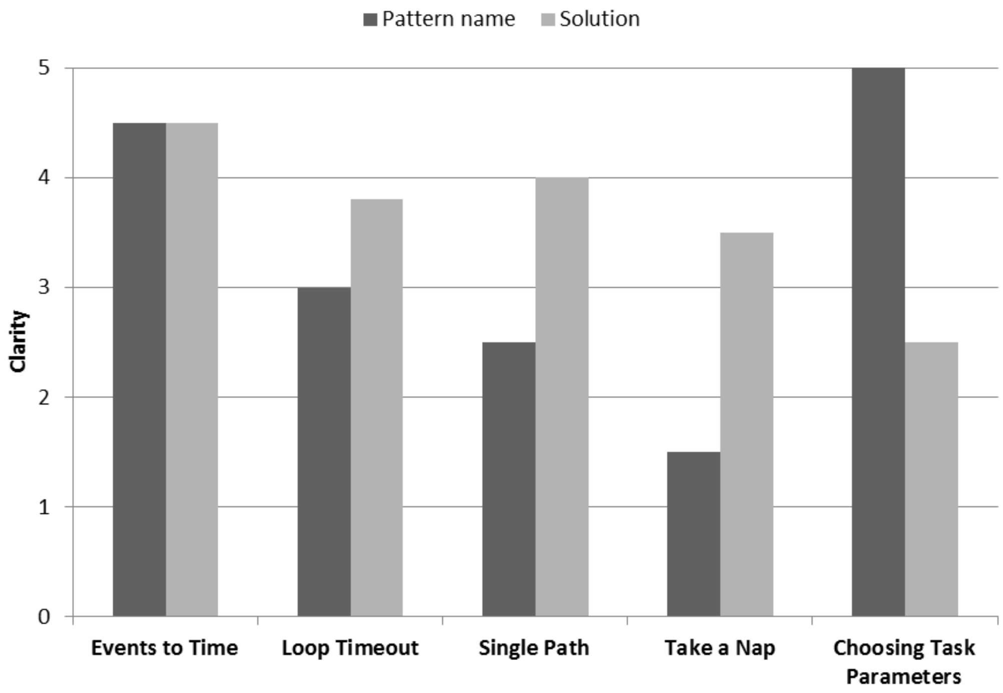


Figure 10-10 Feedback about pattern names and solution (1-5) 1 = Lowest, 5 = Highest

Earlier in this thesis in Section 6.8 the idea of pattern *'Shepherding'* was described. This is a process of bringing improvements in a pattern through the critical review of an expert. The results indicated that patterns that have already passed through *'Shepherding'* and have been assessed at *'Writer's Workshops'* are better understood by users. For example the pattern CHOOSING TASK PARAMETERS is documented in the later stages of the research and has not passed through any PLoP conferences yet and this could be one of the reasons for its lowest rating.

10.9. Discussion

As discussed at the beginning of this chapter, newly proposed patterns must pass through a process of evaluation in order to assess their quality and efficacy for the domain they are designed for. In this regard, two different empirical studies were conducted to assess the novel pattern collection generated as an outcome of this research. The literature provides evidence that evaluation studies for a pattern collection are highly dependent on the characteristics of the application domain and practical constraints faced while conducting such studies. In the light of such evidence, the studies presented in this chapter were designed with the aim of gaining useful results within the presence of constraints.

Experiment 1 discussed in this chapter was designed to test the usability of the PMES collection for reasonably experienced designers/developers of embedded applications. The results have shown that a team of experienced embedded software designers can enhance their insights in software development with the use of patterns during the migration process from ET to TT designs for complex embedded applications. Groups that worked with patterns exhibited better performance in the exercises given to them during the study compared with those who were completely dependent on their previous skills and experience. Figure 10-11 presents a summary of the results that were obtained relating to Experiment 1.

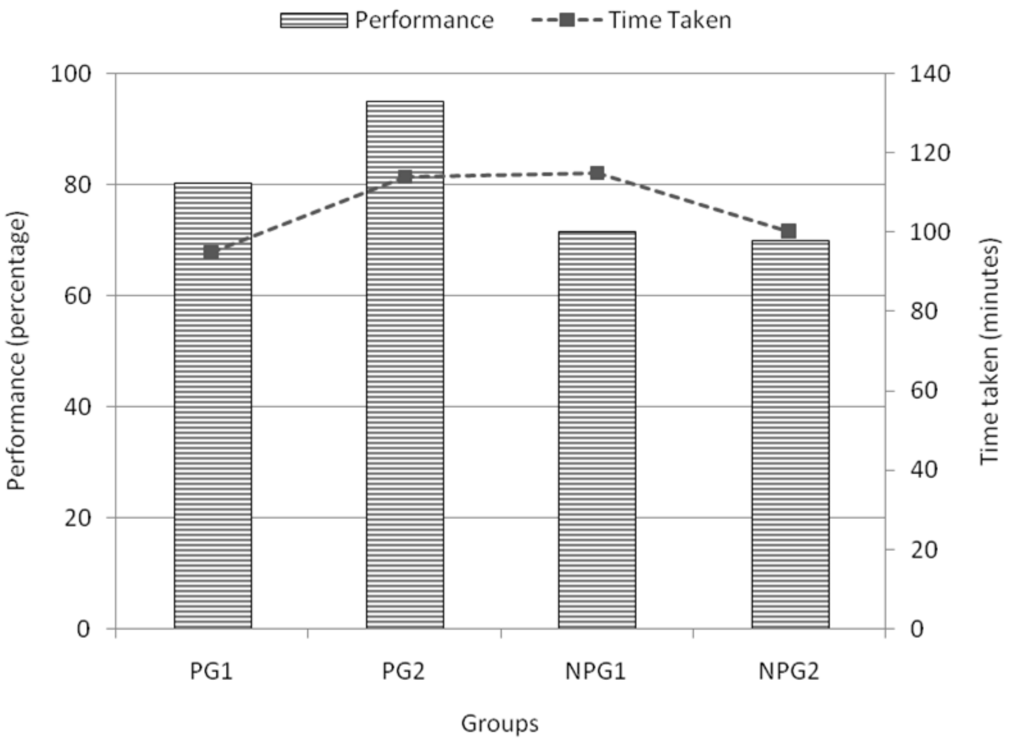


Figure 10-11 Summary of results (Experiment 1)

The graph indicates that:

- The performance of the PG groups is much better compared with the NPG groups
- The performance of PG2 is even better than PG1 possibly because PG2 spent more time on understanding the patterns.
- The time taken by PG2 and NPG1 is almost equal but the performance difference of the two groups is higher with, PG2 producing a 20% higher performance than NPG1 in the same time span.

Experiment 2 was designed to test the usability of patterns as documents compared with other supplementary materials which inexperienced developers of embedded applications refer to early in their career. The results showed that patterns are a useful replacement to other supplementary materials such as research papers, books, reference manuals etc. Patterns presented as structured and more concise documents proved to be a more digestible source of information providing both quick access to the information required and an overall time saving to completion of the exercises. It was observed that participants provided with supplementary material other than patterns experienced a greater difficulty in understanding the tasks. They also spent more time completing them and showed a lower performance in the tasks.

To scrutinise this data further a difficulty index for each individual participant is calculated and in defined in Equation 10-2.

$$\textit{Difficulty index} = (\textit{Observed difficulty} * \textit{time taken})/\textit{maximum absolute difficulty}$$

Equation 10-2

The maximum absolute difficulty (shown in Equation 10-3) is the product of the maximum difficulty level (scale set from 1-4 where 4 = Too Difficult) and maximum time taken by any of the participant to complete the exercise (observed value = 140 minutes).

$$\textit{Maximum absolute difficulty} = \textit{maximum difficulty} * \textit{maximum time}$$

Equation 10-3

The difficulty index for each participant was calculated and plotted against performance in order to analyse the overall impact of using patterns during the study. The results are shown in Figure 10-12. The trend of the results suggests that individuals who worked with patterns during the study showed much better performance in less time (lower values of difficulty index).

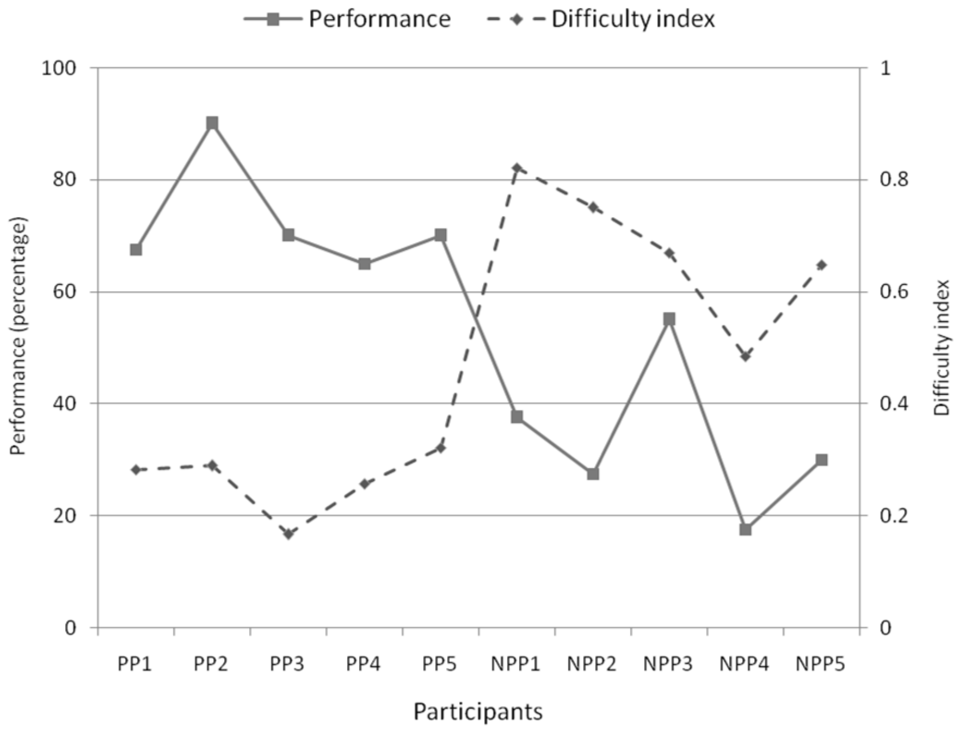


Figure 10-12 Summary of results (Experiment 2)

On the other hand participants who used other supplementary material during the exercise were unable to perform well and worked for a longer time (higher values of difficulty index), as they had the extra demand of extracting related information from the material provided.

In general, the results obtained from both the studies are useful because they helped in the evaluation of the patterns based on the criteria given in Table 10-1. For example, the better performance of pattern users in both the studies suggest that the proposed collection of patterns is 'effective' in the sense that it helped the users to identify the root causes of the problems (given in different tasks in the exercise) and to tackle them with the solution recommended in the patterns.

However, the studies conducted are not free from problems of interpretation - the main one being the small sample size of the subjects. However they do represent a first step towards ascertaining the benefits of applying patterns during the migration of architecture for complex embedded applications.

10.10. Conclusions

In this chapter the primary goal was to investigate the efficacy of the proposed pattern collection for the target users. Two empirical studies were designed to simulate the situations in which experienced and inexperienced developers of embedded applications could benefit from applying patterns during the migration between event-triggered and time-triggered architectures. The results discussed in the chapter suggest that the proposed pattern collection has the potential – provided they are applied correctly – to provide the intended support.

CHAPTER 11. ASSESSING THE PATTERNS: INDUSTRY FEEDBACK

11.1. Introduction

The design patterns that emerged as an outcome of this research were developed with the intention of providing support to the developers of embedded applications. As a first step in the evaluation of their usability for real-time systems CHAPTER 9 described the application of the PMES collection with two examples which are in widespread use in various industrial applications. In the second stage of the evaluation process, two pilot empirical studies were conducted involving students as subjects, to explain how the proposed pattern collection can help in taking appropriate decisions while working on real applications: these studies were discussed in CHAPTER 10.

As the real stakeholders of the proposed pattern collection are the developers and designers working in the industrial sector, it was decided to further evaluate the patterns in this context: the details of this work are provided in this chapter.

The chapter is organized as follows. Section 11.2 presents a discussion on the usability of the PMES collection in the current industrial context and how and why the proposed pattern collection could be of interest to working professionals in the embedded market. Section 11.3 presents a brief account of the aims of the evaluation process, and Section 11.4 discusses the practical constraints involved in the evaluation process in the industrial context. Section 11.5 describes the details regarding the methodology adopted for the process of evaluation with Section 11.6 providing a detailed account of the results obtained with this process. Section 11.7 presents a discussion on the overall results and comments obtained from the evaluation process and conclusions are given in Section 11.8.

11.2. PMES usability in the industrial context

While discussing current trends about software architectures in Section 4.5.3 it is mentioned that for the guaranteed safety of high integrity systems the industry is backed up by international safety standards incorporated in IEC 61508 which is a benchmark standard for developing and validating safety-related electronic systems. The standard is entitled: “Functional safety of Electrical/Electronic/Programmable Electronic Safety related Systems (E/E/PE)” (IEC, 2012). The rules defined in IEC 61508 are quite general and are intended to define a basic functional safety standard applicable to all kinds of industry. It is also mentioned that the automotive industry has adapted IEC 61508 to produce ISO 26262, a new benchmark standard for developing and validating safety-related systems that are installed in passenger cars. The vehicle production industry is now looking forward to design applications for modern passenger cars which are ISO26262 compliant. The aerospace industry has adapted the IEC 61508 and introduced the DO-178 standard for the development of software for aircrafts.

The research aimed to explore ways of assisting developers and organisations in situations where systems need to meet certification requirements. The particular goal was to avoid “reinventing the wheel” by helping people adapt their existing designs to make them suitable for use in systems with what IEC 61508 referred to as “Safety Integrity Level” or SIL. The following sections will describe how the PMES collection can help to achieve this aim.

11.2.1. Predictability in systems

The main reason to introduce predictability in systems is to make them safe and reliable. According to the documentation for IEC 61508 Edition 2, the definition of Safety is, ‘Freedom from unacceptable risk’. One possible way of avoiding risks is to design/configure a software system such that it exhibits predictable behaviour and allows one to determine in advance – before the system begins executing – exactly what it will do at every moment of time during which it is running. In other words, the architecture must allow testing of the system in a convenient way to guarantee the safety and reliability of the system. In the proposed PMES collection patterns TIME FOR TT and TT SCHEDULER has discussed how one can make their system safe and reliable and when and what TT architectures can perfectly match an application requirements.

For example the safety process for automotive vehicles is described in various parts in the documentation for ISO 26262 structure. Part 6 of the documentation specifically refers to the development of the software aspect of the product and section 6-7 of the standard is “Restricted use of interrupts” to guarantee the fault free and safer systems. In the PMES collection the pattern EVENTS TO TIME has described ways of transforming systems with restricted use of interrupts.

11.2.2. System requirements for safety-integrity

During a system design once the system architecture is specified and the design begins the functional safety requirements begin to be refined into specific design safety requirements. In order to achieve functional safety, one of the main emphases of IEC 61508 and ISO 26262 is on risk assessment to determine the steps necessary to reduce the risk of each hazard to an acceptable level, usually through 'safety integrity' for an electronic system. IEC 61508 has defined Safety Integrity Levels 'SIL' to relate to the probability of a dangerous failure. ISO 26262 has adapted these levels for the automotive industry as 'ASIL' or Automotive Safety Integrity Levels; however these are not defined as a probabilistic requirement.

In real practice the software safety requirements are particularly related to the determination of WCET of tasks designed for the system. The accurate WCET prediction matters as the variation in task timings could potentially leads to system failure. In the PMES collection pattern BALANCED SYSTEM and its associated patterns this issue has been discussed in detail and solutions provided to possible hazards that can introduce variation in task timings.

11.2.3. System monitoring

Section 7.11 mentioned that it is almost impossible to design a systems which is 100% fault-free even if designed with due care. At runtime systems are susceptible to fault occurrences which in the worst case scenario could lead to a loss of precious human lives and property. This places a responsibility on designers to avoid any such fault occurrences both at the initial design stage and while systems are in a running state. In the documentation for ISO

26262, Part 6 Annex D is specifically focused on software elements' freedom from interference. These sources of interference could be due to hardware problems, or software mismanagement such as task overruns. In the optimisation patterns of the PMES collection pattern SYSTEM MONITOR and the associated patterns discusses the possible run time error issues and how to cater to them in advance.

11.3. Aims of the evaluation

The aim of this evaluation process is to get feedback from the real stakeholders for whom this research is intended to support. The real stakeholders include a complete team of professionals who are involved at various stages in a complete project development. These include not only developers and designers but system architects, project managers, test engineers and technical mangers. The main objective of this evaluation is:

- To investigate that how the industry is looking at the challenges around the migration between different software architectures
- To introduce the patterns in the PMES collection to the working professionals and to get their opinion about how they can be useful to them.

11.4. Practical constraints involved with industrial evaluation

The ideal way to evaluate the patterns in the industry would be to test and implement the patterns with some 'live' industrial project passing through the process of migration. However there are certain factors that militate against achieving these ideal conditions and these are listed below:

- Industrial projects usually span over long time durations.
- Confidentiality of the organisation.
- Individual developer's personal issue such as it is a real challenge to enquire the practitioners if they had ever faced such a situation in their career.
- Ethical issues concerning research participants such as seeking sensitive information and maintaining anonymity.

Given these practical constraints and ethical factors, a compromise has to be struck on the kind of research that can be done in order to acquire as much information about pattern usage as possible. In this case it was decided that the best strategy was to use a questionnaire as described in the next section.

11.5. Methodology

The choice of the method to collect feedback is determined by the requirements of the investigation to be undertaken and the constraints on the client group with which the researcher is working. For example if the study concerns issues that respondents may feel reluctant to discuss with an investigator, a questionnaire may be the better choice than a face to face interview as it ensures anonymity. The geographical distribution of the potential respondents is also a factor certainly in terms of face to face interviewing, though of course recent technologies such as Skype have alleviated some of the constraints of physical location.

In the context of this research a real challenge was to look for a methodology through which the participants are introduced to the research goals, and the pattern collection, and then are asked to provide their feedback on migration challenges and the usability of the PMES collection. The target audiences for this research are highly skilled and sophisticated personnel in the field of embedded application development and it was important for such busy people to offer some kind of benefit to them in order to engage them in this evaluation research. It was felt that offering some new knowledge about patterns and pattern use that they could apply in their work would be a suitable incentive to enable a high level of co-operation.

Mindful of the practical constraints of this aspect of the research it was decided to go for a web-based survey methodology to obtain the required information on the value of the PMES collection to working professionals. Web-based surveys have the potential for a global audience and are more inclusive allowing a broader reach than phone or postal survey or direct interviews (Archer 2003). Once setup, web-based surveys are easy to carry out, making it easier to recruit large number of participants or to collect data repeatedly. Since the data is captured directly in electronic format, it also makes data analysis faster and cheaper than other survey methods.

11.5.1. Development of the website

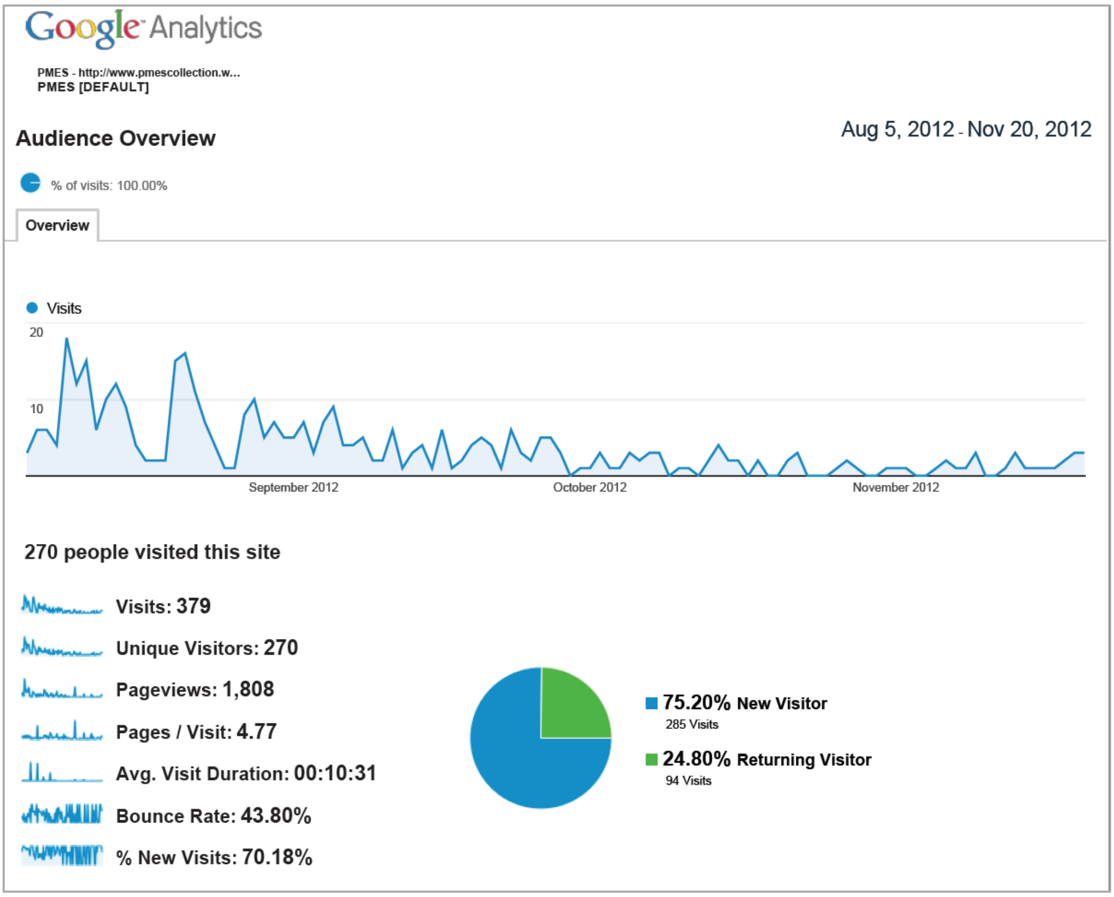
Having decided that a web-based survey would be used to obtain feedback on the PMES collection from industry, a website was designed for this purpose. The website introduced the target audience to the research goals and provided the survey questionnaire that was filled in on line. A free website hosting service was used and a free domain http://www.pmescollection.weebly.com was registered.

The main subject of the website was chosen as ‘Reliable Embedded Systems’. The website design was kept simple and the emphasis was on to the natural flow with transitions between one page and the next. The home page on the website provided an introduction to the research. The same page then gave the website visitors a guided tour to help them browse the pages that would be interest to themselves. There was a ‘Background’ section to help place the research in. The real meat of the research is described in a separate section called ‘PMES overview’ where the website visitor was introduced to the PMES collection which allowed them to look at the pattern association map, thumbnails, and where to the full specifications of the patterns could be downloaded.

To obtain feedback on the PMES collection it was essential to highlight the usability of the patterns in the collection so that the visitor will retain their interest and maybe motivated to read further and to explore applications in the current industrial context. This was discussed in a separate section called ‘PMES applications’ and where the various applications of the patterns (as discussed in Section 11.2) were described and how they can help the user in facing the industry challenges of developing systems designed according to standards such as IEC 61508 and ISO 26262. As part of these discussions the website visitors were given access to the individual patterns related to each application to further illustrate their value.

After navigating the webpages and being exposed to the potential benefits and real applications of the PMES pattern collection, the website visitors were invited to provide their feedback through a short questionnaire (a direct link was provided). In addition to questionnaire the visitors were provided with a comments section and invited to provide their opinions about the patterns' usability in an industrial context. The website also provided additional links to enable related publications and documents on further applications to be downloaded.

A Google analytics account was created to keep a record on the website statistics for new visitors and returning visitors. A snapshot of the report from the date the website was released to the date when the data reported here was collected, is shown in Figure 11-1.



Audience Overview

Aug 5, 2012 - Nov 20, 2012

% of visits: 100.00%

270 people visited this site

Visits: 379

Unique Visitors: 270

Pageviews: 1,808

Pages / Visit: 4.77

Avg. Visit Duration: 00:10:31

Bounce Rate: 43.80%

% New Visits: 70.18%

75.20% New Visitor
285 Visits

24.80% Returning Visitor
94 Visits

Figure 11-1 A preview of the Google analytics report: An account created to monitor the statistics on the visitors for the website www.pmescollection.weebly.com

11.5.2. Development of the questionnaire

The main considerations in designing an effective questionnaire includes reviewing the information requirements of the problem at hand, prioritizing a list of potential research questions that will satisfy the information requirements, determining the type of questions to be asked, and deciding about the structure of questionnaire (Peterson, 2000). A well-designed questionnaire can be used to yield three types of data that includes factual, behavioral and attitudinal data (Dornyei and Taguchi, 2009). Other desired features in developing useful questionnaires as noted by (Dillman, 2007) are to make the questionnaire interesting as well as short and easy so to avoid any inconvenience for the respondents. To respondents a feeling of being asked for help, advice or assistance provides a sense of reward and can motivate them to participate in the survey. In this case the questionnaire went even further by providing the additional benefits of free information on the PMES pattern collection

There are two common ways a survey question can be structured and these are known as open-ended and closed-ended. An open-ended question is one for which no answer choices are provided and respondents are free to provide their own opinion about the topic asked in the question. These are particularly useful for obtaining respondents’ feedback on general questions for which answer categories are difficult to define, for example how they see a general problem in the light of their own experience and expertise. At the same time open-ended questions have the drawback that there is no interviewer to clarify unclear answers and provide prompts to requests more details or explanation (Dillman, 2007). The closed-ended questions can provide answer choices (ordered or unordered) to the respondent. Ordered response categories are useful for the researcher when he/she has a well-defined concept for which an evaluative response is required, for example if something is useful or not useful or the degree of usefulness such as excellent, good or bad. The unordered categories are non-scalar response categories, for example in order to get to know a respondent’s expertise in a particular field, a number of options can be provided randomly.

Considering the factors mentioned above a questionnaire was developed for the evaluation of the PMES collection using the well-established Survey Monkey web tool. A link to the questionnaire was provided on the website designed for this purpose as mentioned in Section 11.5.2. The questionnaire was kept short (only 10 questions altogether) and included open-ended as well as close-ended (ordered/unordered) questions such that responses obtained can be used to achieve the aims mentioned in Section 11.3. The first two questions were close-ended (unordered) and were investigating the respondents' area of expertise and current position in the organisation. In the open-ended question the respondent is then asked to provide his/her own opinion about how the industry is coping with the problem of migration between different architectures. As the respondent is expected to visit the website and go through some of the patterns in the PMES collection he/she is then asked several close-ended questions to provide feedback on patterns.

The last question was an optional request for the respondents' email address.

In order to make sure that the respondent should not skip answering any question (excluding the last one) all the questions were marked with an 'Answer required' check at the design time. In addition at the end of each question an additional text box was provided for the respondent to add anything different from that given in the choices for selection.

A copy of the questionnaire is attached in Appendix D.

11.5.3. Approaching the targeted audience

The aim of this process was to evaluate the PMES collection in the industrial context and the targeted audiences for the questionnaire were the embedded system professionals working in industry specifically in the automotive, avionics and control engineering sectors. With the aim of achieving the maximum possible number of responses a link to the website was introduced on the social network for professionals called ‘Linked In’.

The initial response to this shared link was encouraging as a few people from the industry contacted the author and offered to provide their feedback. To achieve more responses a method of personalized emails was adopted to reach those people who are not very active on public forums. The e-mail addresses were obtained from contacts in the research group. An initial target was set to achieve at least 25 respondents in total (LinkedIn plus e-mail) to be able to draw some useful results.

11.5.4. Interpreting results

Interpreting results from close-ended questions is relatively easier as the options marked by all the respondent fall in the defined categories set by the researcher. This can help to simplify the researcher’s work by counting the number of options marked by the respondents and to provide statistical results. However, analysing results from open-ended questions is more challenging and requires techniques such as ‘text analysis’ to interpret results. Text analysis is a research technique for making replicable and valid inferences from texts to the context of their use (Krippendorff, 2004).This involves establishing categories and then counting the number of instances when those categories are used in a particular item of text (Silverman, 2011). This is usually achieved through creating nodes and coding to find some sort of order and coherence within the dataset and to see how data relates to the research question. Coding is a way of classifying data so that it can be reviewed by category as well as source. A node within a coding scheme is a representation of an idea, theme or category in data. Segments of data from across the dataset are coded to these nodes. This enables to retrieve all the data related to a node.

11.6. Results

Over a period of two months when the website first went live twenty-eight people responded to the questionnaire and provided useful feedback on the challenges of software architecture migration in the industry, and on some of the proposed patterns in the PMES collection. The option of leaving the contact details for respondents at the end of the questionnaire helped to identify some of the organisations from which the participants were associated. Useful feedback responses were received from professionals from organisations based in Australia, India, Malaysia, Miami, Pakistan, Singapore, Switzerland, Tunisia, UK and USA.

Once collected all the responses were analysed using the techniques mentioned in Section 11.5.4 for both close-ended and open-ended questions and the results are presented in the following sections.

Please note that a summary of these results is also included in Appendix D.

11.6.1. Respondents' area of expertise and professional status

In order to determine the credibility of the results obtained from responses to the questionnaire, it is important to know something about the respondent's areas of expertise and their current status in the organisation. The first two questions in the questionnaire were designed to elicit this information and it was interesting to note that the respondents work in a variety of areas in embedded systems development shown in Figure 11-2. .

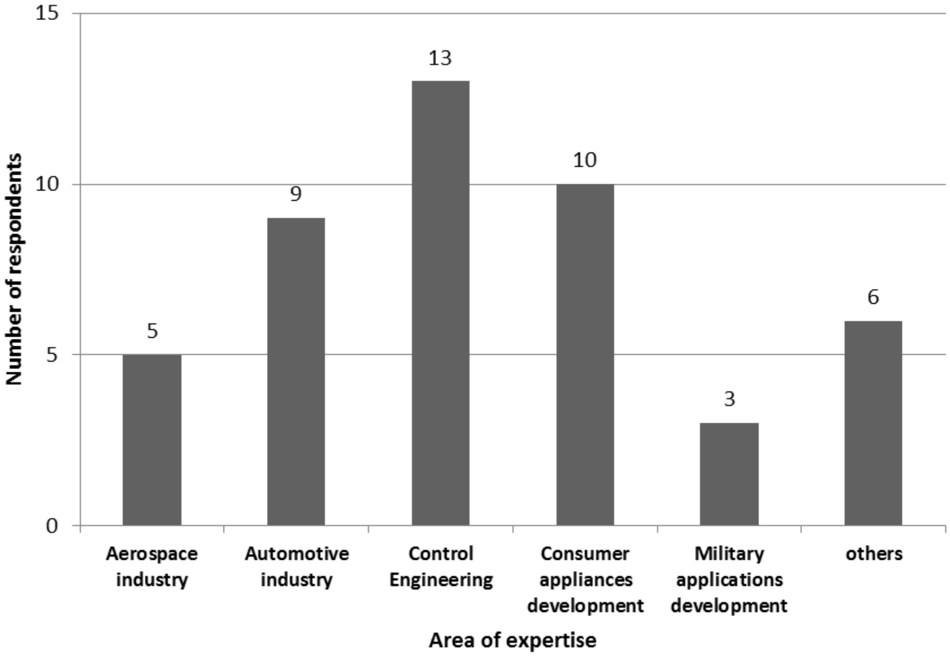


Figure 11-2 Respondents area of expertise

Others in Figure 11-2 were technical manager, technical training leader and planning and project controls manager.

The majority of the respondents were software designers as indicated in Figure 11-3. These results were encouraging as the research aimed to obtain feedback from professionals in diverse fields of embedded software development. Furthermore software designers' views about the pattern collection are the most useful as they are the people who have 'hands on' experience of software challenges. . Most of the respondents work in the field of control engineering, the automotive industry and consumer applications development and a few were in the aerospace sector. Fortunately these are the application areas for which the proposed patterns are specifically targeted which helps in building confidence on the feedback and comments received.

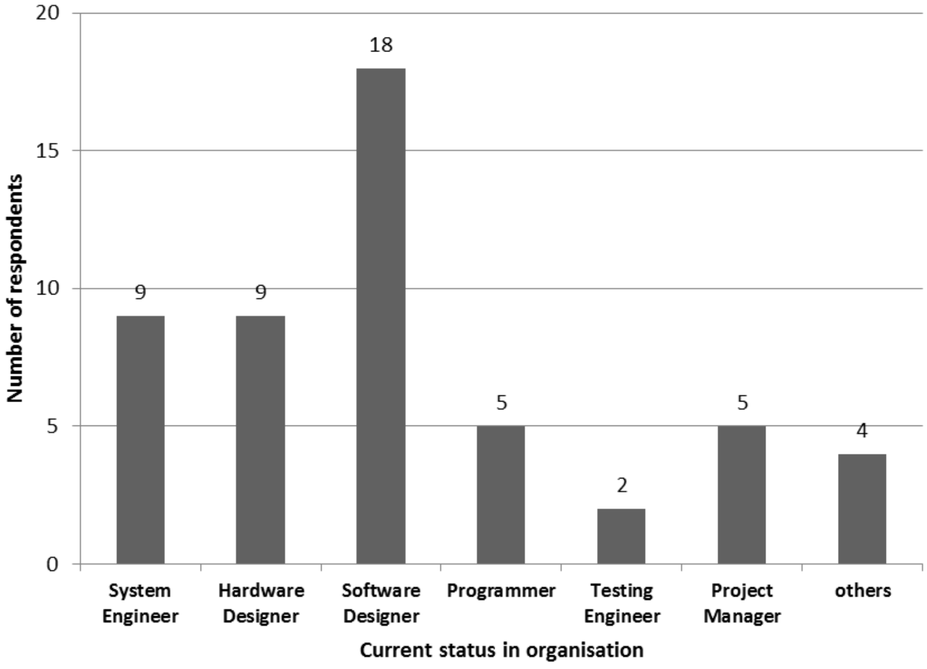


Figure 11-3 Respondents current status in organisation

Other areas of expertise in Figure 11-3 were security sector, telecommunications, building services, fire control panels, security panels and medical devices.

11.6.2. Adapting existing software to match new requirements

It was essential to know the respondents' opinion on whether, in order to meet new system requirements, adapting the existing software architecture is usually preferred, to building the systems from scratch. The respondents' feedback is shown in Figure 11-4.

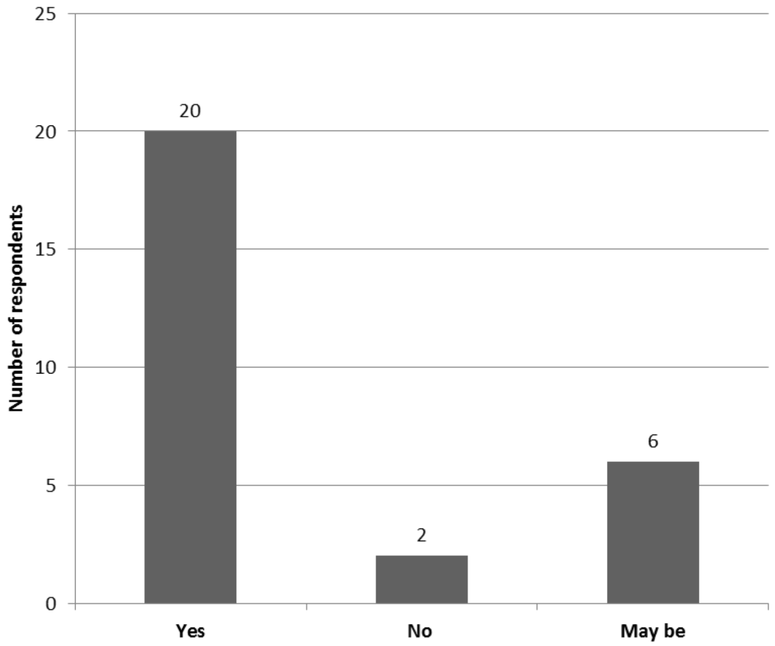


Figure 11-4 Responses to the agreement that "software for future versions of systems will rarely be created from scratch instead existing software will be adapted to match the new requirements"

This question was included in the questionnaire to identify the usual practices that practitioners adopted when upgrading or enhancing a system's functionality. The majority of the people (around 72%) accepted that the common practice is to make changes in the existing application rather than building the system from scratch. This has highlighted the usefulness of patterns for re-design for less experienced people in industry. Such a collection of patterns can help them to follow the best practices of experts with the least possibility of errors, rather than re-inventing the wheel. One of the participants gave the interesting comments below:

“This is primarily a business decision. With the right skills, a team may engage in adapting an existing solution to meet regulatory standards. However, if a new hire is recruited to handle non-functional requirements, they may suggest a completely new architecture if the present design does not conform to certain specific industry standards. I would however expect the new hire to be a specialist with knowledge of an already existing "safety" framework into which the proprietary functional/business-specific design can be incorporated. So in high probability either one of the frameworks will be well-established.”

11.6.3. Feedback on patterns to support migration

To identify the importance and usability of the proposed pattern collection for working professionals it was important to obtain their opinion about what they think about the pattern collection which may help them in changing the underlying software architecture at various stages. The respondents’ feedback was quite encouraging in this regard as the majority (68%)are in favour of such support as shown in Figure 11-5 below.

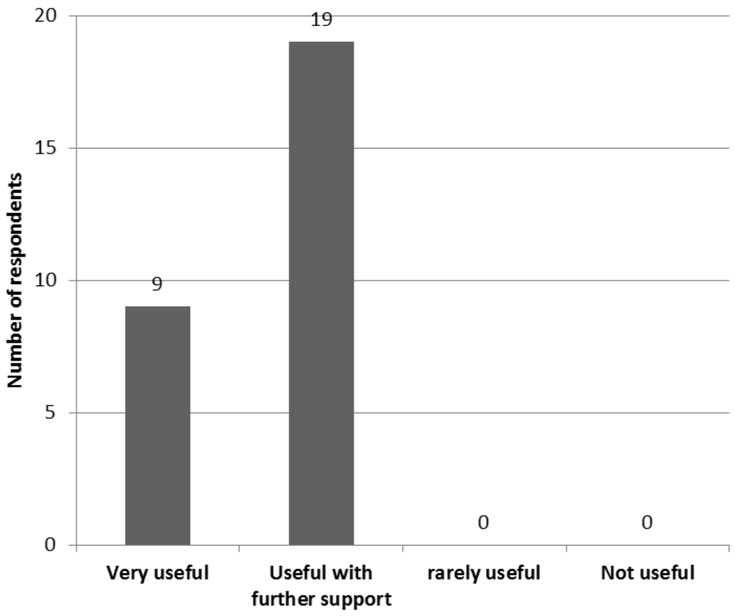


Figure 11-5 Responses to the usability of patterns for supporting the migration between software architectures

Another question was specifically asked about the patterns TIME FOR TT, EVENTS TO TIME and TT SCHEDULER that if the information provided in these patterns is helpful enough for practitioners, if in real practice they need to carry out the transformation from ET design to TT design. Altogether 71% favoured the use of the patterns and 25% were not completely sure. One interesting comment obtained is quoted below:

"Although patterns are a useful tool, there are many real life scenarios where the hypothesis breaks down and they are not applicable anymore in their pure form".

The responses are shown in Figure 11-6.

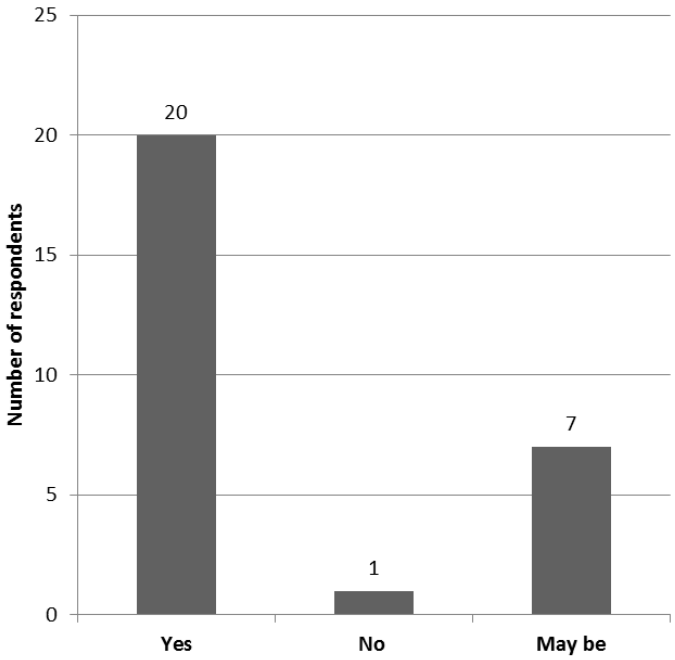


Figure 11-6 Responses on the usability of patterns TIME FOR TT, EVENTS TO TIME and TT SCHEDULER in real practice

Nowadays the industry is actively focusing on standards for functional safety integrity mentioned in the documentation for IEC 61508 and ISO 26262. With this reference on the website the visitors are introduced to the pattern BALANCED SYSTEM and associated patterns, and in the questionnaire it is asked if they think that that the techniques discussed in these patterns if implemented properly can help in achieving the systems' safety integrity requirements. Results of the responses obtained are shown in Figure 11-7.

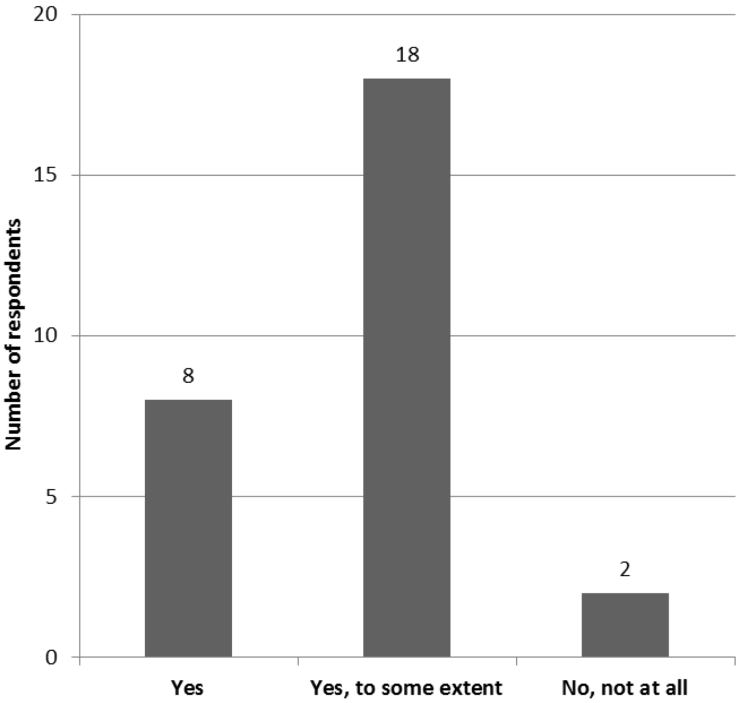


Figure 11-7 Responses to the usefulness of the pattern BALANCED SYSTEM and associated patterns

Altogether 64% agreed that these techniques are useful to some extent while 29% fully favoured the techniques described in these patterns 7% were not in agreement and one of these commented below:

“Success in controlling jitter is also affected by a combination of the estimation of resources for the system and is not always easy to do in advance”.

This is a valid comment as creating systems which are completely jitter free are idealistic and cease to exist in reality as discussed in the background section of the pattern BALANCED SYSTEM.

With regards to safety and reliability concerns the website visitors were introduced to the pattern SYSTEM MONITOR and TASK GUARDIAN and are asked to provide their feedback if they think that these techniques can help them in achieving “fault-tolerant” systems. The responses obtained are encouraging and are shown in Figure 11-8. It was interesting to note that 29% of respondents did not completely agreed with the idea while 57% thought that the techniques are useful partially and 14% were of the opinion that these are not useful at all. One respondent argued that:

"Our systems always have more than one processing unit so it might be really hard to use this if the pattern is meant for a single processor system"

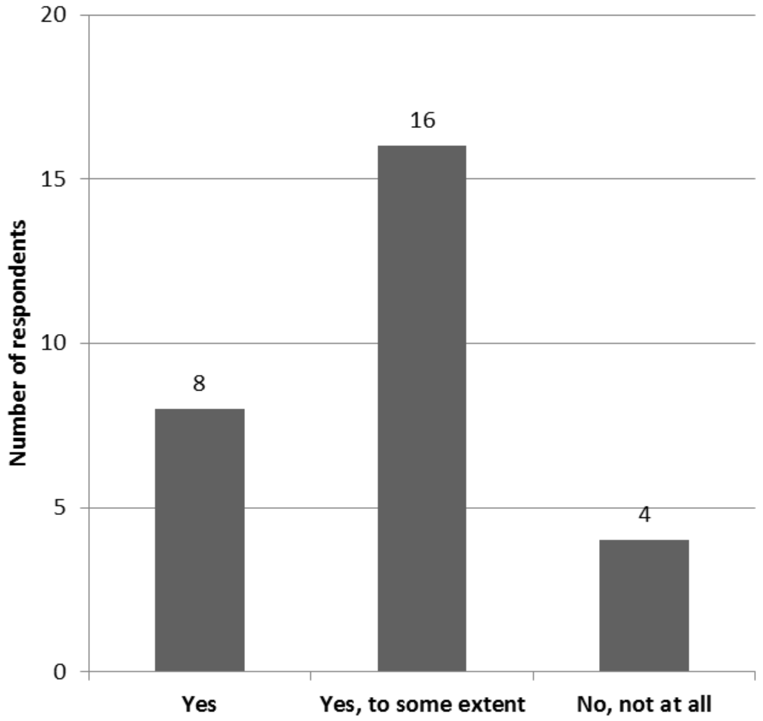


Figure 11-8 Responses about the usability of the pattern SYSTEM MONITOR and associated patterns

Another respondent argued on the pattern TASK GUARDIAN that:

"What if the long task has the I/O in some intermediate state? Ending the task with it in that state could be unsafe. I agree as system monitor is needed, it is question of what it does when a fault is found".

One more critique commented that: *"Monitors can become very complex and be unable to detect certain failure conditions or fail themselves".*

These additional comments are helpful in realising the improvements needed on this aspect of the work.

Finally respondents provided their feedback on the overall application of patterns in achieving currently in use standards of reliable and safe systems. Responses are shown in Figure 11-9.

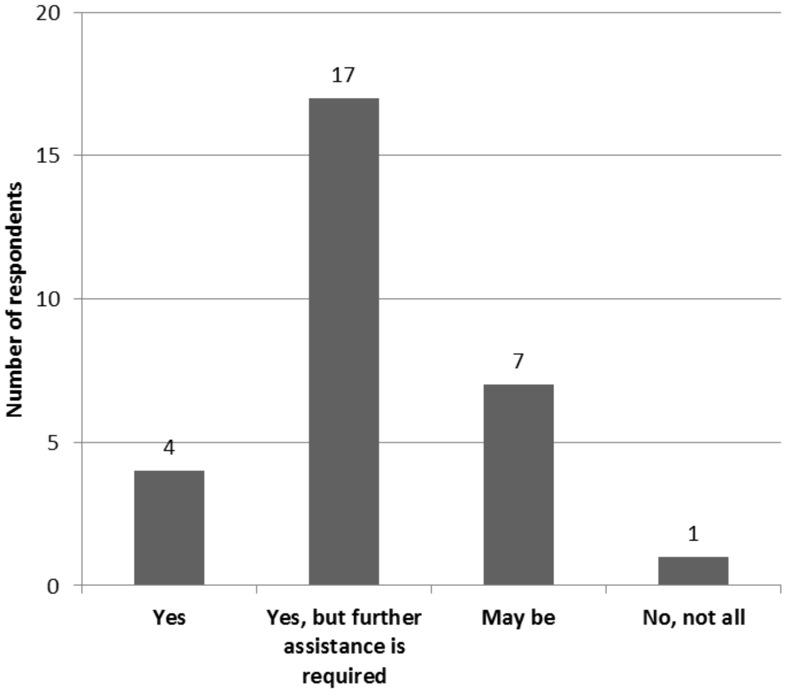


Figure 11-9 Responses regarding the overall usability of patterns in achieving currently in use industry standards

Again the recommendations for further support is higher in number as 61% think that further assistance is necessary along with patterns while 25% are partially agreed and only 11% are full favour. One useful comment on this question is:

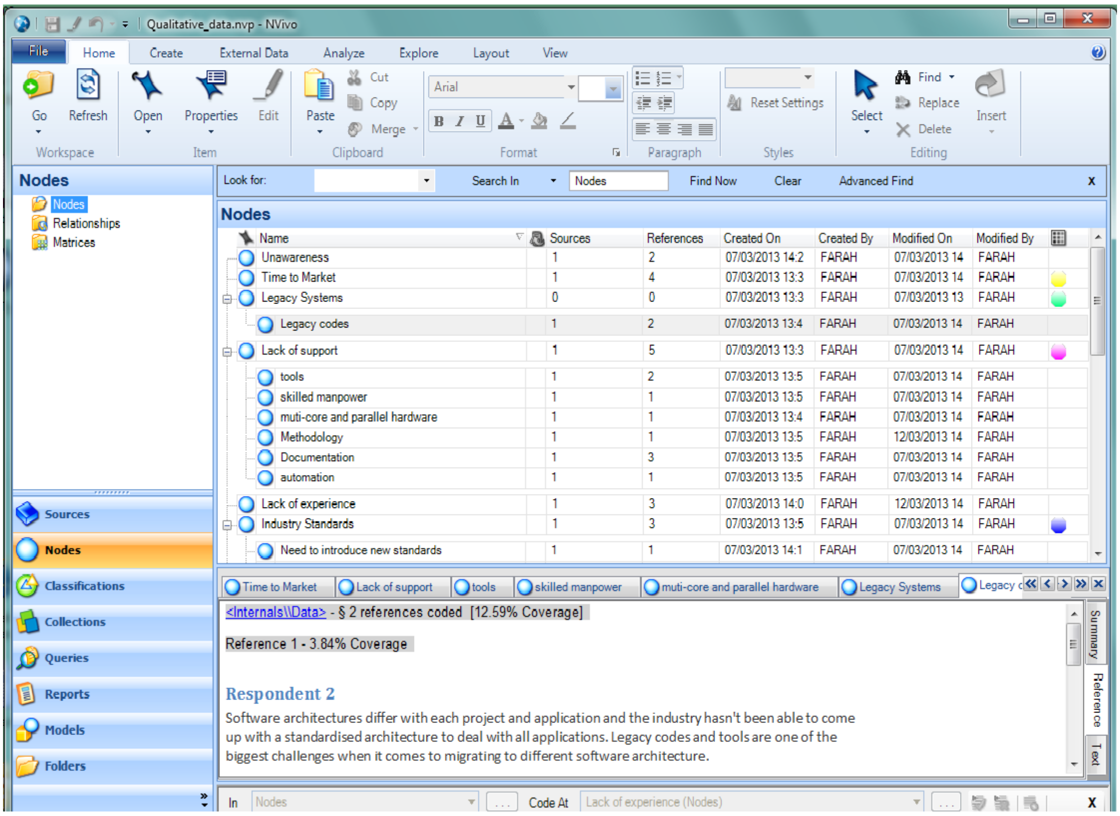
"I think the patterns help the confidence in the initial design choice and implementation but it would be misleading to say that using them would make you compliant - there's so much paperwork (process) that is involved and needs to be covered".

Overall the comments and results obtained have provided a useful view of the real stakeholders and a further discussion on these is presented in the next section.

11.6.4. Industry and the challenge of migration

An open-ended question was included in the questionnaire to obtain the feedback of respondents on how the industry is currently coping with the problem of migration between different software architectures and what the challenges are around it. All of the comments received from the respondents are included in Appendix D. The comments received by most of the respondents are useful and self-explanatory however it was important to apply some text analysis techniques to conclude results from the textual data.

To interpret results from the submitted responses some of the text analysis techniques mentioned in Section 11.5.4 are used to analyse the comments provided by the respondents. In this regard the QSR NVIVO software tool is used for the text analysis and original comments are distributed into different nodes as shown in Figure 11-10.



Nodes

Name	Sources	References	Created On	Created By	Modified On	Modified By	
Unawareness	1	2	07/03/2013 14:2	FARAH	07/03/2013 14	FARAH	
Time to Market	1	4	07/03/2013 13:3	FARAH	07/03/2013 14	FARAH	●
Legacy Systems	0	0	07/03/2013 13:3	FARAH	07/03/2013 13	FARAH	●
Legacy codes	1	2	07/03/2013 13:4	FARAH	07/03/2013 14	FARAH	
Lack of support	1	5	07/03/2013 13:3	FARAH	07/03/2013 14	FARAH	●
tools	1	2	07/03/2013 13:5	FARAH	07/03/2013 14	FARAH	
skilled manpower	1	1	07/03/2013 13:5	FARAH	07/03/2013 14	FARAH	
muti-core and parallel hardware	1	1	07/03/2013 13:4	FARAH	07/03/2013 14	FARAH	
Methodology	1	1	07/03/2013 13:5	FARAH	12/03/2013 14	FARAH	
Documentation	1	3	07/03/2013 13:5	FARAH	07/03/2013 14	FARAH	
automation	1	1	07/03/2013 13:5	FARAH	07/03/2013 14	FARAH	
Lack of experience	1	3	07/03/2013 14:0	FARAH	12/03/2013 14	FARAH	
Industry Standards	1	3	07/03/2013 13:5	FARAH	07/03/2013 14	FARAH	●
Need to introduce new standards	1	1	07/03/2013 14:1	FARAH	07/03/2013 14	FARAH	

<Internals\\Data> - § 2 references coded [12.59% Coverage]

Reference 1 - 3.84% Coverage

Respondent 2

Software architectures differ with each project and application and the industry hasn't been able to come up with a standardised architecture to deal with all applications. Legacy codes and tools are one of the biggest challenges when it comes to migrating to different software architecture.

Figure 11-10 A snapshot taken from Nvivo software to show distribution of comments into various nodes

Related to the research question and analysing the respondents' comments the main nodes are identified as:

- Unawareness
- Time to market
- Legacy systems
- Lack of support
- Lack of experience
- Industry standards
- Cost
- Agile methods
- Adapting architectures

Child nodes are created under ‘Lack of Support’ and ‘Industry standards’ nodes. Detailed distribution of comments under each node/child node is shown in Table 11-1 below:

Time to Market

Respondent 4
Most businesses are concerned with a quick as possible time to market. Migration across software architectures raises this significantly. It can also often be hard to convince others that a different way of thinking can benefit them or the business as a whole.

Respondent 7
I think that people working in the industry (I am speaking about the managers and big boss) tend to keep traditions. I think it is really a hard task to convince big managers to change their way of work that have been adopted for a number of projects (and years). Big managers have fear of switching to other options as their main goal is reaching time to market.

Respondent 9
Thorough background knowledge of software architectures is required in order for successful migration. Some companies are not willing to explore on the new or different architectures due to several factors: [1] cost - companies are not willing to buy new tools to support different architecture [2]lack of manpower - developers need to quickly adapt to new architecture. Due to current economic climate, companies are not willing to hire new people to explore on the newer or different architecture. [3] time to market - by adopting new architecture, time to market will be increased due to learning curve that developers need to endure.

Respondent 24
Consider a company which is using event triggered architecture for last 10 years with well-trained staffs. It is bit difficult for them to migrate into TT within a short period. Challenges they may have, 1) Cost 2) Short period 3) Need to train current staffs or need to recruit experienced people.

Legacy codes

Respondent 2
Software architectures differ with each project and application and the industry hasn't been able to come up with a standardised architecture to deal with all applications. Legacy codes and tools are one of the biggest challenges when it comes to migrating to different software architecture.

Respondent 18
I have just been involved in writing a proposal where the architecture has moved from co-operative to pre-emptive and now porting parts back to co-operative (interim legacy support). There's no easy way to set about doing this, especially as libraries expect pre-emptive threads. Generally the industry knows that the CPU isn't getting faster and that multi thread/process/processor is the way forward but that it is hard to do. Functional languages are suggested but that has never really taken off. Clever additions to the language are being tried, but it’s always going to be hard - the emphasis is pre-emptive rather than co-operative or hybrid. I don't see that changing soon.

Lack of support

Respondent 8
Coping badly

Respondent 15
Present ET architecture is familiar to us , TT arch implementation new and no explanation on that and no support

Respondent 16
This migration requires expertise and good software development practices. Sometimes, this migration might not be that straight forward at all and might require scrapping the previous software design and developing it from scratch. I think the industry needs to be more educated on this problem.

Respondent 18
I have just been involved in writing a proposal where the architecture has moved from co-operative to pre-emptive and now porting parts back to co-operative (interim legacy support). There's no easy way to set about doing this, especially as libraries expect pre-emptive threads. Generally the industry knows that the CPU isn't getting faster and that multi thread/process/processor is the way forward but that it is hard to do. Functional languages are suggested but that has never really taken off. Clever additions to the language are being tried, but it’s always going to be hard - the emphasis is pre-emptive rather than co-operative or hybrid. I don't see that changing soon.

Respondent 28
It is a very mixed bag depending on the coders background. One big issue I find is a lot of people new to writing embedded C starting learning to code for PCs and so have never had to think about issues like CPU or memory resources and lack the understanding of the electronics in the microcontroller to debug issues. This can result in rather "heavy" and unreliable code (littered with ISRs which can create unpredictable behaviour).

Lack of support: tools

Respondent 19

Migration is very poorly done due to lack of knowledge and lack of good migration tools. The migration process itself is usually completely or partially manual, and error prone.

Respondent 25

Industry is not ready to move from the classical ET architecture until and unless some really excellent tools help them to migrate easily. Certainly Patterns can be a help but need lot of automation in the process.

Lack of support: skilled manpower

Respondent 9

Thorough background knowledge of software architectures is required in order for successful migration. Some companies are not willing to explore on the new or different architectures due to several factors: [1] cost - companies are not willing to buy new tools to support different architecture [2]lack of manpower - developers need to quickly adapt to new architecture. Due to current economic climate, companies are not willing to hire new people to explore on the newer or different architecture. [3] time to market - by adopting new architecture, time to market will be increased due to learning curve that developers need to endure.

Lack of support: multicore and parallel hardware

Respondent 6

I don't see migration away from interrupts in telecom. The problems associated with multi-core and parallel hardware need solution and migration more urgently than event-driven system challenges.

Lack of support: methodology

Respondent 21

In my opinion, migrating software architecture from one to another requires substantial effort and cost, and often relies on experience engineers to do it manually. Companies are usually reluctant to invest such a large amount of money to do the software architecture migration, unless a good profit return is envisaged. Of course, a methodology that can simplify the migration process is desirable; since it will maximise the profit margin if software architecture migration is necessary.

Lack of support: documentation

Respondent 5

People of industry (not all) are unaware about it but it still requires some more clarification regarding TT and ET i.e. how to apply TT in the system? How am I going to migrate? Do I really need to migrate? If my system is running perfectly without any huddles why should I need to migrate? (However all the comments may be useful/not-useful because I am not a very highly experienced person I have 2 years of experience of automation and control industry).

Respondent 14

Understanding "tribal knowledge" not captured in the formal documents of a system.

Respondent 19

Migration is very poorly done due to lack of knowledge and lack of good migration tools. The migration process itself is usually completely or partially manual, and error prone.

Lack of support: automation

Respondent 25

Industry is not ready to move from the classical ET architecture until and unless some really excellent tools help them to migrate easily. Certainly Patterns can be a help but need lot of automation in the process.

Industry standards

Respondent 2

Software architectures differ with each project and application and the industry hasn't been able to come up with a standardised architecture to deal with all applications. Legacy codes and tools are one of the biggest challenges when it comes to migrating to different software architecture.

Respondent 11

Plan migration and portability at design time by implementing firmware to be compliant with recognized company or industry wide standards such as AUTOSAR.

Respondent 13

Migration between software architectures is not easy in aerospace industry because of the cost involved in rigorous verification to prove the new changes are airworthy and getting the changes certified by authorities (European Aviation Safety Agency EASA).

Industry standards: need to introduce new standards

Respondent 2

Software architectures differ with each project and application and the industry hasn't been able to come up with a standardised architecture to deal with all applications. Legacy codes and tools are one of the biggest challenges when it comes to migrating to different software architecture.

Cost

Respondent 9

Thorough background knowledge of software architectures is required in order for successful migration. Some companies are not willing to explore on the new or different architectures due to several factors: [1] cost - companies are not willing to buy new tools to support different architecture [2]lack of manpower - developers need to quickly adapt to new architecture. Due to current economic climate, companies are not willing to hire new people to explore on the newer or different architecture. [3] time to market - by adopting new architecture, time to market will be increased due to learning curve that developers need to endure.

Respondent 21

In my opinion, migrating software architecture from one to another requires substantial effort and cost, and often relies on experience engineers to do it manually. Companies are usually reluctant to invest such a large amount of money to do the software architecture migration, unless a good profit return is envisaged. Of course, a methodology that can simplify the migration process is desirable; since it will maximise the profit margin if software architecture migration is necessary.

Respondent 24

Consider a company which is using event triggered architecture for last 10 years with well-trained staffs. It is bit difficult for them to migrate into TT within a short period. Challenges they may have, 1) Cost 2) Short period 3) Need to train current staffs or need to recruit experienced people.

Agile methods

Respondent 12

The main challenge from my point of view is that the industry is very much plan-driven and nowadays you have to respond to change more quickly. So more agile approaches are needed to create the software. In many cases, we use subcontractors and communicating the domain knowledge and the architecture to them is troublesome. For me, it is not very important if the system is time-triggered or event-triggered. Currently we are developing in event-driven fashion and no plans to change that.

Respondent 27

I believe that industry is moving towards Agile Model Driven Development (AMDD). (http://www.agilemodeling.com/essays/amdd.htm)

Adapting architectures

Respondent 1

Getting the whole organisation or team to buy into a migration strategy/process is a big problem. It could be done in stages and within smaller teams before the process is finally adopted by the whole organisation

Respondent 4

Most businesses are concerned with a quick as possible time to market. Migration across software architectures raises this significantly. It can also often be hard to convince others that a different way of thinking can benefit them or the business as a whole.

Respondent 17

Migration between software architectures is a time consuming task and typically not interesting for engineers who are focused on implementing real world solutions. With the need to integrate embedded systems with other systems becoming more and necessary engineers are expected to know many software architectures particularly those centred around the Internet.

Lack of experience

Respondent 5

People of industry (not all) are unaware about it but it still requires some more clarification regarding TT and ET. i.e. how to apply TT in the system? How am I going to migrate? Do I really need to migrate? If my system is running perfectly without any huddles why should I need to migrate? (However all the comments may be useful/not-useful because I am not a very highly experienced person I have 2 years of experience of automation and control industry).

Respondent 24

Consider a company which is using event triggered architecture for last 10 years with well-trained staffs. It is bit difficult for them to migrate into TT within a short period. Challenges they may have, 1) Cost 2) Short period 3) Need to train current staffs or need to recruit experienced people.

Respondent 28

It is a very mixed bag depending on the coders background. One big issue I find is a lot of people new to writing embedded C starting learning to code for PCs and so have never had to think about issues like CPU or memory resources and lack the understanding of the electronics in the microcontroller to debug issues. This can result in rather "heavy" and unreliable code (littered with ISRs which can create unpredictable behaviour).

Table 11-1 Distribution of comments under various nodes

From the comments mentioned above it is observed that currently people in the industry are aware of the fact that migration between different software architectures is a huge challenge but at present there is no sophisticated way adopted to cope with this challenge. However two of the respondents (respondent 22 and 23) showed their unawareness about the issue.

With the help of Nvivo tool a graphical analysis of the text data is achieved and the percentage cover by each node is given in Figure 11-11.

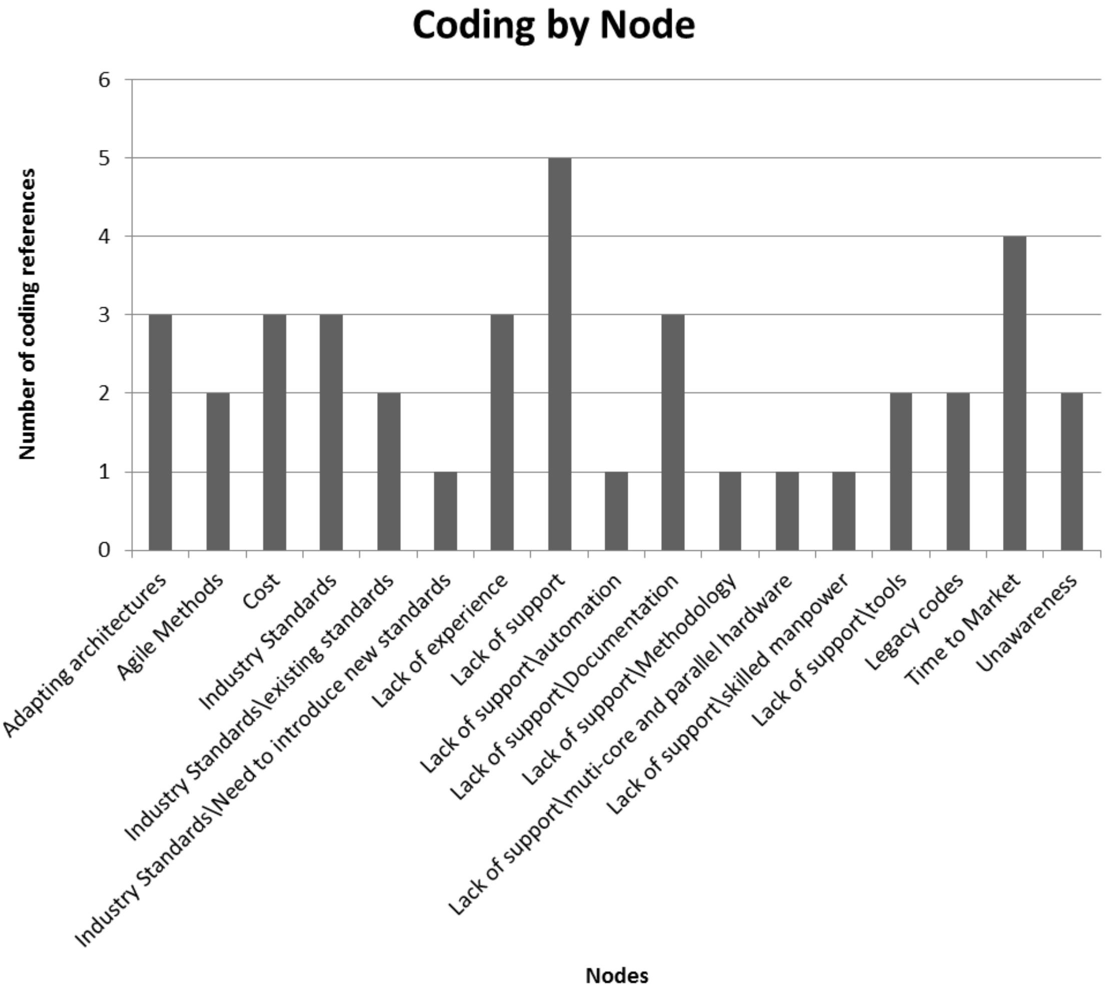


Figure 11-11 Illustrating the number of coding references cover for each node

This coding analysis of text data obtained through respondent’s comments has revealed some interesting results. As emphasized by the respondents shown in Figure 11-11 ‘Lack of support’ appeared as the most important issue during the migration process. Further this lack of support is more towards

‘Lack of documentation’ and the ‘tool support’. The other areas where practitioners feel that industry lack support are towards ‘Methodology’ which should be adopted while migrating architectures, ‘Automation’ that must be required during the various stages and support for ‘Multi-core and parallel hardware’ designs.

The other main challenge indicated is the ‘Time to Market’. ‘Time to Market’ is one of the most important design metrics and is considered as the time required to develop a system to the point that it can be released and sold to customers (Vahid and Givargis, 2002) . In this sense it is directly related to business profit and reputation of the organisation. As the respondent 4 has mentioned that migration across architectures may increase ‘time to market’ and managers at higher level always think in terms of gaining more profit for the organisation. Exactly the same opinion is observed from the comments provided by respondent 7 and respondent 9 and respondent 24. Overall it is concluded that people in industry are reluctant to any change in existing architectures unless required by the customer as migration might not be a favourable decision in terms of organisation’s repute and business profit.

As ‘Time to Market’ is directly related to ‘Cost’ so the next important concern shown by the practitioners is the involvement of ‘Cost’ for industrial projects which are required to pass through the process of migration. The cost might be in form of outsourcing manpower or tools required during the process and therefore avoided unless a reasonable profit return is envisioned by a team of experts in the organisation. Therefore ‘Adapting architectures’ is not favoured by the industry professionals because of the lack of support, high time to market concern and cost involved in the process. However, certain respondents talked about 'Industry standards' such as AUTOSAR (Automotive Industry Open System architecture) which provides firmware to support migration at design time. Respondent 7 from the aerospace industry has mentioned his concerns about meeting the safety standards set by European Aviation Safety Agency 'EASA' and making any changes would require rigorous testing which in turn could incur cost overheads. It is also emphasized that industry need to introduce new standards. Few respondents (Respondent 12 and Respondent 27) mentioned about following agile approaches to software development. This truly makes sense as agile practices bring a number of business benefits as better project adaptability and reaction to changes, reduced production cost and better performance (Bozheva and Gallo, 2010)

11.7. Discussion

This chapter has described the evaluation of patterns in the industrial context which is attempted to determine the current practices in dealing with the migration process and the potential of the proposed pattern collection in real practice. The overall results obtained through this process have provided useful insights on the major issues that people in industry are mainly concerned with. It is apparent that there is no standard approach to deal with the issues however a commonality was observed on the issues themselves. A majority of the respondents expressed concerns about cost considerations, time to market and expertise in adapting the software architectures for applications.

The industry has appreciated the idea of the use of patterns in assisting the working force of professionals to deal with the challenges of migration, but at the same time recommended further support in addition to the documented pattern collection. This further support would ideally be in the form of a framework for migration which could lead to a tool support or an expert system of migration which can intelligently help in taking the appropriate decisions. However the cost involved in providing such support is an additional factor to discuss at this stage.

The evaluation process also helped in obtaining useful comments for improving or enhancing the pattern collection. The responses obtained have also highlighted the need for patterns for multiprocessor instead of single-processor applications as an increasing number of the application designs nowadays make use of more than one processor. It is also apparent from the survey that the optimisation section of the pattern collection needs to be enhanced further with more information and deeper study. One engineer from the aerospace industry mentioned that *"the scheduler patterns are useful but they need to be further classified as fixed-priority or non-fixed-priority along with support for sporadic tasks"*. With regards to achieving industry standards for the aerospace industry one senior engineer mentioned that *"the techniques described are relevant and the pattern clearly explain the considerations, however the target audience appear to be someone with limited experience of* *designing software to achieve DO-178b compliance".* These comments are considered as fairly judgmental as the proposed pattern collection is not functionally complete in its present state and needs further expansion for its wider implications in the industrial context.

At this stage it is important to mention that design patterns previously developed for creating new embedded applications based on time-triggered architecture (Pont, 2001) have been used in a range of research projects (Mwelwa, 2006; Kurian and Pont, 2007; Bautista-Quintero and Pont, 2008; Hughes and Pont, 2008) however these patterns were not passed through a rigorous evaluation process as described for the PMES collection in Chapter 9, 10 and 11. Also, most of the patterns have been subjected to preliminary evaluation process at various PLoP conferences mentioned in Section 10.4.

As a matter of fact it is almost impossible to carry out a rigorous evaluation for huge pattern collections such as Alexander's pattern for architecture design (Alexander, Ishikawa *et al.*, 1977) , patterns for time-triggered embedded system (Pont, 2001) and patterns for fault-tolerant design (Hanmer, 2007). For such huge collections it is even not practical to carry out a preliminary evaluation of patterns such as shepherding and writer's workshop discussed in Section 6.8 as it involves substantial amount of time and cost incur in participation of such events. For enthusiastic pattern authors still remain the option of testing patterns in more than one example of a problem and the more examples tested the better.

11.8. Conclusion

The evaluation process described in this chapter was designed to obtain feedback from the community of working professionals in embedded development on the issues of migration between software architectures. The evaluation process also helped to gain some useful feedback on the proposed collection of design patterns their usability, shortcomings and further expansion and improvements. Overall the professional community has appreciated the idea of the use of design patterns in migrating between different software architectures for embedded applications as there is currently no standard approach to follow in this field. The majority of the practitioners have recommended for further support along with the documented patterns and few have recommended further expansion in the current collection.

CHAPTER 12. CONCLUSIONS AND FUTURE WORK

12.1. Reasons and motivation for the thesis work

The work described in this thesis is concerned with improving the reliability of complex embedded applications by introducing time-triggered designs into their architectures in place of existing event-triggered designs.

It has been argued that time-triggered designs are the preferred choice for many embedded applications and especially for high-integrity and safety critical applications because of their predictable nature compared with event-triggered designs. In spite of this, event-triggered designs are sometimes preferred because of the flexibility they offer in their designs and the high responsiveness in normal situations. Such a choice however may end up with failure under peak load situations and unpredictable behaviour resulting in the need for a complete or partial change in the design architecture. In such a situation migration to time-triggered designs can contribute to higher reliability and applications with more predictable behaviour. However, carrying out the migration of architectures in complex embedded applications is not an easy task and there are a number of issues which must be considered during and after the migration process. One of the most important challenges is in finding ways of supporting developers during the migration process which would result in time-saving and improved efficiency in what is otherwise a rather tedious process.

Design patterns provide the best practice solution to commonly recurring problems in a particular domain. The outcomes of the research into the use of design patterns in embedded software development have proven to be very effective. However, most of the work to date in this field has addressed system construction rather than migration that is to say in the process of designing applications from scratch. The motivation for the research described in this thesis is to address this gap in the re-design process in the area of system migration.

To this end the work presented in this thesis aimed to overcome some of the difficulties faced by the developers in the migration process of embedded applications. This has been achieved by helping them in the choice of appropriate time-triggered designs and overcoming further difficulties such as the handling of long task problems in co-operative designs, and shared resource access in pre-emptive designs. In addition this work has provided a few generic patterns that aim to enable the designer to optimise the accomplished time-triggered designs after migration.

To summarise, this thesis has identified the lack of a standard approach in the use of design patterns in the process of migrating between the different architectures of embedded applications. This research is therefore aimed to provide support to the practitioners in this field by introducing a new pattern language to assist in the migration process from event-triggered to time-triggered designs. To the best of the author’s knowledge, this study represents the first attempt to use design patterns for the purposes of such complex system (architecture) migration.

12.2. A review of the contributions

This section reviews the key contributions of the research presented in this thesis and discusses the extent to which the initial aims of the research were achieved.

12.2.1. Migration from ET to TT designs

The work presented in this thesis started by giving a very brief introduction to the world of embedded systems design and the commonly used software architectures used to design embedded applications. CHAPTER 1 was specifically focused on a description of the problem this research was addressing during the course of the study. CHAPTER 2 then described three example applications described in the literature which has provided the inspiration for the work carried out during the research. The examples have demonstrated how changes in the software architecture can bring improvements in system performance and reliability. CHAPTER 3 has described the overall methodology adopted to carry out the research.

CHAPTER 4 gave an overview of some of the possible scheduling schemes and the two main software architectures that are the preferred choice for developers when designing embedded applications. There followed a discussion on the comparative features of both architectures and the trade-offs involved within each architecture. It was shown that for systems which are designed for worst-case requirements such as hard real-time systems, the time-triggered architecture is a better choice if reliability is a major concern. CHAPTER 5 has presented an account of the causes of migration in embedded software, and dependencies between different software components in embedded applications. CHAPTER 5 also provided a brief coverage of some existing techniques described in the literature to carrying out migration of software architecture in embedded applications.

12.2.2. Design patterns to support migration

CHAPTER 6 has introduced design patterns and their adoption in various fields. It has also explored the notion that patterns have the potential to provide support to developers of embedded applications in the complicated process of migration under consideration.

As mentioned in Section 12.2.1, CHAPTER 2 explored example applications which have passed through a migration of architecture during their life-cycle. The examples demonstrated that migration in complex embedded applications always incurs overheads in terms of time, effort and the cost required to make the necessary changes. These overheads may reach seriously high values if the designers/developers are not fully aware of the architectural design issues that could arise during the application migration process. At this stage a set of guidelines and a set of proven and tested techniques by experts can play a vital role in assisting developers resulting in a saving of time, effort and cost. CHAPTER 7 started by explaining the rationale for introducing the ‘PMES’ collection of design patterns which is the key contribution to this research. The chapter then described the pattern mining process and how new patterns for migration were derived and developed with logical grounds for each individual pattern. CHAPTER 8 then introduced each of the newly documented patterns. At the current stage, this collection contains 23 patterns that can help in the migration process initially by guiding the users towards determining whether their application is the right candidate for time-triggered designs.

12.2.3. Assessment of the ‘PMES’ collection

The central claim in this research is that the proposed pattern collection will support developers of embedded applications in the process of migrating applications. It is therefore important to investigate to what extent this claim is justified. For this purpose a detailed assessment of the patterns is conducted as described in CHAPTER 9, CHAPTER 10 and CHAPTER 11.

In the first stage, studies were carried out to evaluate the performance of applications after transforming their existing event-triggered designs to time-triggered design and the results are described in CHAPTER 9. These transformations were carried out not only to demonstrate the improvement in performance of the applications, but also to illustrate how, and at what stages the various patterns in the PMES collection can help to achieve the transformation.

In the second stage, two empirical studies were designed involving MSc. students as subjects. The first study was designed to test the effectiveness of patterns when employed by experienced developers/designers. The results suggested that applications of patterns can help developers to take appropriate decisions during the migration of complex embedded applications, and to produce reliable systems after migration. The second study was designed to demonstrate the usability of patterns for inexperienced developers. The results suggest that patterns provide an effective way of representing information which is easier to understand, digest and implement when compared with other resources such as text books, research papers and other supplementary material.

In the third stage the research was evaluated in the industrial context to obtain feedback from real stakeholders. This is achieved by using a web-based survey instead of interviewing in order to cover a wider audience from different parts of the world. The feedback obtained as a result of this process provided useful insights for making improvements, enhancing the current collection and providing back up for further support.

12.3. Research implications and shortcomings

Patterns in the PMES collection are applicable to a wide range of systems. For example, the advent of new industry standards such as ISO 26262 for the functional safety of passenger cars presents many new challenges to organisations in the automotive sector. Some of the principles defined for the software architectural design in ISO 26262 are as follows:

- Hierarchical structure of software components
- Restricted size of software components
- Restricted coupling between components
- Appropriate scheduling properties
- Restricted use of interrupts.

Some mechanisms are also recommended for error detection and error handling at the software architectural level. These issues are covered by patterns in the collection. For example, appropriate scheduling properties are discussed in the pattern TT SCHEDULER and restricted use of interrupts is emphasize in the pattern EVENTS TO TIME. The pattern BALANCED SYSTEM is focused on techniques to keep the execution time of the software components fixed and restricted and the pattern SYSTEM MONITORS has discussed error detection and correction techniques.

The detailed evaluation of the research has helped in identifying the wider implications and shortcomings of the proposed pattern collection. The people involved in the survey from industry indicated that migration between software architectures is a difficult task, and there is no standard approach at present to cope with the challenges it presents. Therefore it is contended that a collection of design techniques documented as design patterns can be viewed as an aid to the developers' community especially for those who are less experienced. It is indicated by the feedback obtained from the practitioners during the third phase of the evaluation, that industry encouraged the use of design patterns especially in companies where design/development resources are stretched and hence documentation tends to be a much needed but neglected resource. About 32% respondents agreed that such patterns are very useful, while 68% suggested that they can be more useful with some kind of further support such as an automated tool. Professionals in industry agreed that the patterns provide an architectural insight into what needs to be addressed when attempting to consider alternative architectures, and/or when shifting between architectures, but at the same time asked for more practical knowledge to be provided in the area to ascertain their sole use. It is clear from this work that professionals in industry are seeking a set of excellent tools which could help to automate the process of migration.

12.4. Scope for future work

There are number of areas of future development based on the research described in this thesis and these are described below.

12.4.1. Multi-processor designs

The work presented in this thesis was focused on issues related to the migration of embedded applications based on a single processor. This is a limitation as embedded applications are becoming more complex and tend to use multi-processor designs. Therefore, future work will need to be extended to applications based on multi-processor designs. With this expansion there is a lot of scope to explore patterns related to the handling of communication between different nodes and messaging services.

12.4.2. Completeness of the pattern language

For single processor applications, the proposed collection of patterns has gaps and is focused only on the core issues such as the selection of appropriate schedulers in time-triggered designs, handling of long task problems in co-operative designs and shared resource access in pre-emptive designs. There are also number of other issues which could rise during the migration process such as response time degradation in moving from event-triggered to time-triggered designs which are not considered yet. Finally there is a lot more scope pertaining to the expansion of the pattern language morphologically in which patterns fit together to form a complete structure without any gaps.

12.4.3. Exploration of implementation examples

Another interesting extension of the current work would be to explore and document pattern implementation examples (PIEs) for all the newly proposed patterns. This will further enhance the collection for a wide range of hardware and software applications.

12.4.4. Formalising the pattern language

One more challenging direction for further expansion of this research is the formalisation of the proposed pattern collection. Formal specification of design patterns are not meant to replace the existing textual/graphical description but rather to complement them to achieve well-defined semantics, allow rigorous reasoning about them, and facilitate tool support (Taibi and Ling Ngo, 2003). The formalisation strategies are either based on rigorous mathematical formulae (Mikkonen, 1998; Eden, 2000; Mak, Choy *et al.*, 2003; Taibi and Taibi, 2006) or the use of the Unified Modeling Language (Kim, France *et al.*, 2002; Mapelsden, Hosking *et al.*, 2002) to represent patterns or pattern solutions. Therefore it is proposed that a notation for formalising the pattern language and production rules can be helpful for an automated tool generation for migration between software architectures.

12.4.5. Computer assisted tool support

In the present state, the patterns are in a documented form and so can only be applied manually. An intelligent software tool that is capable of assisting the users in choosing appropriate time-triggered designs in the migration process is far more promising. Such a tool support would identify the constant and variable elements of a pattern solution. The constant factors can provide the template framework for example conversion from event-triggered to time-triggered. The variable factors can be queried from the user to incorporate into the code generation process. Also to further facilitate the process, the tool may be able to provide packaged software components to use based on design techniques discussed in some of the patterns for example BUFFERED OUTPUT, POLLED INPUT and TASK GUARDIAN.

12.5. Final conclusions

Overall, the work described in this thesis has made three major contributions: First, it presents the novel idea of using design patterns for managing the complexities involved in migrating between different software architectures of embedded applications. Second, it has introduced a new pattern language (called PMES) to support the migration from event-triggered to time-triggered architectures. Finally evidence has been presented that the proposed pattern language can be applied to a range of applications and bears the potential to assist designers and developers in the field. In this regard the feedback from the industry is also very encouraging.

The migration between different architectures of embedded applications is almost a research discipline in itself, and this research is at an early stage of a novel and increasingly important branch of this discipline. Hopefully the work described will provide the necessary inspiration for further research progress in this critical area.

PART D: APPENDICES

- **Appendix A:** Pattern forms.
- **Appendix B:** Full specification of newly documented patterns in the PMES collection
- **Appendix C:** Exercises used in empirical studies for the assessment of the PMES collection
- **Appendix D:** Documents used in industrial evaluation of the PMES collection

APPENDIX A: PATTERN FORMS

Alexandrian form

PATTERN NAME

A picture that shows a typical example of the architectural pattern.

Introductory paragraph that sets the context of the pattern

♦♦♦

Headline in bold that gives the essence of the problem (1-2 sentences)

Body of the problem (longest section of the pattern)

Solution in bold that is always stated in the form of an instruction

Diagram illustrating the solution graphically

♦♦♦

Resulting context (that connects the pattern to smaller patterns in the language that are required to complete the pattern)

Modified Alexandrian form

PATTERN NAME

Introductory paragraph that sets the context of the pattern

♦♦♦

Headline in bold that gives the essence of the problem (1-2 sentences)

Body of the problem (longest section of the pattern)

Solution in bold that is always stated in the form of an instruction

Example of use

♦♦♦

Resulting context (that connects the pattern to smaller patterns in the language that are required to complete the pattern)

GoF form

PATTERN NAME and CLASSIFICATION

PATTERN NAME: A concise name of the pattern. This needs to be carefully selected because it will eventually become a component of the design vocabulary.

Classification: Creational, Structural, Behavioural

- **Intent**

A short statement about what the design pattern does/ the problem it addresses/ the rationale behind the pattern

- **Also Known as**

Other names for the pattern (if any)

- **Motivation**

A scenario that illustrates a design problem and how the structures in the pattern solve the problem

- **Applicability**

This section presents situations where the design pattern can be used

- **Structure**

A graphical representation of the classes in the design pattern

- **Participants**

The classes and/or objects participating in the design pattern (and their responsibilities).

- **Collaborations**

This section describes how the participants mentioned in the previous section carry out their responsibilities.

- **Consequences**

What are the pros and cons of using the pattern? How does the pattern supports this objectives?

- **Implementations**

Hints and techniques for implementing the pattern

- **Sample Code**

Code fragments that illustrate how the pattern might be implemented.

- **Known Uses**

This section presents examples of the pattern encountered in real systems.

- **Related Patterns**

This section presents other patterns that are strongly related to the pattern, and the important references between the patterns.

Coplien form (after James Coplien)[22]

PATTERN NAME

- **Problem**

Problem addressed by the pattern

- **Context**

Situations/settings where the pattern is applicable

- **Forces**

Addresses the issue: "What makes the pattern so difficult?". Commonly a list of bullet points.

- **Solution**

The solution to the problem – this is generally layered with the most general interpretation at the highest level and providing more detail as one progresses through the section.

- **Resulting context**

The pros and cons of the pattern.

- **Rationale**

This section describes how the pattern works, why it works and why it is "good".

[22] Source: From James Coplien's patterns listed in "A Generative Development – Process Pattern Language" that appears in Rising, L. (editor), "The Patterns Handbook: Techniques, Strategies and Application", (Cambridge University Press, 1998)

POSA form[23]

PATTERN NAME

The name and a short summary of the pattern

- **Also known as**

Other names for the pattern, if any exist

- **Example**

A real-world example that demonstrates the existence of the problem and the need for the pattern. This helps to explain the problem and related forces better.

- **Context**

Situations / settings where the pattern is applicable.

- **Problem**

Problem addressed by the pattern including a discussion of the related forces.

- **Solution**

Fundamental solution to the underlying problem.

- **Structure**

Discusses the structural aspects of the pattern

- **Dynamics**

Typical scenarios describing the run-time behaviour of the pattern.

- **Implementation**

These are guidelines for implementing the pattern.

- **Example resolved**

Further aspects for resolving the example not discussed in earlier sections are presented here.

- **Variants**

Very brief descriptions of alternative solutions to the pattern.

- **Known uses**

Examples of the use of the pattern, taken from existing systems

- **Consequences**

Pros and cons of applying the pattern

- **See also**

References to related patterns

[23] This pattern template was first used by Buschmann et al. (1996) in their book Pattern-Oriented Software Architecture (Volume 1) It is very similar to the structured and detailed style of the GoF pattern template.

PTTES form

PATTERN NAME

- **Context**

Summarizes the situations where the pattern may be useful. This is normally in the form of a bulleted list.

- **Problem**

Provides a brief summary of the problem addressed by the pattern.

- **Background**

Provides information that will help less experienced developers make full use of the pattern.

- **Solution**

Describes one or more solutions to the problem addressed by the pattern. This may include software designs, code listings and/ or hardware schematics

- **Hardware resource implications**

Every pattern provides and/or consumes hardware resources. This section helps to balance the need for, and provision of, these resources.

- **Reliability and safety implications**

Many patterns have potential reliability and safety implications: such issues are discussed in this section.

- **Overall strengths and weaknesses**

This section considers issues involved in porting the pattern to a different microcontroller.

- **Related patterns and alternative solutions**

Discusses alternative solutions to problems and provides references to other, related patterns that may be of interest.

- **Examples**

At least one example of the application of the pattern is given here.

- **Further reading**

Gives suggestions for sources for additional information that the reader may wish to look at while (or after) reading the pattern

APPENDIX B: SPECIFICATIONS OF THE PMES COLLECTION

TIME FOR TT

Abstract pattern

Context

- You already have at least a design or prototype for your system based on some form of Event-triggered architecture.
- You are in the process of creating or upgrading an embedded system, based on a single processor unit.

Reliable system operation is a key design requirement.

Problem

Should you use a TT architecture in your system?

Solution

Some systems are obvious candidates for TT architectures. These systems involve periodic data sampling or data playback, or other periodic activities.
Some simple examples:

- Data acquisition and sensing systems (for example, environmental systems for temperature monitoring) usually involve making data samples on a periodic basis. Some cases (high-frequency systems) may involve making millions of samples per second: other cases (e.g. temperature monitoring at a weather station) may involve making one sample per hour. Whatever the rate, a TT architecture will usually be used in order to guarantee high-quality ("jitter free") data sampling at a known signal-to-noise ratio.

- Control systems (for example, primary flight control in an aircraft or helicopter, cruise control in a passenger car, temperature control in an industrial furnace, control of a hard disk in a computer). Such systems all involve three core – periodic – activities: measuring some aspect of the system to be controlled (e.g. the furnace temperature), calculating changes required to the control system (e.g. calculating new gas burner settings) and applying the changes to the control system (e.g. updating the settings on the gas burner). Use of a TT architecture will ensure high-quality control without jitter in the input or output.

- Data output systems (for example, music players, video playback, head-up displays) are required to generate output signals at precise times (for example, "CD quality" sound will be played back at 44,400 samples per second. Any jitter in the playback will result in degradation of the music quality.

It is important to appreciate that – in many of these cases - use of a TT solution allows the system to perform the above periodic activities **and also perform other functions** (such as reading switches, updating displays, receiving data over serial communication links, performing calculations, etc) **without interfering in any way** with the processing outlined in the above examples. It is the ability to perform multiple tasks and still **guarantee that critical tasks will always execute as required** that makes a TT solution so attractive to developers of high-integrity, safety-related and safety-critical systems.

Clearly, not all systems fall into the “periodic sampling / playback” category. In particular, if your system must respond only to events which may occur at “random” times, it may not be a good match for this architecture.

For example, consider a simple radio transmitter which is used to open your garage doors a few times a week. We could use TT architecture to poll the switch on this system every 20 ms (just in case the switch has been pressed). However, while such a solution would undoubtedly work, it would be likely to have a shorter battery life than a simple event-triggered design which operates in power down mode except when the switch on the unit was pressed. **As there are not likely to be safety concerns with this system and the number of tasks is probably very small, an ET solution will probably be more appropriate in this situation.**

In between these two extremes there are many systems which involve both periodic tasks and the handling of “random” events. Such designs are typically characterised by the use of multiple interrupt service routines (ISRs). In these situations it may not be practical (or necessary) to create a “pure TT” (single interrupt) solution. However, it may well be practical to create a “more TT” solution which reduces the number of interrupts to a level at which it becomes possible to predict the system behaviour sufficiently accurately to meet the needs of the application.

Related patterns and alternative solutions

See patterns EVENTS TO TIME and TT SCHEDULER

Overall strengths and weaknesses

☺ Use of a TT architecture tends to result in a system with highly predictable patterns of behaviour.

☹ Inappropriate system design using this approach can result in applications which have a comparatively slow response to external events and / or shorter battery life.

EVENTS TO TIME

Abstract pattern

Context

- You and / or your development team have programming or design experience with "event-triggered and / or pre-emptive" (ET/P) system architectures: that is, architectures which may involve use of a conventional real-time operating system (RTOS) and / or multiple interrupt-service routines (linked to different interrupt sources) and / or task pre-emption.
- You are in the process of creating or upgrading an embedded system, based on a single processor.
- You already have at least a design or prototype for your system based on some form of ET/P architecture.
- Because predictable and highly-reliable system operation is a key design requirement, you have opted to employ a "time-triggered system architecture in your system, if this proves practical.

Problem

How can you convert event triggered / pre-emptive designs and code (and mindsets) to allow effective use of a TT SCHEDULER as the basis of your embedded system?

Background

If we were forced to sum up the difference between "embedded" and "desktop" systems in a single word we'd say "interrupts".

Event triggered behaviour in systems is usually achieved through the use of such interrupts. The system is designed to handle interrupts associated with a range of sources (e.g. switch inputs, CAN interface, RS-232, analogue inputs, etc). Each interrupt source will have an associated priority. Each interrupt source will also require the creation of a corresponding "interrupt service routine" (ISR): this can be viewed as a short task which is triggered "immediately" when the corresponding interrupt is generated.

Creating such (ET/P) systems is – on the surface at least – straightforward. However, challenges often begin to arise (in non-trivial designs) at the testing stage. It is generally impossible to determine what state the system will be in when any interrupt occurs, which makes comprehensive testing almost impossible.

A time-triggered system also requires an understanding of interrupts, but the operation is fundamentally different. At the heart of a TT system is a scheduler which determines when the tasks in the system will be executed. In such a system, there is only a single interrupt source (usually a periodic timer "tick"): is used to drive the scheduler.

Solution

Here's what you need to do to migrate to a TT design:

- You need to ensure that only a single – periodic - timer interrupt is enabled (all other interrupt sources will be converted to flags, which will be polled as required).
- You have to determine an appropriate "tick interval" for your system (that is, you need to determine how frequently the timer interrupt need to take place).
- You have to convert any ET (event-triggered) ISRs into periodic tasks and add these to the schedule.
- You need to decide which TT architecture will best suite your application requirements. Pattern TT SCHEDULER provides comprehensive details. To summarise:
- Choose Co-operative architecture if your system requirements could be met without any pre-emption involved. All the tasks in the system will be co-operative. For details see pattern CO-OPERATIVE SCHEDULER
- Choose HYBRID SCHEDULER if limited pre-emption (only a single pre-emptive task) can fulfil the requirements of the system. All the other tasks in the system will be co-operative. Details about implementing such an architecture are documented in pattern
- Choose PRE-EMPTIVE Scheduler for full pre-emption in the system

To illustrate part of the translation process, consider a simple ET system (Listing B-1) running two interrupts, as a result two tasks X and Y will execute. These tasks are invoked by separate interrupts and implemented by associated ISRs.

```
void X_ISR(void) interrupt IEIndex1
   {
   }
void Y_ISR(void) interrupt IEIndex2
   {
   }
void main(void)
   {
   X_init();
   Y_init();
   EA = 1 ; // Enable all interrupts
   while(1)
      {
      PCON |= 0x01;
      }
   }
```

Listing 1: Possible ET design

There are various possibilities to convert ET designs to TT designs with any of the possible TT architectures listed in the solution. One possible design using CO-OPERATIVE SCHEDULER is illustrated in Listing 2.

```
void main(void)
   {
```

```
SCH_Init(); // Set up the scheduler and tasks
X_Init();
Y_Init();

// Add tasks to scheduler
SCH_Add_Task(X_Update(), 0, 100);
SCH_Add_Task(Y_Update(), 20, 200);

// Start the scheduler
SCH_Start();

while(1)
   {
   SCH_Dispatch_Tasks();
   }
}
```

Listing 2: Possible TT design

Please note that X and Y are two separate tasks created in separate .c files with their init and update functions.

Related patterns and alternative solutions

The PTTES collection

The PTTES collection (Pont, 2001) describes, in detail, a range of techniques which can be used to implement embedded systems with TTC architecture. This book can now be downloaded (free of charge) from the following WWW site:
http://www.tte-systems.com/books/pttes

TT Schedulers

The pattern TT SCHEDULER provides relevant background information and the situations in which it may be appropriate to use a TT scheduler in your application.

Reliability and safety implications

When compared to pre-emptive schedulers, co-operative schedulers have a number of desirable features, particularly for use in safety-related systems (Allworth, 1981; Ward, 1991; Nissanke, 1997; Bate, 2000) .

For example, Nissanke (1997, p. 237) notes: “[Pre-emptive] schedules carry greater runtime overheads because of the need for context switching—storage and retrieval of partially computed results. [Co-operative] algorithms do not incur such overheads. Other advantages of [co-operative] algorithms include their better understandability, greater predictability, ease of testing and their inherent capability for guaranteeing exclusive access to any shared resource or data”.

Allworth (1981, pp. 53–54) also notes: “Significant advantages are obtained when using this [co-operative] technique. Since the processes are not interruptable, poor synchronisation does not give rise to the problem of shared data. Shared subroutines can be implemented without producing re-entrant code or implementing lock and unlock mechanisms”.
Although not the main focus of this pattern, the advantages of a TT approach also apply in distributed systems: see, for example, (Scarlett and Brennan, 2006).

Overall strengths and weaknesses

☺ Use of TT architecture tends to result in a system with highly predictable patterns of behaviour.

☹ Inappropriate system design using this approach can result in applications which have a comparatively slow response to external events.

Examples

For examples of systems which are true candidates for TT architecture see pattern TIME FOR TT

References

Allworth, S. T. (1981). *Introduction to Real-Time Software Design,* Macmillan.

Bate, I. J. (2000). Introduction to Scheduling and Timing Analysis, The use of Ada in Real-Time System. IEEE.

Nissanke, N. (1997). Real time Systems, Prentice Hall.

Scarlett, J. J. and R. W. Brennan (2006). Re-evaluating Event-Triggered and Time-Triggered Systems. IEEE conference on Emerging technologies and factory automation: 655-661.

TT SCHEDULER

Abstract pattern

Context

- You already have at least a design or prototype for your system based on some form of ET/P architecture.
- You and / or your development team are using pattern EVENTS TO TIME
- You are in the process of creating or upgrading an embedded system, based on a single processor.

Because predictable and highly-reliable system operation is a key design requirement, you have opted to employ a “time-triggered” system architecture in your system.

Problem

How will you decide which form of time-triggered scheduler should you use for your application?

Background

TT schedulers that we can use can take two forms: Co-operative and Pre-emptive. Both of these types of schedulers provide various options, some of these options are given below:

1. Co-operative Schedulers
    a. Super loop
    b. TTC -ISR
    c. TTC Dispatch

2. Pre-emptive Schedulers
    a. Full Pre-emption
        i. TTRM Scheduler
    b. Limited Pre-emption
        i. TTH Scheduler

In co-operative scheduling, tasks co-operate with each other and wait for their turn to execute until the currently running task finishes execution.
In pre-emptive scheduling a task of higher priority which is ready to execute can pre-empt a currently running task of lower priority.

Overview of TT Schedulers

TTC-SL Scheduler

The simplest way of implementing a TTC scheduler is by means of a “Super Loop” or “endless loop” (e.g. Pont, 2001; Kurian and Pont, 2007). A possible implementation of such a scheduler is illustrated in Listing 1.

```
int main(void)
{
  while(1)
    {
    TaskA();
    Delay_6ms();
    TaskB();
    Delay_6ms();
    TaskC();
    Delay_6ms();
      }
  // Should never reach here
    return 1;
    }
```

Listing 1: TTC SuperLoop Scheduler

Applications based on a TTC-SL SCHEDULER have extremely small resource requirements. Systems based on such a pattern (if used appropriately) can be both reliable and safe, because the overall architecture is extremely simple and easy to understand, and no aspect of the underlying hardware is hidden from the original developer, or from the person who subsequently has to maintain the system.

TTC-ISR Scheduler

Unlike TTC-SL Scheduler, TTC-ISR Scheduler is suitable for use with systems which have hard timing constraints. The basis of a TTC-ISR Scheduler is an interrupt service routine (ISR) linked to the overflow of a hardware timer.

TTC Dispatch Scheduler

The implementation of a TTC-ISR Scheduler is highly system dependent. In addition, the implementation requires a significant amount of hand coding (to control the task timing), and there is no division between the “scheduler” code and the “application” code.

The TTC scheduler implementation referred to here as a “TTC-Dispatch” scheduler provides a more flexible alternative see Listing 2.

The type of TTC scheduler implementation discussed in this pattern is usually implemented using a hardware timer, which is set to generate interrupts on a periodic basis (with “tick intervals” of around 1 ms being typical). In most cases, the tasks will be executed from a “dispatcher” (function), invoked after every scheduler tick. The dispatcher examines each task in its list and executes (in priority order) any tasks which are due to run in this tick interval (see Figure 1). The scheduler then places the processor into an “idle” (power saving) mode, where it will remain until the next tick.

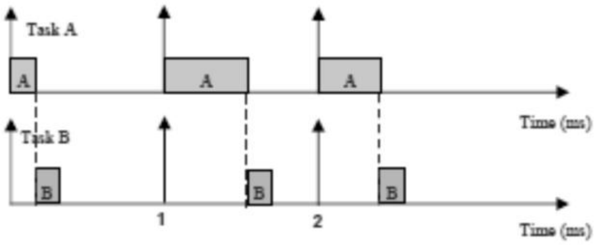


Figure 1: Tasks scheduled in a TTC design

Provided that an appropriate implementation is used, a time-triggered, co-operative (TTC) architecture is a good match for a wide range of low-cost, resource-constrained applications.

TTC architectures also demonstrate very low levels of task jitter (Locke, 1992) and can maintain their low-jitter characteristics even when techniques such as dynamic voltage scaling (DVS) are employed to reduce system power consumption.

```
void main(void)
   {
   // Set up the scheduler
   SCH_Init_T2();

   // Init tasks
   TaskA_Init();
   TaskB_Init();

   // Add tasks (10 ms ticks)
  //  Parameters  are  filename,  offset  (ticks),  period
(ticks)
   SCH_Add_Task(TaskA, 0, 3);
   SCH_Add_Task(TaskB, 1, 3);
   SCH_Add_Task(TaskC, 2, 3);

   // Start the scheduler
   SCH_Start();

   while(1)
      {
      SCH_Dispatch_Tasks();
      SCH_Go_To_Sleep();
      }
   }
```

Listing 2: TTC Implementation

TTRM architectures

Where a TTC architecture is not found to be suitable for use in a particular resource constrained embedded systems, fixed-priority scheduling has been proposed as the most attractive alternative (Audsley, Burns *et al.*, 1991; Bate, 1998).

“Time-triggered rate monotonic” (TTRM) is a well-known fixed-priority scheduling algorithm that was introduced by (Liu and Layland, 1973) in 1973. Technically, TTRM is a pre-emptive scheduling algorithm which is based on a fixed priority assignment (Kopetz, 1997) . In particular, the priorities are assigned to periodic tasks accord to their occurrence rate or, in other words, priorities are inversely proportional to their period, and they do not change through out of the operation (because their periods are constant).

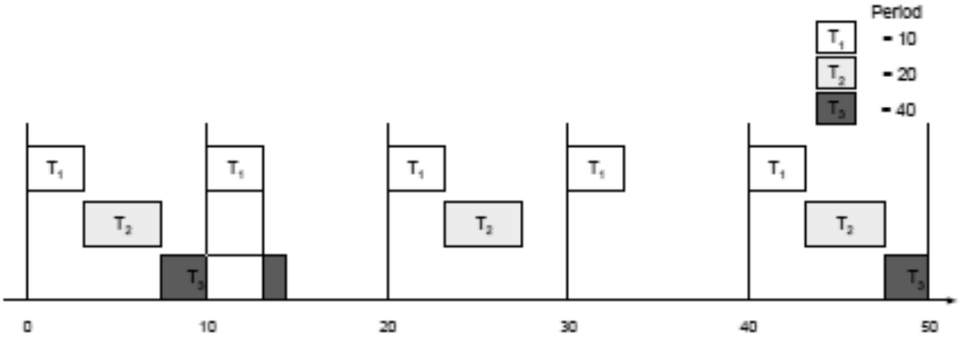


Figure 2: Illustrating TTRM architecture

To illustrate the use of TTRM scheduling, Figure 5 above shows how a set of periodic tasks can be scheduled by this algorithm. Task T1 is executed periodically at the fastest rate, every 10 ms, and is determined to be the highest priority in this scheduling policy, while task T2 and T3, which are run every 20 and 40 ms respectively, have lower priority levels according to their rates. A task scheduled by the TTRM algorithm can be pre-empted by a higher priority task. As illustrated in Figure 2, task T3 - which is running - is pre-empted by task T1 is at time 10: it carries on after the completion of task T1.

TTH architectures

Where a TTC architecture is not found to be suitable for a particular system, use of a TTRM design may not be necessary. For example, a single, time-triggered, pre-empting task can be added to a TTC architecture, to give what we have called a “time-triggered hybrid” (TTH) scheduler (Pont, 2001; Maaita and Pont, 2005) and others have called a “multi-rate executive with interrupts” (Kalinsky, 2001).

Use of a TTH scheduler allows the system designer to create a static schedule made up of (i) a collection of tasks which operate co-operatively and (ii) a single – short - pre-empting task (see Figure 3) . In many of the systems employing a TTH architecture, the pre-empting task will be used for periodic data acquisition, typically through an analogue-to-digital converter or similar device. Such requirements are common in, for example, control systems (Buttazzo, 2005) and applications which involve data sampling and Fast-Fourier transforms (FFTs) or similar techniques:

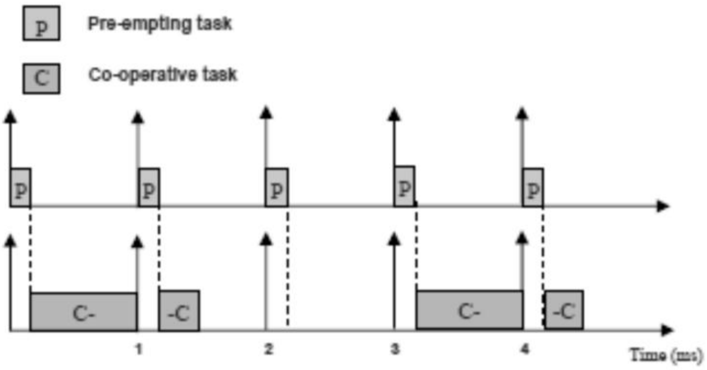


Figure 3: Illustrating TTH design

Solution

Here are the guidelines about choosing appropriate TT architecture.

When to use TTC

Use TTC architecture where ever possible as first choice because of its simple and efficient design. Of course, this architecture is not always appropriate. The main problem is that long tasks will have an impact on the responsiveness of the system. This concern is succinctly summarised by Allworth: "[The] main drawback with this [co-operative] approach is that while the current process is running, the system is not responsive to changes in the environment. Therefore, system processes must be extremely brief if the real-time response [of the] system is not to be impaired." (Allworth, 1981).

We can express this concern slightly more formally by noting that if the system must execute one of more tasks of duration X and also respond within an interval T to external events (where T < X), a pure co-operative scheduler will not generally be suitable. In more simple words duration of a task (execution time) must be less than the tick interval of the system.

In practice, it is sometimes assumed that a TTC architecture is inappropriate because some simple design options have been overlooked, see pattern BUFFERED OUTPUT

When to use TTH

For systems where TTC is not an appropriate choice, avoid jumping to fully pre-emptive architectures as they incur higher overheads because of context switching involved during task pre-emption. Check for TTH solution which provides a limited level of pre-emption. For example, consider a wireless electrocardiogram (ECG) system. An ECG is an electrical recording of the heart that is used for investigating heart disease. In a hospital environment, ECGs normally have 12 leads (standard leads, augmented limb leads and precordial leads) and can plot 250 sample-points per second (at minimum). In the portable ECG system considered here, three standard leads (Lead I, Lead II, and Lead III) were recorded at 500 Hz. The electrical signal were sampled using a (12-bit) ADC and – after compression – the data were passed to a "Bluetooth" module for transmission to a notebook PC, for analysis by a clinician see (Phatrapornnant and Pont, 2006).

In one version of this system, we are required to perform the following tasks:

- Sample the data continuously at a rate of 500 Hz. Sampling takes less than 0.1 ms.
- When we have 10 samples (that is, every 20 ms), compress and transmit the data, a process which takes a total of 6.7 ms.

In this case, we will assume that the compression task cannot be neatly decomposed into a sequence of shorter tasks, and we therefore cannot employ a pure TTC architecture. However, even if you cannot – cleanly - solve the long task / short response time problem, then you can maintain the core co-operative scheduler, and add only the limited degree of pre-emption that is required to meet the needs of your application.

For example, in the case of our ECG system, we can use time-triggered hybrid architecture. In this case, we allow a single pre-empting task to operate: in our ECG system, this task will be used for data acquisition. This is a time-triggered task, and such tasks will generally be implemented as a function call from the timer ISR which is used to drive the core TTC scheduler. As we have discussed in detail elsewhere (Pont, 2001: Chapter 17) this architecture is extremely easy to implement, and can operate with very high reliability. As such it is one of a number of architectures, based on a TTC scheduler, which are cooperatively based, but also provide a controlled degree of pre-emption.

When to use TTRM

If both TTC and TTH architectures are not appropriate for your application and full pre-emption is a necessary, then TTRM architecture may match your requirements.

Overall, it has been claimed that the main advantage of TTRM scheduling is flexibility during design or maintenance phases, and that such flexibility can reduce the total life cost of the system (Locke, 1992; Bate, 1998). The schedulability of the system can be determined based on the total CPU utilisation of the task set: as a result - when new functionalities are added to the system – it is only necessary to recalculate the new utilisation values. In addition, unlike a TTC design, there is no need to break up long individual tasks in order to meet the length limitations of the minor cycle. The need to employ harmonic frequency relationships among periodic tasks is also avoided. Finally, the scheduling behaviour can be predicted and analysed using a task model proposed by Liu and Layland (1973).

However, the scheduling overheads of TTRM schedulers tend to be larger than those of TTC schedulers because of the additional complexity associated with the context switches when saving and restoring task state (Locke, 1992). This is a concern in embedded systems with limited resources.

Locking mechanisms

If you use any architecture which involves pre-emption (TTH or TTRM), you need to consider ways of preventing more than one task from accessing critical resources at the same time. See Patterns CRITICAL SECTION for more details.

Overall strengths and weaknesses

☺ Use of a TT scheduler tends to result in a system with highly predictable patterns of behaviour.

☹ Inappropriate system design using this approach can result in applications which have a comparatively slow response to external events.

References

Albert, A. and R. Bosch GmbH (2004). Comparison of Event-Triggered and Time-Triggered Concepts with regard to Distributed Control Systems. Embedded World. Nurnberg**:** 235-252.

Allworth, S. T. (1981). Introduction to Real-Time Software Design, Macmillan.

Audsley, N. C., A. Burns, et al. (1991). Hard Real-time Scheduling: The Deadline Monotonic Approach. Eight IEEE Workshop on Real-time operating Systems and Softwares., Atlanta, USA.

Ayavoo, D. (2006). The Development of reliable X-by-Wire Systems: Assessing The Effectiveness of a Simulation First Approach. Department of Engineering, University of Leicester. **PhD Thesis**.

Ayavoo, D., M. J. Pont, et al. (2005). A Hardware-in-the-Loop' testbed representing the operation of a cruise-control system in a passenger car. Proceedings of the Second UK Embedded Forum. Birmingham, UK, University of Newcastle upon Tyne.

Baker, T. P. and A. Shaw (1988). "The Cyclic Executive Model and Ada. "Real-Time Systems Springer Netherlands **1**(1): 7-25.

Bate, I. J. (1998). Scheduling and timing analysis for safety critical real-time systems. Department of Computer Science, University of York. **PhD Thesis**.

Bate, I. J. (2000). Introduction to Scheduling and Timing Analysis, The use of Ada in Real-Time System. IEEE.

Burns, A. and A. J. Wellings (1994). "HRT-HOOD: a structured design method for hard real-time systems." Real-Time Systems **6**(6): 73-114.

Burns, A. and B. Wellings (1997). Real-Time Systems and Programming Languages, Addison Wesley.

Buttazzo C, G. (2003). Rate Monotonic vs EDF : Judgement Day. LNCS, Springer. **2855/2003:** 67-83.

Buttazzo, C. G. (1997). Hard Real-Time Computing Systems Predictable Scheduling Algorithms and Applications, Kluwer Academic Publishers.

BUFFERED OUTPUT

Design pattern

Context

- You are applying the pattern EVENTS TO TIME (TTC)
- You need to deal – cleanly – with a "long task" (that is, a task which may have an execution time greater than your chosen tick interval.
- You need to send a significant amount of data between your processor / system and an external device: the data transfer process will take some time.

Problem

How can you structure the data-transfer tasks in your application in a manner which is compatible with TTC architecture?

Background

We illustrate the need for the present pattern with an example.

Suppose we wish to transfer data to a PC at a standard 9600 baud. Transmitting each byte of data, plus stop and start bits, involves the transmission of 10 bits of information (assuming a single stop bit is used). As a result, each byte takes approximately 1 ms to transmit.

Now, suppose we wish to send this information to the PC:

```
Current core temperature is 36.678 degrees
```

If we use a standard function (such as some form of printf()) the task sending these 42 characters will take more than 40 milliseconds to complete. In a system supporting task pre-emption, we may be able to treat this as a low-priority task and let it run as required. This approach is not without difficulties (for example, if a high-priority task requires access to the same communication interface while the low-priority task is running). However, with appropriate system design we will be able to make this operate correctly under most circumstances.

Now consider the equivalent TTC design. We can't support task pre-emption and a long data-transmission task (around 40 ms) is likely to cause significant problems. More specifically, if this time is greater than the system tick interval (often 1 ms, rarely greater than 10 ms) then this is likely to present a problem as shown in Figure 1. The RS-232 task is a "long task" has duration greater than the system tick and so is missing the next tick intervals.

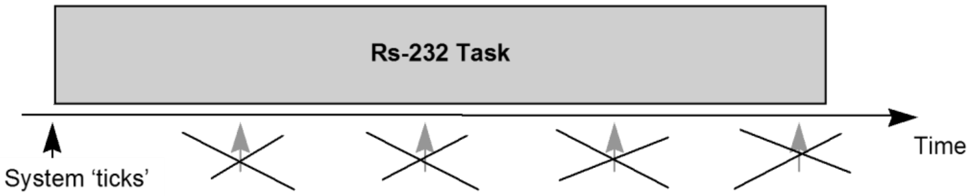


Figure 1: A schematic representation of the problems caused by sending a long character string on an embedded system. In this case, sending the massage takes 42 ms while the System tick interval is 10 ms.

Perhaps the most obvious way of addressing this issue is to increase the baud rate; however, this is not always possible, and - even with very high baud rates - long messages or irregular bursts of data can still cause difficulties.

More generally, the underlying problem here is that the data transfer operation has a duration which depends on the length of the string which we wish to submit. As such, the worst-case execution time (WCET) of the data transfer task is highly variable (and, in a general case, may vary depending on conditions at run time). In a TTC design, we need to know all WCET data for all tasks at design time. We require a different system design. As (Gergeleit and Nett, 2002) have noted " Nearly all known real-time scheduling approaches rely on the knowledge of WCETs for all tasks of the system." The known WCET of tasks will be helpful for developers in designing the offline schedule and preventing task overrun.

Solution

Convert a long data-transfer task (which is called infrequently and may have a variable duration) into a periodic task (which is called comparatively frequently and which has a very short – and known – duration).

A BUFFERED OUTPUT consists of three key components:

- A buffer (usually just an array, implemented in software)
- A function (or small set of functions) which can be used by the tasks in your system to write data to the array.
- A periodic (scheduled) task which checks the buffer and sends a block of data to the receiving device (when there are data to send).

Figure 2 provides an overview of this system architecture. All data to be sent are first moved to a software buffer (a very fast operation). The data is then shifted – one block at a time – to the relevant hardware buffer in the microcontroller (e.g. 1 byte at a time for a UART, 8 bytes at a time for CAN, etc): this software-to-hardware transfer is carried out every 1ms (for example), using a (short) periodic task.

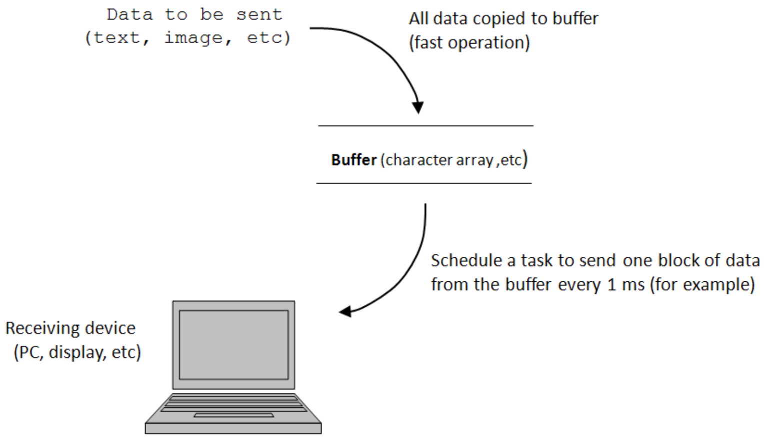

Data to be sent
(text, image, etc)
All data copied to buffer
(fast operation)
Buffer (character array ,etc)
Schedule a task to send one block of data
from the buffer every 1 ms (for example)
Receiving device
(PC, display, etc)

Figure 2: An overview of the BUFFERED OUTPUT architecture.

Hardware resource implications

In most cases, the CPU requirements for BUFFERED OUTPUT are very limited, provided we take reasonable care at the design stage. For example, if we are sending message over a CAN bus and we know that each message takes approximately 0.15 ms to transmit; we should schedule the data transmission task to check the buffer at an interval > 0.15 ms. If we do this, the process of copying data from the software buffer to the (CAN) hardware will take very little time (usually a small fraction of a millisecond).

For very small designs (e.g. 8-bit systems) the memory requirements for the software buffer can prove significant. If you can't add external memory in these circumstances, you will need to use a small buffer and send data as frequently as possible (but see the comment above).

In some cases, hardware support can help to reduce both memory requirements and processor load. For example, if using UART-based data transmission, UARTs often have 16-byte hardware buffers: if you have these available, it makes sense to employ them.

Portability

This technique is generic and highly portable.

Reliability and Safety Issues

- Special care must be taken while defining buffer length, the data transfer should not cause any buffer overflow.
- Applications that involve high amount of data transfer like video and DSP applications or data acquisition systems the use of buffer might not be a viable solution.

Overall strengths and weaknesses

☺ Use of buffered output is an easy solution for faster data transfer from a task running in an embedded application

☹ One has to be very careful while defining the buffer length, inappropriate buffer definitions may cause buffer overflow and data loss.

POLLED INPUT

Design pattern

Context

- You are applying the pattern EVENTS TO TIME
- You need to make your system responsive to external inputs through interface like switches.

Problem

How do I build a time-triggered (TT) system which is equivalent of my event-triggered (ET) system such that it can respond to all (external/internal) input interfaces?

Background

Designing a TT system requires more planning efforts. In a time-triggered co-operative (TTC) design the possible occurrence and the execution times of all the tasks needs to be known in advance. The designer has to plan a task schedule which must execute all the tasks periodically at their allocated time intervals. This effort makes the system more predictable. In contrast to this, in an event triggered system the schedule executes the tasks dynamically as the events arrive thus no guarantee that they meet any timeliness constraints. This is the reason that ET designs are not recommended for safety critical applications. The event triggered behaviour in systems is achieved through the use of interrupts. To support these interrupts, Interrupt Service Routines (ISRs) are provided. Whenever an interrupt occurs it stops the currently running task and ISR executes to respond to the interrupt. This “context switching” is an overhead that sometimes raised serious complications in systems.

The abstract pattern EVENTS TO TIME provides more relevant background information.

Solution

A POLLED INPUT should meet the following specification:

- It should include a periodic task which polls for the occurrence of the event.
- The period of the above task should be set to some value less than or equal to minimum inter-arrival time[24] of the event in question.

The interrupt associated with this event should not be enabled. In fact only one interrupt associated with the timer responsible for generating system “ticks” should be enabled.

[24] In ET systems the exact arrival time of events is not known so we assume a minimum distance between the arrivals of two consecutive events.

Hardware resource implications

Different interfaces have different implications under various circumstances. Reading a switch input imposes minimal loads on CPU and memory resources whereas scanning the keypad interface imposes both a CPU and memory load.

Reliability and safety issues

One major concern here in migrating from event triggered to time triggered is to make systems more predictable. Characteristic for the time triggered architecture is the treatment of (physical) real time as a first order quantity (Kopetz and Bauer 2002) this implies to the fact that time triggered systems must be very carefully designed, the task activation rates must be fixed according to the system dynamics i.e. how frequent an input needs to be polled.

Portability

This technique is generic and highly portable.

Overall strengths and weaknesses

☺ A flexible technique, programmer can easily do changes in code for example if auto repeat is required

☺ It is simple and cheap to implement.

☹ Provides no protection against out of range inputs or electrostatic discharge (ESD)

☹ More processor utilisation in polling for tasks

CHOOSING TASK PARAMETERS

Design pattern

Context

- You are in the process of creating or upgrading an embedded system, based on a single processor.
- Because predictable and highly-reliable system operation is a key design requirement, you have opted to employ a "time-triggered" system architecture in your system, if this proves practical.

Problem

How can you choose your tasks parameters such as offset and task order to allow effective use of a TT Scheduler as the basis of your embedded system?

Background

Whether a TTC or TTH implementation is used, a number of key scheduler/task parameters must be determined (including the tick interval, task order, and initial delay or phase of each task). Inappropriate choices may mean that a given task set cannot be scheduled at all or inappropriate decisions may still lead to unnecessarily high levels of task jitter. The following parameters are used to characterise each task (Liu and Layland, 1973; Tindell, Burns *et al.*, 1994; Buttazzo, 1997).

1. **Period (P_i):** is the time interval after which task T_i should be repeated, in other words it is the length of time between every two invocations.
2. **Offset (O_i):** is the time, measured from the start of the system power on, after which the first period of task T_i starts.
3. **Release time (r_i):** is the time, measured from the start of the task period, after which task T_i becomes ready to run.
4. **Deadline (D_i):** is the time before which task T_i should be completed. Deadline can be measured from the start of the system power on, in which case it is called absolute deadline. Alternatively it can be measured from the start of the task period, in which case it is called relative deadline

These parameters are shown in Figure 1.

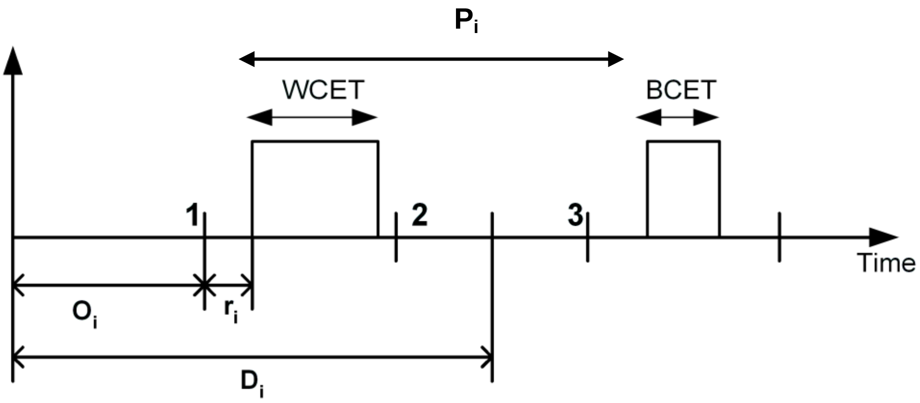


Figure 1: Illustrating task parameters

Another important parameter is the order of tasks in which they are added to the schedule. Inappropriate choice of these parameters may lead to high values of jitter, increased power consumption or a task set cannot be scheduled at all.

Solution

Choosing the correct offset

A task offset specifies when a task should start or more precisely it specifies the first tick at which the first instance of a task is ready to run. For a task set given with WCET, period and deadline start scheduling all the tasks with offset 0 if the sum of execution times of all the tasks is less than or equal to the length of tick interval.
While assigning task offsets you can take care of the following situations.

For example, the tasks in Table 1 can be scheduled if a tick interval of 3 ms is used and the tasks will meet their deadlines as well.

Table 1: Task specifications for a system in which task offsets are appropriate (all the tasks will meet their deadlines)

Task	WCET(ms)	Deadline (ms)	Period (ms)	Offset (ticks)
A	0.5	3	3	0
B	0.75	3	6	0
C	1.5	3	6	0

On the contrary, task set in Table 2 can be scheduled with a tick interval of 5ms but Task C will missed its deadline as shown in Figure2.

Table 2: Task specifications for a system in which task offsets are in appropriate (Task C will not be able to meet its deadline)

Task	WCET(ms)	Deadline (ms)	Period (ms)	Offset (ticks)
A	1	5	5	0
B	1.5	5	10	0
C	3	5	10	0

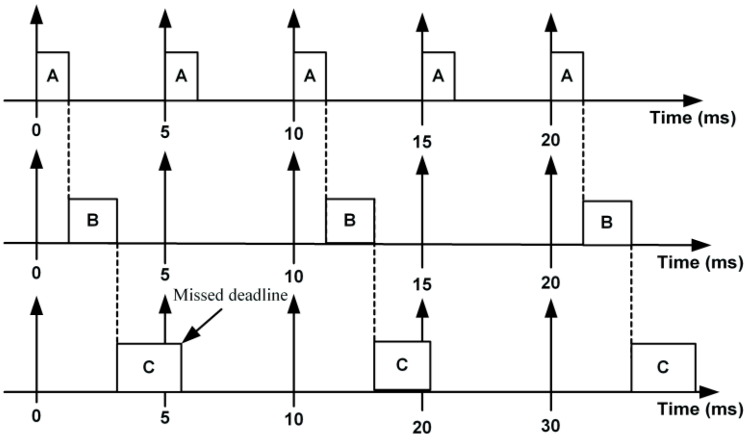


Figure 2: Task C missed deadline because of incorrect offset

By changing the offset to 1, Task C will meet its deadline as shown in Figure 3.

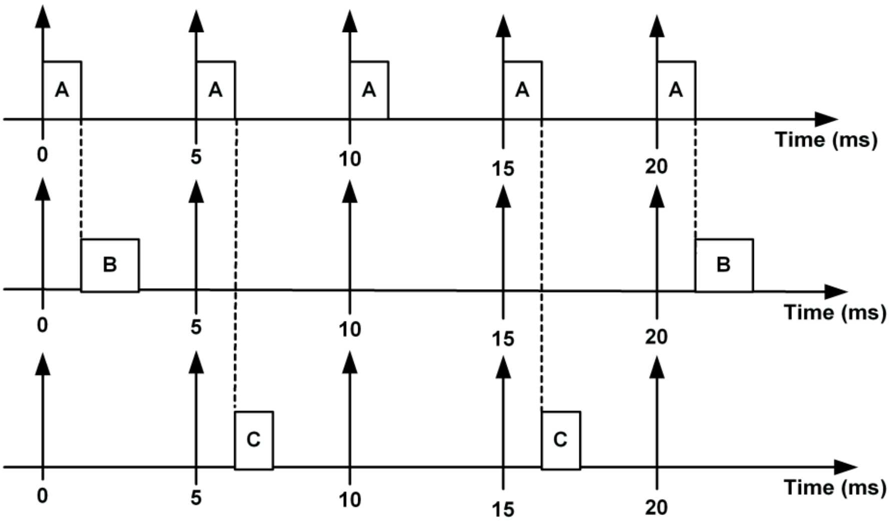


Figure 3: Task C will meet deadline after changing offset to 1

Inappropriate assignment of task offsets can also increase the jitter value in task. For example consider the task set given in Table 3. With the given set of parameters Task C will run after Task A and Task B in some ticks and just after Task A in some other ticks (see Figure 4). This kind of situation poses a kind of unpredictability in system behaviour. This can be adjusted by keeping the jitter constant in Task C in all ticks. Changing the offset of Task C can help to keep the jitter value constant in all the ticks see Table 4 and Figure 5.

Table 3: Task offset that can cause varied jitter in Task C

Task	WCET(ms)	Deadline (ms)	Period (ms)	Offset (ticks)
A	1	5	5	0
B	1.5	20	20	0
C	1	10	10	0

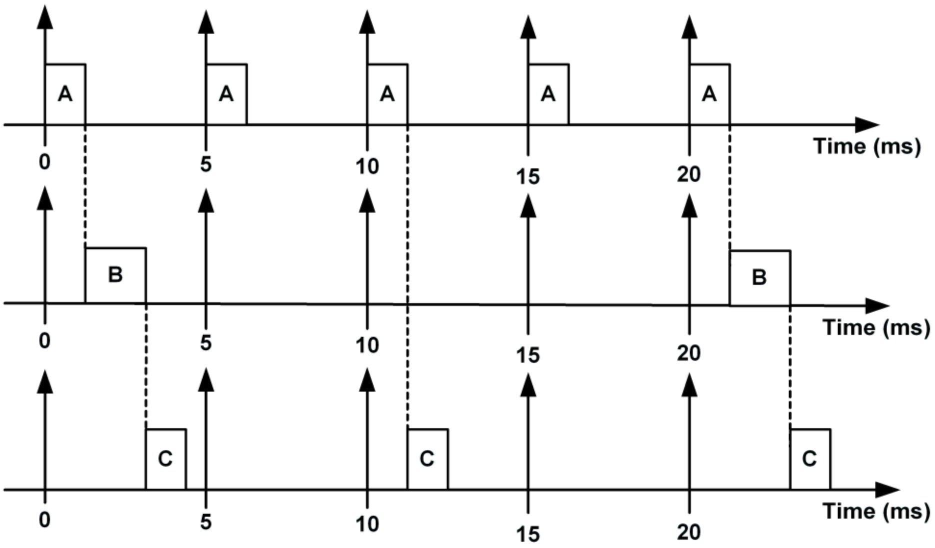


Figure 4: Illustrating Tasks shown in Table 3 above:

Table 4: Changing the offset of Task C to 1 can make the jitter constant for all task instances

Task	WCET(ms)	Deadline (ms)	Period (ms)	Offset (ticks)
A	1	5	5	0
B	1.5	20	20	0
C	1	10	10	1

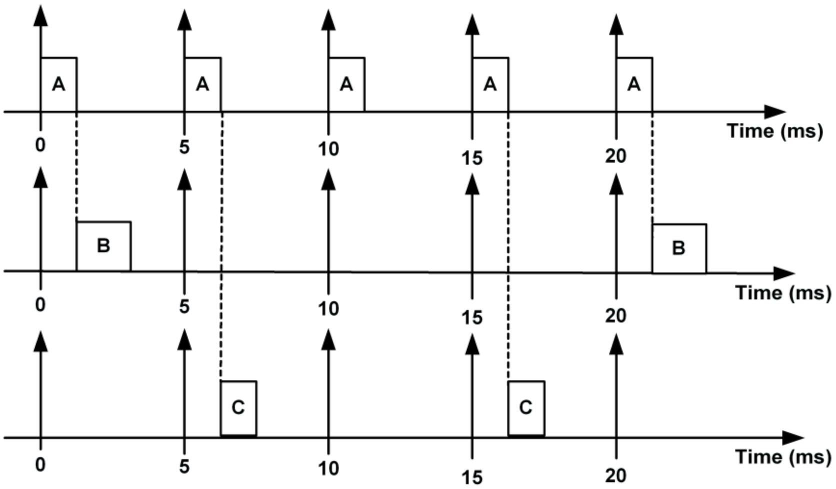


Figure 5: Changing of task offset can keep the jitter constant

Choosing correct task order

Task order can also affect jitter. It is important to consider the task order for jitter sensitive tasks. For example consider the schedule given in Table 5 and Figure 6.

Table 5: Example of incorrect order of task set which can cause varied jitter in task C

Task	WCET(ms)	Deadline (ms)	Period (ms)	Offset (ticks)
A	2	10	10	0
B	3	20	20	0
C	4	30	30	0

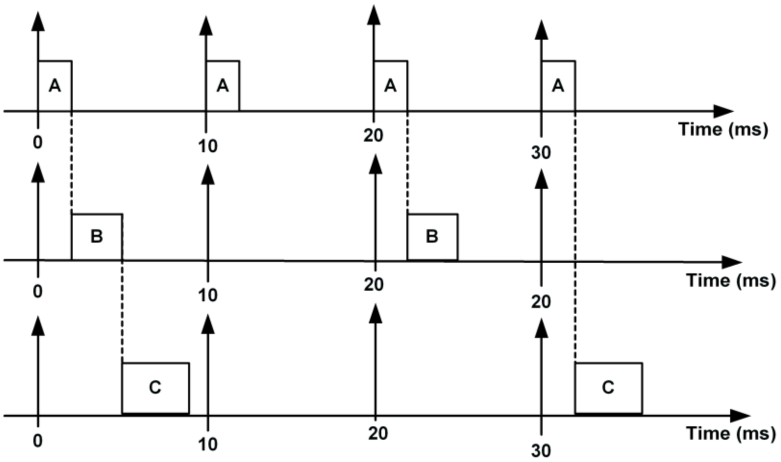


Figure 6: Task C showing variations in jitter because of incorrect task order

Changing the task order can keep the jitter constant in Task C. The new order of tasks is shown in Table 6 and Figure 7.

Table 6: Rearrangement of the task order to keep the jitter constant in task C

Task	WCET(ms)	Deadline (ms)	Period (ms)	Offset (ticks)
A	2	10	10	0
C	4	30	30	0
B	3	20	20	0

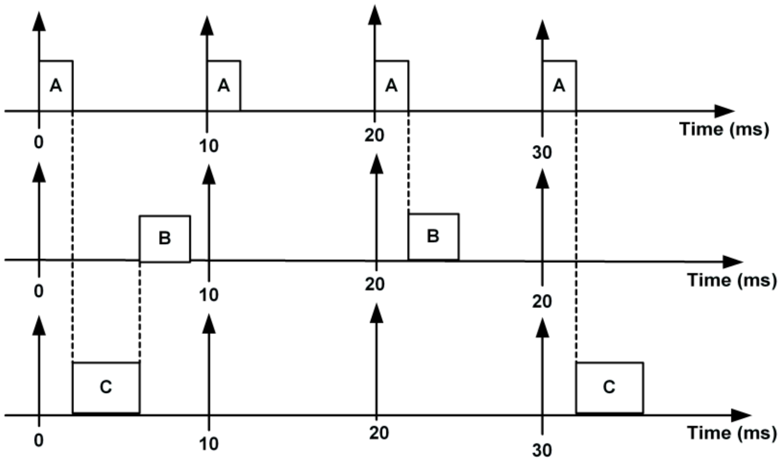


Figure 7: Rearrangement of task order will make task C run after task A every time

Reliability and safety implications

Inappropriate selection of task parameters can make the system unreliable

Overall strengths and weaknesses

☺ Uses of appropriate task parameters will result in stable system with all the tasks meet their deadlines and reduced/constant values of jitter.

☹ Inappropriate system design using this approach can result in applications which have a comparatively slow response to external events.

References

Liu, C. L. and J. W. Layland (1973). "Scheduling algorithms for multiprogramming in a hard real-time environment." Journal of the ACM **20**(1): 46-61.

Tindell, K. W., A. Burns, et al. (1994). "An extendible approach for analyzing fixed priority hard real-time tasks." Real-Time Systems 6: 133-151.

Buttazzo, C. G. (1997). Hard Real-Time Computing Systems Predictable Scheduling Algorithms and Applications, Kluwer Academic Publishers.

BALANCED SYSTEM

Abstract pattern

Context

- You are developing an embedded system.
- You have decided to move to or are already working with TT architectures.
- Predictable timing behaviour is the key requirement.

Problem

How can you ensure that your TT system has minimum possible jitter?

Background

To get a guaranteed predictable system, choice of appropriate architecture on top of the application is extremely important. In applications which are based on an ET architecture, tasks run sporadically in response to interrupts whereas in a TT system, tasks are periodic. However, there is a single interrupt which generates “ticks” to control the task periods.

To achieve certification standards it is advisable to avoid the use of arbitrary interrupts in running the tasks because of the increased difficulty in attaining sufficient test coverage. One particular reason is, arbitrary interruptions lead to a vast increase in the potential paths within software when compared to code with no interruptions (Bate, 1998).

From the predictability point of view, this would make TT architecture an appropriate choice for a number of applications.

Only choosing TT architecture does not fully guarantee the system predictability as there are a number of other factors which could make a TT system unpredictable. The parameters of tasks which are running under a TT architecture such as release time, execution time, finish time and deadline are required to be known in advance. The prior knowledge of these parameters plays an important role in guaranteeing the overall predictability of the system. However, systems those run in practice generally show considerable variations in these parameters. These variations are termed as jitter.

Jitter in tasks

To understand the concept of jitter more clearly, consider the different instances of a task (Task A) as shown in Figure 4. For tasks in TT systems, release time can be considered as the point at which we would ideally expect a task to start its execution. In actual practice this is delayed due to factors such as scheduler overhead and variable interrupt response times (Liu, 2000; Maaita and Pont, 2005). The actual start time of a task is always deviated from its (pre-determined) release time and we can say that tasks always suffer from release jitter - see unequal values of x1, x2 and x3 in Figure 1.

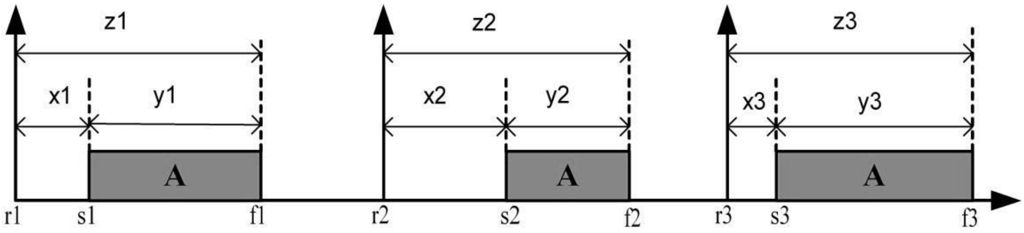


Figure 1: Illustration of jitter in different calls of a periodic task

In real-time systems one important parameter is the upper bound of the execution time for a task, known as worst case execution time (WCET). Unfortunately, determining WCET of tasks is rarely straightforward (Puschner, 2002; Puschner, 2003). This is because the program code of a task may contain conditional branches and / or loops and each may take different times to execute (Liu, 2000). The decision between one branch and the other during task execution is dependent on the input data. This makes predicting a branch prior to execution a very difficult task. All these factors lead to variable execution time of a task and this is known as execution jitter (see imbalanced values of y1, y2 and y3 in Figure 1). The cascading effects of release and execution jitter will result in the deviation of task finish time, shown as z1, z2 and z3 in Figure 1.

Ideally, a predictable system should be jitter free. Considering Figure 1 once again, we can say that in a zero jitter system:

$$x1 = x2 = x3$$
$$y1 = y2 = y3$$
$$z1 = z2 = z3$$

Some hardware features such as variations in the frequencies of oscillator and use of cache memories (Kirner and Puschner, 2003) also contribute to jitter in tasks.

For some applications, such as data, speech or music playback (for example) these variations may make no measurable difference to the system. However, for applications in real-time control systems which involve sampling, computation and actuation, such delays in operations are very risky for the overall performance of the system. The presence of jitter can have a degrading impact on the performance of real-time systems or can even lead to critical failure (Martin, 2005).

Solution

A BALANCED SYSTEM is one which has minimum values for all types of jitters. One way we can address the challenges discussed in the previous section is to tackle them directly. For example, rather than hoping that we can predict the WCET (for example, through static code analysis or measurement) we can set out from start to ensure that our code is “balanced” and that the WCET and BCET (best case execution time) are always fixed (and equal). Once we have balanced the code, it becomes comparatively easy to determine (during system testing and during system execution) whether the system tasks actually have a fixed execution time.

Related patterns

In a task, balancing can be required at different levels. For example, we may need to balance the whole task or just sections (for example, areas with loops and conditional code). In some cases, the goal may be to fix the timing of an activity in the task relative to the start of the task (for example, we may wish to ensure that exactly 0.2 ms after the start of a task a sample is taken from a data source).

- SANDWICH DELAY provides a simple solution to balance a task through exclusive use of a hardware timer.
- SINGLE PATH DELAY is a programming approach to ensure that blocks of code involving loops or decision structures will have a single execution path.
- TAKE A NAP is an alternative to achieving balanced code for power constrained systems.
- PLANNED PRE-EMPTION provides a way of achieving balancing for pre-emptive systems.

Reliability and safety implications

Extra care is needed while selecting the tasks/sections of code to be balanced. This is because balancing makes use of additional hardware and software. Devoting resources unnecessarily to balance tasks which are not critical could lead to a fatal rather than predictable system.

Overall strengths and weaknesses

☺ A BALANCED SYSTEM is more robust against the presence of various types of jitters in the system.

☺ Results in a more predictable timing behaviour of the system.

☹ Requires (non-exclusive) access to some hardware resources, for example, timers.

☹ Balancing a system requires extra effort in writing code for balancing which in turn increases CPU utilisation.

References

Kirner, R. and P. Puschner (2003). Discussion of Misconceptions about WCET Analysis. 3rd Euromicro International workshop on WCET Analysis.

Liu, J. W. S. (2000). Real-Time Systems, Prentice Hall.

Maaita, A. and M. J. Pont (2005). Using 'planned pre-emption' to reduce levels of task Jitter in a time-triggered hybrid scheduler. Second UK Embedded Forum. Birmingham, UK: 18-35.

Martin, T. (2005). The insider's guide to the Philips ARM7 Based microcontrollers, Coventry, Hitex, UK, Ltd.

Puschner, P. (2002b). Is WCET Analysis a non-problem? Towards new Software and Hardware architectures. 2nd International Workshop on Worst Case Execution Time Analysis, Vienna, Austria.

Puschner, P. (2003). The single-path approach towards WCET-analysable software. IEEE International Conference on Industrial Technology.

SANDWICH DELAY

Design pattern

Context

- You are using the pattern BALANCED SYSTEM.
- In your application you are running two activities, one after the other.

Problem

How can you ensure that the execution time of the tasks is always predictable so that the release time of the two activities is known and fixed?

Background

Suppose we have a system executing two functions periodically using a timer ISR, as outlined in Listing 1.

```
//Interrupt service Routine (ISR) invoked by timer overflow
every 10ms
void Timer_ISR (void)
{
      Do_X();   //WCET approx 4.0ms
      Do_Y();   //WCET approx 4.0ms
}
```

Listing 1: System executing two functions using timer ISR

According to Listing 1, function Do_X() will be executed every 10ms. Similarly, function Do_Y() will be executed every 10 ms, after Do_X() completes. For many resource-constrained applications (for example, control systems) this architecture may be appropriate. However, in some cases, the risk of jitter in the start times of function Do_Y() may cause problems. Such jitter will arise if there is any variation in the duration of function Do_X(). In Figure 1, the jitter is reflected in differences between the values of *ty1* and *ty2* (for example).

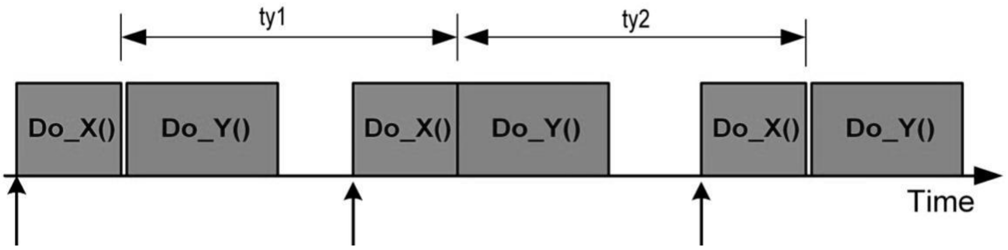


Figure 1: The impact of variations in the duration of Do_X() on the release jitter of Do_Y()

Solution

A SANDWICH DELAY can be used to solve this type of problem. More specifically, a SANDWICH DELAY provides a simple but highly effective means of ensuring that a particular piece of code always takes the same period of time to execute: this is done using two timer operations to “sandwich” the activity you need to perform. Please refer to code segment in Listing 1.

```
//ISR invoked by timer overflow every 10ms
void Timer_ISR (void)
{
   /*Execute Do_X() in a ‘Sandwich Delay’  - BEGIN
         Set timer to overflow after 5 ms*/
      Set_Sandwich_Timer_overflow(5);
      Do_X();    //WCET approx. 4.0ms

     //Wait for timer to overflow
      Wait_Sandwich_Timer_Overflow();

      //Execute Do_X() in a ‘Sandwich Delay’  - END
         Do_Y();    //WCET approx 4.0ms
   }
```

Listing 1: Pseudo code for SANDWICH DELAY

The timer is set to overflow after 5 ms (a period slightly longer than the WCET of Do_X()). We then start this timer before we run the function and - after the function is complete - we wait for the timer to reach 5 ms value. In this way, we ensure that as long as Do_X() does not exceed a duration of 5 ms – Do_Y() runs with minimum jitter as shown in Figure 2.

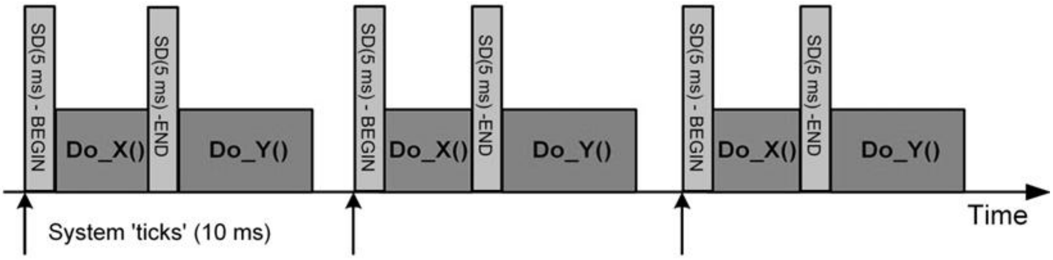


Figure 2: Reducing the impact of variations in the durations of Do_X() on the release jitter of Do_Y() through the use of SANDWICH DELAY

Sandwich delays are also found to be useful for systems involving pre-emption, for example, TTH designs. In such designs controlling the execution jitter of a pre-emptive task using a delay (slightly bigger than the WCET of the pre-emptive task) showed considerable reduction in the period jitter of the co-operative task - see Figure 3.

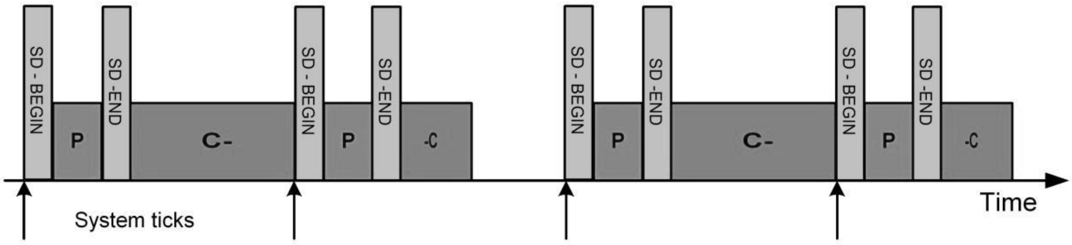


Figure 3: Reducing the impact of variations in the durations of pre-emptive task on the release jitter of co-operative tasks through the use of SANDWICH DELAY in TTH designs

Reliability and safety implications

Use of Sandwich Delay is generally straight forward, but there are three potential issues of which you should be aware.

You need to know the duration WCET of the functions to be sandwiched. If you underestimate this value, the timer will already have reached its overflow value when your function(s) complete, and the level of jitter will not be reduced (indeed the Sandwich Delay is likely to slightly increase the jitter in this case)
You must check the code carefully, because the "wait" function may never terminate if the timer is incorrectly set up. In these circumstances a monitoring technique may help to rescue the system. See patterns System Monitors, Watchdog, Loop Timeout and Task Guardian.

You will rarely manage to remove all jitter using such an approach, because the system cannot react instantly when the timer reaches its maximum value (at the machine-code level, the code used to poll the timer flag is more complex than it may appear, and the time taken to react to the flag change will vary slightly). A useful rule of thumb is that jitter levels of around 1 microsecond will still be seen using a Sandwich Delay.

Overall strengths and weaknesses

☺ A simple way of ensuring that the WCET of a block of code is highly predictable.

☹ Requires (non-exclusive) access to a timer.

☹ Will only rarely provide a "jitter free" solution: variations in code duration of around 1 microsecond are representative.

SINGLE PATH

Design pattern

Context

- You are using the pattern BALANCED SYSTEM.
- You have decided to balance sections of code involving loops and decision structures implemented within the application tasks.

Problem

How would you ensure that the execution time of your application code sections involving loop and decision structures will remain fixed every time they run?

Background

Variable execution times of tasks can lead to unpredictable behaviour in systems. To understand this more clearly, consider a system running tasks A, B and C as shown in Figure 1.

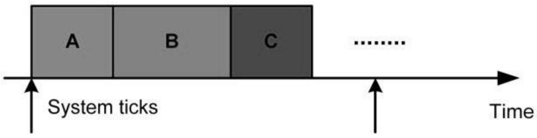


Figure 1: Tasks scheduled to be run in a TT system

If for any reason, task A takes a longer time to run than expected, task C will run before task B (if it has higher priority than task B) and task B will not be able to finish within the system tick as shown in Figure 2.

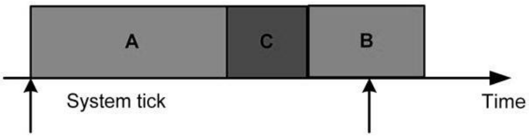


Figure 2: Illustration of overall change in system behaviour if the execution time of task A takes longer than expected

The point to be noted here is, if task A varies in duration it will affect the overall system behaviour. Tasks involving loops and decision structures (e.g., 'if-else', 'switch', etc.) are more likely to have variable execution times. If such tasks can be been balanced, we can achieve more stable and predictable system behaviour.

Solution

SINGLE PATH helps to achieve fixed execution time for tasks involving decision structures and loop statements. The single-path programming approach was introduced by Peter Puschner (Puschner, 2003) as part of his extensive research on WCET analysis. According to single-path programming paradigm, programs that involve loops and decision structures (e.g., ‘if-else’) will have a single execution path. This could be achieved at the expense of higher but fixed and predictable execution time as compared to traditional programming. Single-path can be achieved by replacing input-data dependencies in the control flow by predicated code instead of branched code. Thus, the instructions are associated with predicates and get executed if the predicate evaluates to true. In other case (if instruction evaluates to false), the microprocessor replaces the instruction with a NOP (no-operation) instruction.

Translation of Conditionals

Consider a piece of code where the developer is using an if statement to check whether or not a particular condition is true, as shown in the left hand side code segment in Listing 3. If the condition being evaluated (cond) is true, the value of the variable result is set to expr1 otherwise the value of result is set to expr2. As we cannot be sure which of the two expressions (expr1 or expr2) will be calculated, or in other words, which execution path the code will follow, it becomes difficult to predict the execution time of the section of the task with the conditional statement.

Using Single Path Delay, we assign temporary variables temp1 and temp2 for storing the results of expr1 and expr2 respectively. The conditional move instruction “movt” copies the value of temp1 to the variable result if the test condition evaluates to true, otherwise processor performs a “no operation” (NOP) instruction. On the other hand, if the test condition evaluates to false, “movf” will copy the value of temp2 to result otherwise NOP instruction will be executed. In this way the translation basically generates a sequential code as shown in the right hand side code segment in Listing 1.

```
if(cond)                                 temp1 = expr1;
{                                        temp2 = expr2;
  result = expr1;
 }                   ⇨                   test cond;
else
{                                        movt result, temp1;
  result = expr2;                        movf result, temp2;
}
```

Listing 1: Sequential code generated from a branching statement using if-conversion [adapted from Puschner, 2003]

Translation of loops

Consider a while loop as shown in Listing 4 which executes a set of statements based on two conditions being ‘true’ – a pre-condition, cond-old and a condition, cond-new.

```
--precondition: cond-old

while cond-new do max expr times
      {
      stmts
      }
```

Listing 2: Original while loop [adapted from Puschner, 2003a]

To translate this loop to have a single path of execution, a boolean variable finished is introduced – this variable stores information as to whether the original loop has executed the current iteration or has already terminated. The while loop shown in Listing 9 above can be translated as follows (Puschner, 2003a): First the loop is translated to a simple counting loop (e.g., a for loop) with the iteration count set to be equal to the maximum iteration count of the original loop (in this case, expr). The pre-condition, cond-old, is used to build a new branching statement inside the new loop.

A new conditional statement that has been generated from the old loop condition (cond-new) is transformed into a conditional assignment (using the newly introduced boolean variable finished) with constant execution time. As a result, the entire loop executes in constant time.
This new conditional statement is placed around the body of the original loop and simulates the data dependent termination of the original loop in the newly generated counting loop.

```
finished := false;
for i := 1 to expr do
begin
if not cond-new
then finished := true;
if cond-old and not finished
then stmts
end
```

Listing 3: while loop with a constant execution count [adapted from Puschner, 2003]

Overall strengths and weaknesses

- ☺ Helps to produce constant execution time for code sections involving loops and conditional statements.
- ☹ Its use is limited to hardware which supports “conditional move” or similar instructions.
- ☹ It is likely to increase the power consumption because the CPU will always execute the single-path code for a fixed (maximum) period. During this time, the processor will be in “full power” mode.

References

Puschner, P. (2003). The single-path approach towards WCET-analysable software. IEEE International Conference on Industrial Technology.

TAKE A NAP

Design pattern

Context

- You are using the pattern BALANCED SYSTEM.
- You are using a system which is extremely power constrained.

Problem

How would you ensure the WCET of your application code sections involving loop and decision structures remains constant with reduced power consumption?

Background

SANDWICH DELAY and SINGLE PATH DELAY provide ways to achieve fixed execution time. In systems where power consumption is a concern, neither a SANDWICH DELAY nor a SINGLE PATH DELAY is an attractive solution, because – to achieve balanced code – we need to run the CPU at "full power" at all times. For such systems we need to find out a way to achieve balanced code without any extra power consumption.

Solution

For systems which are extremely resource constrained (especially power) TAKE A NAP provides a way to achieve balanced code with reduced power consumption.

Create balanced code by putting the control flow statement within a 'Sandwich Delay' (see pattern SANDWICH DELAY). This will ensure that the particular piece of code will always have a constant execution time. For example consider the code segment given in Listing 1.

```
for (i = 0; i < x;  i++)
{
  // body of the loop
}
```

Listing 1: Simple For Loop

The execution time of the loop is dependent on the value of the variable x. Let MAX be equal to the maximum number of iterations the loop can execute. Let Time(x) be equal to the time spent in executing x iterations. The value of Time(x) may be measured using hardware timers. Therefore, the time spent in performing (MAX – x) iterations may be calculated using the value of Time(x) as follows:

$$\text{Time (MAX – x) = (MAX – x) * Time(x)/x} \quad \textbf{(1)}$$

Once the for loop executes x number of times, the processor is put to sleep for a duration equal to Time (MAX – x). A timer interrupt may be generated when the hardware timer count reaches the value Time(MAX - x)and this can be used to awaken the processor. Using this technique, code segment in Listing 6 is ensured to always – irrespective of the value of x – have a constant execution time equal to the value of Time(MAX) (i.e. the time spent in executing MAX number of iterations of the for loop). Thus, in addition to enabling a ‘power – saving mode of the processor, the resulting ‘balanced’ code with the SANDWICH DELAY incorporated, provides an additional layer of predictability to the real-time system.

The balanced version of the code segment in Listing 6 may be written as shown in Listing 2.

```
//start the timer
Timer_Start();
for( i = 0; i < x; i++)
{
   //body of the loop
}

//stop the timer
Timer_Stop();

//Store timer count value after x iterations
Time(x) = Timer_Count_Value;

//Determine value of Time(MAX – x)
Time(MAX – x) = (MAX-x) * Time(x)/x;



//Reset the timer
Timer_Reset();

//Set the timer interrupt to occur after duration Time(MAX-x)
Set_Timer_Intterrupt(Time(MAX-x) + “safety margin”);

//Put processor to sleep
Processor_Sleep();
```

Listing 2: Balancing of sections with reduced power consumption

It must be noted that the for loop in code segment above must run at least once for the value of Time(MAX - x) to be determined. Furthermore, a small ‘safety margin’ has been added to the calculated time to ensure that there is sufficient time for the processor to enter sleep mode even when the loop is executed for the maximum number of iterations.

TAKE A NAP may also be applied to other control flow and conditional branching statements such as while, if-else and switch.

Overall strengths and weaknesses

☺ A simple technique for improving system reliability by providing an additional layer of predictability is described here.

☺ Ensures fixed execution time for each task in the system along with reduced power consumption.

☹ The maximum number of iterations of the control flow statement (i.e. the value of MAX) must be known in advance.

☹ Requires access to a hardware timer.

PLANNED PRE-EMPTION

Design pattern

Context

- You are using the pattern BALANCED SYSTEM.
- Your system is based on a time-triggered scheduler – specifically on TTH architecture.

Problem

How would you ensure the predictable scheduler behaviour in TTH designs?

Background

Some background material related to this pattern is already presented in the introduction of this report (see section TTH design).

During normal operation of the systems using the TTH Scheduler architecture, function main() runs an endless while loop (see Listing 1) from which the function C_Dispatch() is called: this in turn launches the co-operative task(s) currently scheduled to execute. Once these tasks are completed, C_Dispatch() calls Sleep(), placing the processor into a suitable “idle” mode.

```
while(1)
   {
   C_Dispatch();  // Dispatch Co-op tasks
   }

void C_Dispatch(void)
   {
   // Go through the task array
   // Execute Co-operative tasks as required
   // The scheduler may enter idle mode at this point
   Sleep();
   }

void P_Dispatch_ISR(void)
   {
   P_Task();
   }
```

Listing 1: TTH Scheduler

A hybrid scheduler provides limited multi-tasking capabilities to the system. Such systems could exhibit unpredictable behaviour because of two reasons (Maaita, 2008):

1. Existence of unbalanced code branches in the timer ISR which leads to variable ISR execution times. This in turn leads to unpredictable scheduler behaviour represented by the appearance of task starting jitter.

2. The existence of CPU instructions with different execution times (i.e. in terms of CPU cycles required to execute the instruction). This leads to variable timer interrupt response times as each of the periodic timer interrupt which take place throughout the life cycle of the application can occur while the CPU is in one of the two different states. The CPU may either be running in sleep (idle) mode shown in Figure 1, or while it is running an instruction and where the interrupt is only serviced once the currently executing instruction is finished shown in Figure 2.

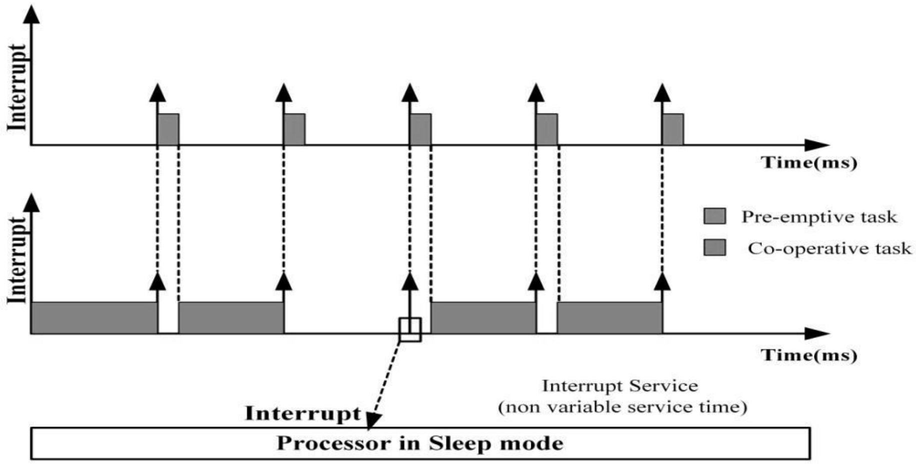


Figure 1: Timer Interrupt when CPU is in sleep mode

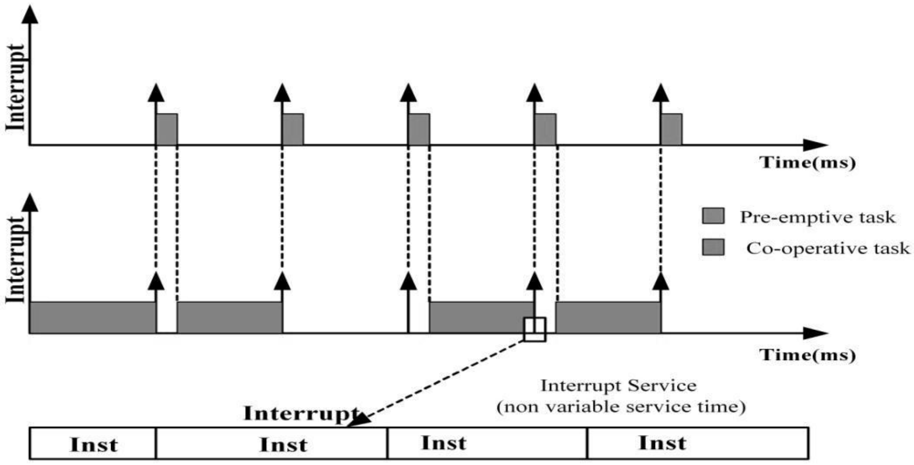


Figure 2: Timer Interrupt when CPU is in sleep mode

The possible occurrences of timer interrupts could lead to variable timer ISR response times translate in to task release jitter. In TTH design this release jitter has the largest impact on tasks which regularly execute after a timer tick has occurred and is, therefore, referred to as “tick jitter”.

Solution

By keeping the processor in the same state as all interrupts takes place would likely to reduce the tick jitter (Maaita and Pont, 2005). Planned pre-emption makes use of another hardware timer to put the processor to power saving mode before the scheduler timer interrupt occurs thus keeping the processor in the same state every time. Power saving mode or sleep/idle mode is available in almost all embedded processor for example ARM7 and 8051 family of processors.

We are naming the extra timer used for this purpose as “PP-timer” being use for Planned Pre-emption. To set the overflow value of the PP-timer it is important to know the WCET in advance so that the processor can have enough time to go to sleep mode before the scheduler timer interrupt occurs. Planned Pre-emption will reduce the tick jitter as the time required to leave the sleep mode and pursue normal execution is a static value (Martin, 2005). Figure 3 and Listing 2 illustrates the implementation of Planned Pre-emption.

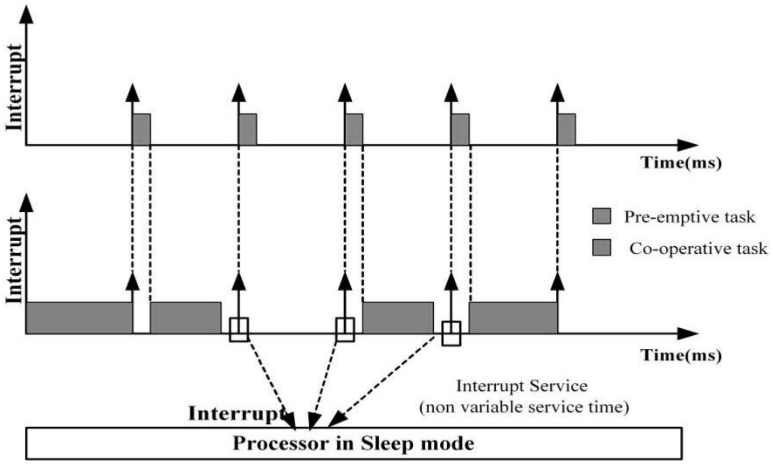


Figure 3: Operation of Planned pre-emption. All interrupts occur when the processor is in sleep mode

```
while(1)
    {
    C_Dispatch();  // Dispatch Co-op tasks
    }
    void C_Dispatch(void)
    {
    //Go through the task array
    //Execute Co-operative tasks as required
    //The scheduler may enter idle mode at this point
    Sleep();
    }

    void P_Dispatch_ISR(void)
```

```
{
ITimer();  //Start idle timer
P_Task();  //Dispatch pre-emptive task
}

void Idle_Timer_ISR(void)
{
Sleep();
}
```

Listing 2: TTH-PP Scheduler

Reliability and safety implications

Designers have to be careful while using the second timer. The timer should overflow after most of the interval between “pre-emptive ticks” has elapsed. A more efficient implementation in terms of the hardware utilised is to use a second match register on the original scheduler timer. For example, ARM7TDMI supports multiple match registers per timer (UM10211, 2009) .

Overall strengths and weaknesses

☺ Produces a more predictable TTH system.

☺ Provides a simple way of getting non variable timer interrupt response times which reduce the tick jitter.

☹ Makes use of additional hardware timer.

☹ Slight increase in memory requirements because of increased code size than normal TTH scheduler.

References

Maaita, A. (2008). Techniques for enhancing the temporal predictability of real-time embedded systems employing a time-triggered software architecture. Department of Engineering, University of Leicester. **PhD Thesis**.

Maaita, A. and M. J. Pont (2005). Using 'planned pre-emption' to reduce levels of task Jitter in a time-triggered hybrid scheduler. Second UK Embedded Forum. Birmingham, UK**:** 18-35.

UM10211 (2009). LPC 23XX User manual Rev. 03.

SYSTEM MONITOR

Abstract pattern

Context

- You are in the process of creating or upgrading an embedded system, based on a single processor.
- Because predictable and highly-reliable system operation is a key design requirement, you have opted to employ a "time-triggered" system architecture in your system.

Problem

How will you make sure that your system will not 'hang' and will keep on functioning despite unfavourable conditions?

Background

If your application is to be reliable, you need to be able to guarantee that system should be capable of handling situations which could possibly hang the system. Some of such possibilities are:

- Incorrect initialisation of hardware peripherals or variables associated with hardware for example ADC or DAC
- Hardware devices may subjected to an excessive input voltage and may not work at all
- Task overrun in the system i.e. if a task exceeds its estimated execution time it will disturb the entire schedule

Solution

In order to ensure system reliability, SYSTEM MONITORS can be implemented to keep an eye on system functionality. They act like guards to be responsible to make sure that system is working fine and can take appropriate actions if anything detected unexpected with the system. Such SYSTEM MONITORS can be implemented with use of hardware for example timers (see pattern WATCHDOG) or a separate software task can be design for this purpose (see pattern TASK GUARDIAN) . Also, There are some good programming practices which could help to avoid possibilities of hanging system see pattern (LOOP TIMEOUT).

Related patterns and alternative solutions

- WATCHDOG
- LOOP TIMEOUT
- TASK GUARDIANS

Overall strengths and weaknesses

☺ System monitors provides a way of ensuring system reliability

☹ Require use of additional hardware or code to implement monitoring techniques.

☹ Implementing such techniques requires care

TASK GUARDIAN

Design pattern

Context

- You are using the pattern SYSTEM MONITORS.
- You application is based on simple TTC design

Problem

How would you ensure that the any task overrun in the system should be detected and handled so that system can continue functioning properly?

Background

Despite many advantages, a pure TTC architecture has a failure mode which has the potential to greatly impair system performance: this failure mode relates to the possibility of task overruns (see Figure 1).

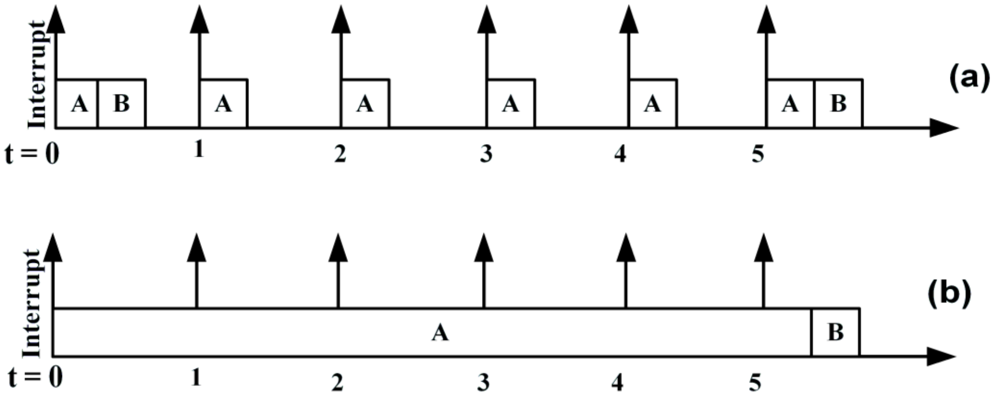


Figure 1: Illustrating the impact of task overrun on TTC based system

Figure 1(a) illustrates a TTCS design running two tasks, A and B. Task A runs every millisecond and Task B runs every 5ms. This system operates as required, since the duration of Task A never exceeds 0.4 ms. Figure 1(b) illustrates the problems that result when Task A overruns: in this case, we assume that the duration of Task A increases to approximately 5.5ms. The co-operative nature of the scheduling in this architecture means that this task overrun has very serious consequences.

In practice, the situation may be even more extreme: in this example, if Task A never completes, then Task B will never run again.

During normal operation of the TTC architecture described by (Pont, 2001), the first function to be run (after the startup code) is Main. Main calls Dispatch which in turn launches any tasks which are currently scheduled to execute. Once these tasks are completed, Dispatch calls Sleep, placing the processor into a suitable “idle” mode. A timer-based interrupt occurs every millisecond (in most implementations) which wakes the processor up from the idle state and invokes the ISR Update. Update identifies tasks which are due to be launched during the next execution of Dispatch. The function calls return all the way back to Main, and Dispatch is called again. The cycle thereby continues (Figure2).

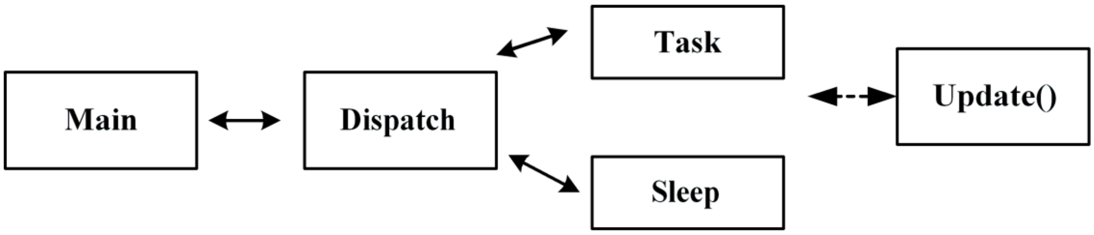


Figure 2: Function call tree for TTC architecture (normal operation)

If a task overrun occurs, then - instead of Sleep being interrupted by the ISR - the overrunning task is interrupted (Figure 3).

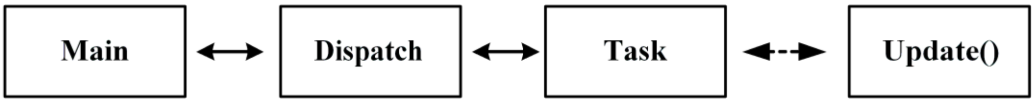


Figure 3: Function call tree for TTC architecture (task overrun)

If Task keeps running then - in the standard TTCS scheduler - it will be periodically (very briefly) interrupted by Update. However, it cannot be shut down.

Solution

The TASK GUARDIAN approach is proposed in (Hughes and Pont 2004) implemented mainly in a revised version of Update - is required to shut down any task which is found to be executing when the Update ISR is invoked.
This will be carried out as follows (see Figure 4):

- The Update ISR will detect the task overrun.
- Update will return control to End_Task (rather than to the problematic task). End_Task will be responsible for unwinding the stack, as required, to locate the return address to Dispatch.
  Control will then be returned to Dispatch, and - as far as possible - normal program operation will continue.

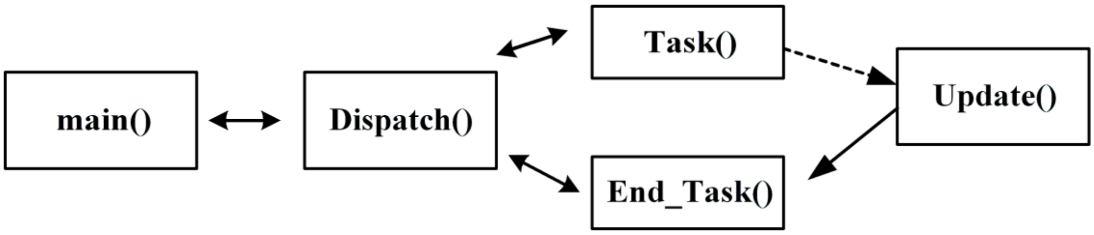


Figure 4: Task Guardian mechanism

Detection of task overrun

In order for the update task function to know that an overrun has occurred and take appropriate action, a simple and reliable method is required to detect overruns. Modifying the code in Dispatch, where the tasks are launched, enables this to be achieved.

Using a variable such as 'Taskover' the task ID can be stored before the task is executed (Listing 15). When the task complete, 'Taskover' is assigned a value of 255, a reserved ID to indicate successful task completion.

```
Taskover = Index                //Store task ID
(*SCH_tasks_G[Index].pTask)();  //Run the task
Taskover = 255;                 //Task completed
```

Listing 1: Detection of task overrun in dispatch

Returning from update

If a task overrun has occurred then Update must alter its return address so that - instead of returning to the overrunning task - it returns to End_Task (see Figure 43).

Please note that the End_Task function is required because Update is an FIQ (Fast Interrupt Request) ISR which - in the ARM architecture used here - has a separate stack and hence a different set of frames: unlike Update, End_Task runs in User mode and therefore has access to the stack and frames used by the overrunning task. It is the job of End_Task to back-trace and rewind the function calls until the return address to Dispatch is located. The end task function then returns control to Dispatch. Update must determine how the return address is stored and check if the address lies outside the critical code labels, indicating that the task has not returned before setting the 'taskover' variable. The original Update return address is stored so that End_Task can determine where the task overruns. The Update return address is then replaced with the start address of End_Task.

Shutting down the task

Having detected a task overrun in Update and changed the return address, control is transferred to End_Task which must shut down the overrunning task.
End_Task determines whether the overrunning task was a leaf function or a function containing sub functions (with frames in the stack). The frame pointer (fp) register is compared with the saved fp register value in Dispatch: if these values are equal, this indicates that the overrunning task is a leaf function. If the values are different then the function calls are back traced (using the ATPCS standard) until the frame-stored fp register is equal to the fp register value saved in Dispatch. The current processor fp register is then made equal to the stored fp register.

End_Task is then able to check if any registers contents were stored on the stack when the task was called. The store instruction is found by looking at the contents of the address referred to by the frame pointer (which identifies the first line of code in the function). By subtracting 12 from this address, locating it and comparing the contents with the instruction number for a block transfer function, a store instruction can be identified.

The store block transfer instruction is then decoded to recover important settings (such as the names of registers which were stored on the stack and whether the stack is ascending or descending). The required bits of the store instruction are modified to convert it to a load instruction. The new instruction is then pointed to by the address in r13, which is loaded into the program counter. The microprocessor then runs this instruction from RAM: this initiates the return sequence to Dispatch with the saved registers restored. Note that all the important registers are stored before and after the task is called to add an extra level of security from register corruption.

If the overrunning task is a leaf function then the end task function simply returns where the r14 (link) register, which contains the return address of the dispatch task function, is loaded into the PC (program counter) register.
When a task is to be shut down a flag is set in Update so that Dispatch will loop through the task array again: this avoids the possibility of a tick offset error.

Overall strengths and weaknesses

☺ Task guardians provides a way of detecting and handling task overrun in the system

☹ Addition of task guardians adds to the complexity of the code.

☹ Controlling the transfer of control from Update to End_Task requires care

References

Hughes, Z. M. and M. J. Pont (2004). Design and test of a task guardian for use in TTCS embedded systems. UK Embedded Forum, Published by University of Newcastle.

APPENDIX C: DOCUMENTS RELATED TO EMPIRICAL STUDIES

Exercise for Experiment 1

Important Instructions:

- You are given with example embedded applications designed with event-triggered and/or pre-emptive architectures. Your job as a system designer is to convert these applications to time-triggered design
- You are not required to implement any system programmatically BUT you are required to analyse each system design and answer the questions given at the end of each exercise.
- In each of the system you should consider that highly predictable behaviour is the key design requirement and every system has hard timing constraints (highly precise timing is a requirement)

Please answer your questions as clear as possible

System A: Traffic Light System

You are quite familiar with the Traffic light system in the UK. As an embedded system developer you are asked to work on a project which is about improving the software architecture of controller for the traffic lights and pedestrian crossing lights used at a typical crossroads in the UK:
UML state diagram of this system is given below:

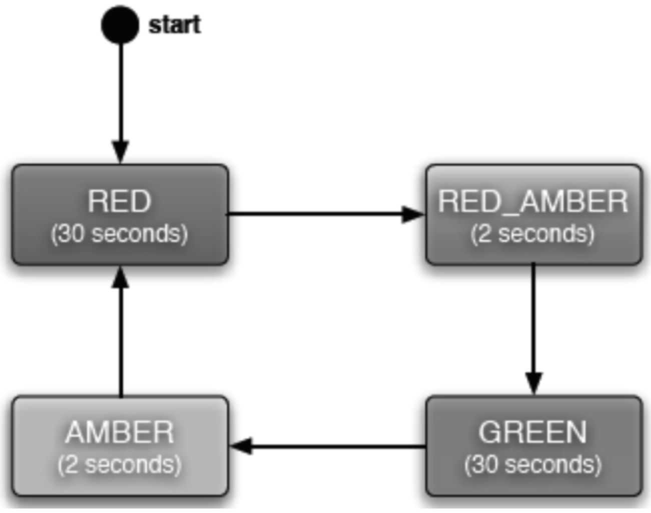


Figure 1: State diagram for the Traffic Control System in UK

As you can see in the photo above, crossroads can have traffic lights in each of the four directions and pedestrian crossing lights on all four sides. A pedestrian crossing light consists of two images — a green walking man and a red standing man — which tell a pedestrian if it is safe to cross. In addition, there is a button used to indicate a pedestrian's desire to cross and a 'wait' signal, showing that the button has been pressed:

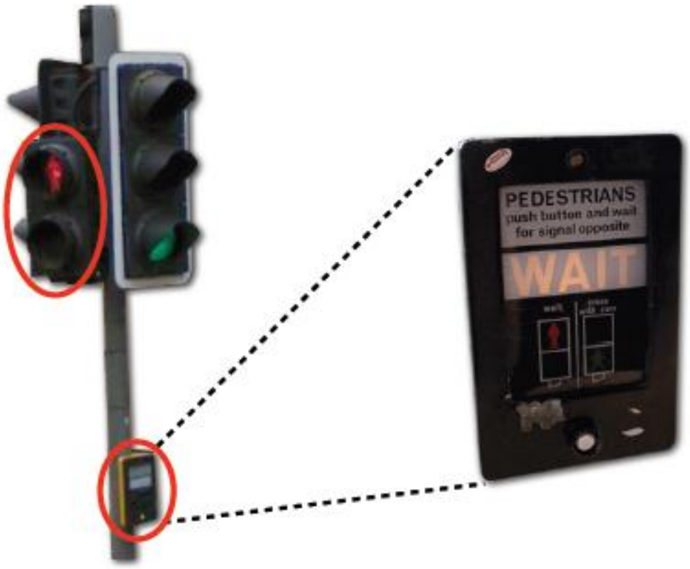


Figure 2: Pedestrian crossing for the Traffic Control System in UK

Under normal operation, these traffic lights will cycle through the usual states shown earlier, with one light always in the red state. The light currently not in the red state must transition all the way to red before the second set may leave the red state.
When a pedestrian presses the button the ‘wait’ indicator will light up. When the system reaches ***one*** of the states where both lights have transitioned to red, the pedestrian crossing light will transition to green. At this point, the wait indicator should turn off and pedestrians are given time to cross. As a warning to both drivers and pedestrians, returning to normal operation after a crossing will involve a state where the amber traffic and green pedestrian lights both flash for a period of time.

The current implementation of this system is based on a real time operating system called FreeRTOS (an open source for embedded applications). The overall architecture of the system is pre-emptive. There are following tasks in the system with the following specifications.

Table1: Task specifications for System A

Task Name	Priority	Delay/Period	Execution Time	Pre-emptive
Heartbeat	3	50ms	50 µsec	Y
Button test	2	10ms	75µsec	Y
Update Lights	1	500ms	18ms	Y

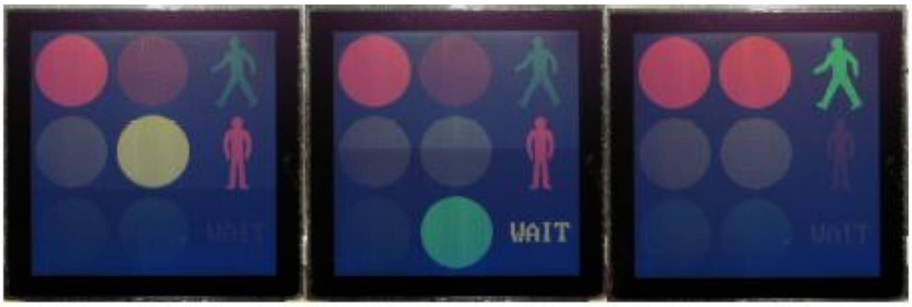


Figure 3: Output for System A on LCD

The system is simulated graphically using the LCD of LPC-2378 development boards. ***The LCD is a shared resource*[25] *and is handled it accordingly*** with API available in the FreeRTOS for use with critical sections.

Because of the overheads in the system are high it is decided to port to the entire system to a more reliable architecture which incur lower overheads as much as possible.

As a system designer you are asked to resolve the following issues.

1. What type of scheduler is your choice for converting this system to time-triggered architecture?
2. How will you incorporate shared resource protection in your proposed system? Explain
3. What are the pros and cons of the architecture you chose?
4. Once you achieve "time driven architecture you are required to achieve an optimized/balanced time driven system with jitter reductions wherever possible
5. Can you implement any of the monitoring techniques to keep an eye on the system that it should not hang? Briefly discuss your technique

System B: Data Acquisition System

You are given with a simple embedded system called DAQ implemented on ARM7 LPC2378STK. This system takes readings from an analogue-to-digital (ADC) converter available on board translates the value to an appropriate string of characters and transmits the resulting information to the on board LCD controller when a user press BUT1 on the board. In addition, it can also display an elapsed time on LCD since the microcontroller was rest last when a user press BUT2.

The system has hybrid design (event + time triggered) because it is running aperiodic as well as periodic tasks.

There are three interrupts working in the system. There is a TIMER0 interrupt which is controlling all the periodic tasks in the system and update all the tasks every one second. In addition, there are two external interrupts (EINT0 and EINT3) working in the system too. Each of the external interrupt is called when a user presses a button to see ADC value or the elapsed time value on the LCD screen. For each of the interrupts their interrupt service routines (ISR) are defined to handle these interrupts

[25] A shared resource is one which is accessed by more than one task at the same time. Shared resources includes areas of memory (two tasks needs to access the same global variable) or hardware such as ADC. The code which accesses such shared resources are referred to as a "critical section"

Table 2: Task specifications for System B

Task name	Period (ms)	Execution time (µsec)	Pre-emptive
ADC_Sample	5	62	N
ADC_Translate	500	68	N
ADC_Display	500	325	N
	Inter-arrival time[26]	Execution time	
ISR_EINT3	2000	200	Y
ISR_EINT0	2000	250	Y

As a system designer you are asked to migrate this system to a fully time-triggered design

1. As a first step what changes will you think about in the system?
2. What type of scheduler is your choice for converting this system to time-triggered architecture?
3. What value of tick interval will you choose for your system?
4. What are the pros and cons of the architecture you chose?
5. Once you achieve "time driven architecture you are required to achieve an optimized time driven system with jitter reductions wherever possible i.e. how can you achieve a balanced system?
6. Can you implement any monitoring techniques, which could keep an eye on the system so that it should not hang in case of any unexpected errors

System C: FFT Analyser

FFT Analyser is based on two main activities (1) data sampling using analogue-to-digital converter (ADC) and (2) time-frequency conversion using Fast Fourier Transform (FFT). Such a setup is common in medical devices for example ECG machines. This simple embedded application samples the generated signal (at 1 KHz), carry out a Fast Fourier Transform on the sampled data and finally output the first harmonic frequency to the hyper-terminal using RS-232 if a user pressed a push button available on the board. Otherwise the system keeps on displaying "FFT Analyser" on hyper-terminal.

Current implementation of this system is completely event-triggered and pre-emptive with all the tasks in the system are pre-emptive and button press is an event-triggered task.

[26] Inter-arrival time is the minimum expected time for an interrupt to occur

Table 3: Task specifications for System C

Task name	Period (ms)	Priority	Execution time
ADC_Sample	1	4 (highest)	62µsec
FFT_Convert	50	3	18ms
LCD_Display	500	2	270 µsec
Flashing_LED	500	2	27 µsec
UART_Transmit	10	1(lowest)	82 µsec
	Inter-arrival time[27]		
ISR_EINT3	2000ms		200 µsec

As a system designer you are asked to translate this system to a fully time-triggered design.

1. What type of scheduler is your choice for converting this system to time-triggered architecture?
2. Are there any flaws in your chosen design? How can you overcome them?
3. Can you further optimize your system to make it more safe? If yes, please explain how?

[27] Inter-arrival time is the minimum expected time for an interrupt to occur

Exercise for Experiment 2

Instructions:

- Solve all the exercises given below.
- Please record the start time and finish time for each question.
- Please try to write short but clear answers. To help you solve this exercise you are provided with related research papers and other materials, you can make use of them.
- Please provide references of given documents (if you will use any of them) in your answers.

Start time:

Task 1:

You are given with an embedded application design running three different tasks A, B and C. Each of the tasks invoke when an event happens. Examples of some events are pressing a switch, arrival of a character ready to be displayed and completion of some analogue to digital conversion. Prototype of the systems is given in Listing 1 below: Please note that to serve each of the interrupts a separate interrupt service routine is provided to get the desired functionality. Such a design is called an event-triggered design.

```
void main(void)
{
/* Initialization of tasks*/
   A_init();
   B_init();
   C_init();

   EA = 1 ; // Enable all interrupts

while(1)
{
  PCON |= 0x01; //Put the processor in idle mode
 }
 }
/*Definition of Interrupt Service Routines*/

void A_ISR(void) interrupt IEIndex
{
   //code to get the desired functionality
}

void B_ISR(void) interrupt IEIndex
```

```
  {
//code to get the desired functionality
  }
  void C_ISR(void) interrupt IEIndex
  {
     //code to get the desired functionality
  }
```

Listing 1: Prototype for the ET system

You are now required to convert this system to a time-triggered design.

i. List at least two necessary changes will you do in your system?
ii. Rewrite the given system prototype reflecting a time-triggered design.

Finish time:

<u>Task 2:</u>

Start time:

(a) Time-triggered systems which are designed with due care are still susceptible to jitter. The presence of jitter can make the system unreliable. Assuming yourself as a system designer list at least two techniques (with brief details) you will employ during system design to handle different situations to make your system more reliable and less vulnerable to jitter.

Finish time:

Start time:

(b) Some people argue that there are benefits in system design where best case execution time (BCET) of tasks is similar or identical to worst case execution time (WCET) of the tasks. Assuming that creation of code in this format is a requirement for your organisation, rewrite the code segment given below to meet these requirements.

```
if (x >= 10)
{
  z = 0;
      }
      else
      {
  z = 1;
       }
```

Finish time:

Start time:

(c) For the above question (b) do you think that the approach you have adopted to make BCET=WCET could affect power constrained systems?

i. YES

ii. NO

Finish time:

Start time:

(d) If your answer is YES could you suggest any changes in your proposed technique

Finish time:

Start time:

Task 3:

One serious issue in a reliable system design is task overrun. A task overrun occurs when a task is unable to finish within its estimated execution time as shown in Figure 1 below:

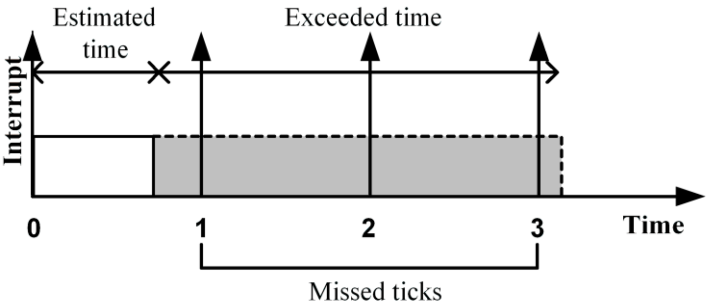


Figure 1: Task overrun in the system

This can raise serious problems for real-time systems which must react within precise time-constraints. Task overrun can essentially 'hang' the entire system (until reset) waiting for some specific operation to be completed.

In this exercise you are given with a code snippet which is prone to such a situation. One of the tasks in your application is responsible for analogue to digital conversion (ADC) and includes the following code

```
//wait until AD conversion finishes
 While( (ADCON & ADCI) == 0);
```

This code is potentially unreliable because there are circumstances in which this application could hang. For example, if the ADC device is not properly initialized OR if it has been subjected to an excessive input voltage, the expression written above might not be safe.

What necessary changes you can make in the code given above to make it more reliable. Please rewrite code with your suggested changes.

Finish time:

Start time:

Task 4:

For the task set given below you need to choose appropriate offset such that all task should meet their deadlines (as Task C is missing its deadline with current parameters Shown in Figure below). Redesign task set (change values as you think appropriate to meet all task deadline).

Table 1: Task parameters that leads to missed deadline

Task	WCET(ms)	Deadline (ms)	Period (ms)	Offset (ticks)
A	2	10	5	0
B	3	10	10	0
C	4	10	10	0

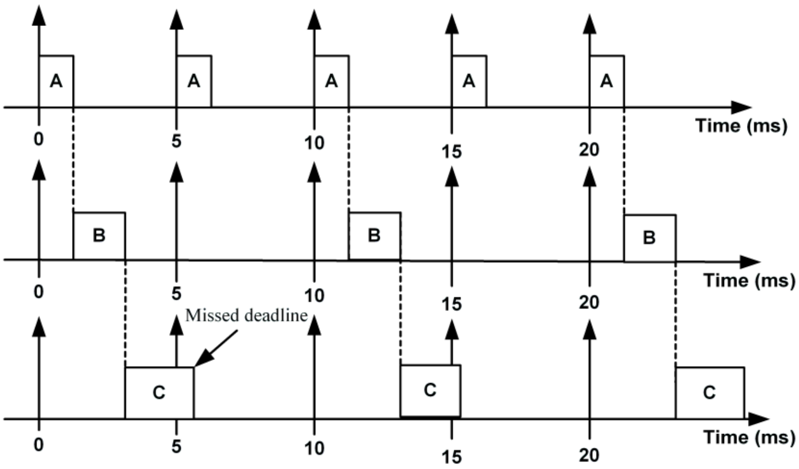


Figure 2: Timeline for tasks shown in Table 1

Table 2: Task parameters which should prevent missing deadline

Task	WCET(ms)	Deadline (ms)	Period (ms)	Offset (ticks)
A	2	10	5	
B	3	10	10	
C	4	10	10	

Do you think changing task order can affect the jitter in the system? If yes, please rearrange the task order. Make a new table with new order of tasks and if possible draw timeline as well.

Finish time:

Questionnaire for Pattern Participants

- How difficult did the exercise appear to you given there was a requirement to give extra effort to find the relevant information from the additional resource material provided ?.
  1. Too easy
  2. Easy
  3. Difficult
  4. Too difficult

- Did you find the given documents helpful to solve the tasks given in the exercise? Mark a separate number(BW 1 and 4 given below) for each exercise in the table below:
  1. Not Helpful
  2. A little bit helpful
  3. Helpful
  4. Very helpful

Exercise 1	Exercise 2	Exercise 3	Exercise 4
	a) b) c) d)		

- Was it easy to find related information from the patterns given to you? Mark a separate number (BW 1 and 4 given below) for each exercise in the table below:
  1. Too easy
  2. Easy
  3. Difficult
  4. Too difficult

Exercise 1	Exercise 2	Exercise 3	Exercise 4
	a) b) c) d)		

- What do you think that language used in patterns , was it
  1. Too Easy
  2. Easy
  3. Difficult
  4. Too Difficult

5.

- If you are asked to rate the level of difficulty (for individual exercise) to find relevant information from patterns given to you (1-4, 1= least difficult 4 = most difficult) what would be your answer? Just write a number between 1 and 4. If you think there is no information available mark 0 in front of that exercise.

Exercise 1	**Exercise 2**	**Exercise 3**	**Exercise 4**
	a) b) c) d)		

- In your opinion to what degree the name of a pattern helped you in guessing what it is about or to which problem it is referring to: (answer between 1 and 4 1= lowest and 4 = highest)

	1	**2**	**3**	**4**
Exercise 1				
Exercise 2				
Exercise 3				
Exercise 4				

- Write down the name of the patterns which you chose to solve each of the following exercises

Exercise 1	**Exercise 2**	**Exercise 3**	**Exercise 4**
	a) b) c) d)		

- Do you think that the patterns are described in a language that is understandable and the root causes of the problem are identified and addressed by the solution
  1. YES
  2. NO

- As a computer scientist, do you feel that the given set of patterns can be operational or implementable (in any programming language you know) as described?
  1. YES
  2. NO

- In the table below can you please rate the overall pattern collection given to you (1-5, 1 = lowest and 5=highest) for the criteria given in each column

Patterns useful for given exercise	**Patterns are useful for Design Task**	**Patterns are useful for other projects**

- In the table below can you please rate the individual patterns (1-5, 1=lowest and 5=highest) for the criteria given in each column

	Pattern useful for given exercise	**Easy to Understand**	**Patterns are useful for other projects**
EVENTS TO TIME			
BALANCED SYSTEM			
SINGLE PATH			
TAKE A NAP			
LOOP TIMEOUT			
CONFIGURING TASK PARAMETERS			

- In the table below can you please rate the individual sections of individual patterns (1-5, 1=lowest and 5=highest) for their clarity

	Name	**Context**	**Problem**	**Solution**
EVENTS TO TIME				
BALANCED SYSTEM				
SINGLE PATH				
TAKE A NAP				
LOOP TIMEOUT				
CONFIGURING TASK PARAMETERS				

- Your overall experience of this exercise? Choose one of the option below:
  1. Not interesting
  2. Least Interesting
  3. Interesting
  4. Very Interesting
- Please give your suggestions about making pattern collection more useful

- Did you refer to any of the materials other than patterns (research papers/books) provided to you?
  1. YES
  2. NO

- If YES did you find them more useful than patterns?
  1. YES

2. NO

- In your opinion why a pattern collection is a better choice than research papers/books in real practice to solve problems like the one you were given with.

Questionnaire for Non-Pattern Participants

- How difficult did the exercise appear to you given there was a requirement to give extra effort to find the relevant information from the additional resource material provided?
  1. Too easy
  2. Easy
  3. Difficult
  4. Too difficult

- Did you find the given documents helpful to solve the tasks given in the exercise? Mark a separate number(BW 1 and 4 given below) for each exercise in the table below:
  1. Not Helpful
  2. A little bit helpful
  3. Helpful
  4. Very helpful

Task 1	Task 2	Task 3	Task 4

- Was it easy to find related information from the papers/materials given to you? Mark a separate number(BW 1 and 4 given below) for each exercise in the table below:
  1. Too easy
  2. Easy
  3. Difficult
  4. Too difficult

Task 1	Task 2	Task 3	Task 4

- What do you think about the language used in given research papers , was it
  1. Too Easy
  2. Easy
  3. Difficult
  4. Too Difficult

- If you are asked to rate the level of difficulty (for individual exercise) to find relevant information from papers/materials available (1-4, 1= least difficult 4 =

most difficult) what would be your answer? Just write a number between 1 and 4.

If you think there is no information available mark 0 in front of that exercise.

Task 1	**Task 2**	**Task 3**	**Task 4**

- In your opinion to what degree the title of research papers helped you in guessing what it is about or to which problem it is referring to: (answer between 1 and 4 1= lowest and 4 = highest)

	1	**2**	**3**	**4**
Task 1				
Task 2				
Task 3				
Task 4				

- Your overall experience of this exercise? Choose one of the options below:
  - ○ Not interesting
  - ○ Least Interesting
  - ○ Interesting
  - ○ Very Interesting

- Any other comments

APPENDIX D: DOCUMENTS RELATED TO INDUSTRIAL EVALUATION

Questionnaire

1. What defines your area of expertise? *

□ What defines your area of expertise?   Aerospace industry

□ Automotive industry

□ Control Engineering

□ Consumer appliances development

□ Military applications development

Other (please specify)

2. What defines your current status in the organisation?*

□ What defines your current status in the organisation?   System Engineer

□ Hardware Designer

□ Software Designer

□ Programmer

□ Testing Engineer

□ Project Manager

Other (please specify)

3. In your opinion how the industry is coping with the problem of migration between different software architectures and what are the main challenges around this? *

4. Do you agree that software for future versions of systems will very rarely be created from scratch instead existing software will be adapted to match the new requirements such as "Safety Integrity Levels"? *

● Yes

○ No

○ May be

Other (please specify)

5. What is your opinion about a system of documented patterns which could guide the designers/developers of embedded applications at various stages during the process of changing the underlying software

architecture (such as some of those which you reviewed during the site tour) is *

○ Very useful

○ Useful with further support

○ Rarely useful

○ Not useful

Other (please specify)

6. Do you think that the information provided in the patterns "TIME FOR TT?", "TT SCHEDULER" and "EVENTS TO TIME" which you reviewed during the site tour are really useful for users to help them decide if TT (or ET) architecture is appropriate for their application and to carry out the transformation? *

○ Yes

○ No

○ May be

Any more comments (please specify)

7. As you reviewed the pattern "BALANCED SYSTEM" and associated patterns, do you think that the techniques described in these patterns can help in achieving system safety-integrity requirements? *

○ Yes

○ Yes, to some extent

○ No, not all

Any more comments (please specify)

8. Do you think that the techniques described in the pattern "System Monitor" can help in achieving fault tolerant systems? *

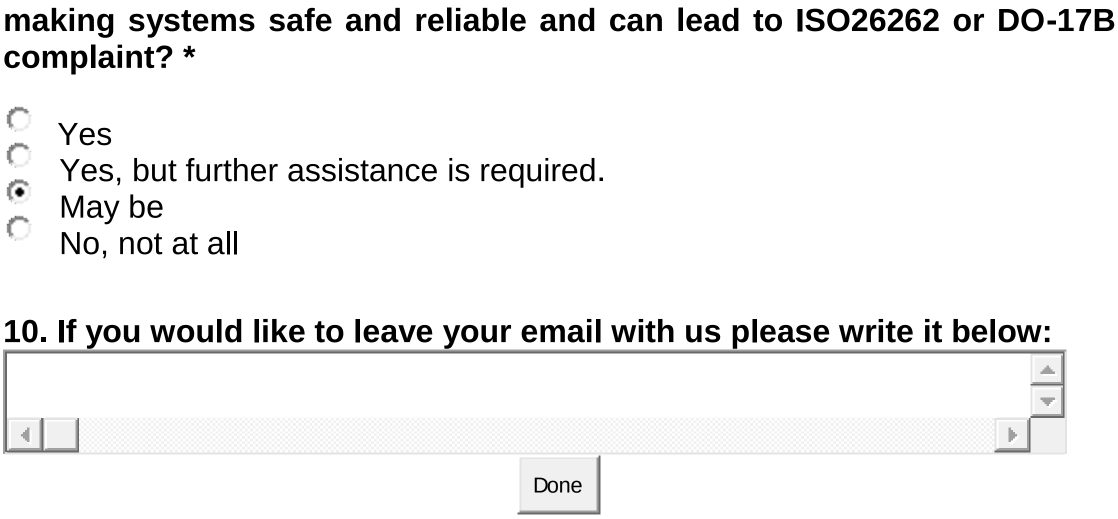
○ Yes

○ Yes, to some extent

● No, not all

Other (please specify)

9. As you have reviewed the patterns during the site tour do you think that if applied carefully the techniques described in them can help in



making systems safe and reliable and can lead to ISO26262 or DO-17B complaint? *

○ Yes
○ Yes, but further assistance is required.
● May be
○ No, not at all

10. If you would like to leave your email with us please write it below:

Summary of Results

1. What defines your area of expertise?		
Options	**Response %**	**Response count**
Aerospace industry	17.9%	5
Automotive industry	32.1%	9
Control Engineering	46.4%	13
Consumer appliances development	35.7%	10
Military application development	10.7%	3
Others		6
Answered question		28

2. What defines your current status in organisation you are working?		
Options	**Response %**	**Response count**
System Engineer	32.1%	9
Hardware designer	32.1%	9
Software designer	64.3%	18
Programmer	17.9%	5
Testing Engineer	7.1%	2
Project Manager	17.9%	5
Others		4
Answered question		28

4. Do you agree that software for future versions of systems will very rarely be created from scratch instead existing software will be adapted to match the new requirements such as "Safety Integrity Levels"?		
Options	**Response %**	**Response count**
Yes	71.4%	20
No	7.1%	2
May be	21.4%	6
Answered question		28

5. What is your opinion about a system of documented patterns which could guide the designers/developers of embedded applications at various stages during the process of changing the underlying software architecture (such as some of those which you reviewed during the site tour) is		
Options	**Response %**	**Response count**
Very useful	32.1%	9
Useful with further support	67.9%	19
Rarely useful	0.0%	0
Not useful	0.0%	0
Answered question		28

6. Do you think that the information provided in the patterns "TIME FOR TT?", "TT SCHEDULER" and "EVENTS TO TIME" which you reviewed during the site tour are really useful for users to help them decide if TT (or ET) architecture is appropriate for their application and to carry out the transformation?

Options	**Response %**	**Response count**
Yes	71.4%	20
No	3.6%	1
May be	25.0%	7
Answered question		28

7. As you reviewed the pattern "BALANCED SYSTEM" and associated patterns, do you think that the techniques described in these patterns can help in achieving system safety-integrity requirements?

Options	**Response %**	**Response count**
Yes	28.6%	8
Yes to some extent	64.3%	18
No not at all	7.1%	2
Answered question		28

8. Do you think that the techniques described in the pattern "System Monitor" can help in achieving fault tolerant systems?

Options	**Response %**	**Response count**
Yes	28.6%	8
Yes to some extent	57.1%	16
No not at all	14.3%	4
Answered question		28

9. As you have reviewed the patterns during the site tour do you think that if applied carefully the techniques described in them can help in making systems safe and reliable and can lead to ISO26262 or DO-17B complaint?

Options	**Response %**	**Response count**
Yes	10.7%	3
Yes but further assistance is required	60.7%	17
May be	25.0%	7
No, not at all	3.6%	1
Answered question		28

Original Responses to Question 3

Respondent 1
Getting the whole organisation or team to buy into a migration strategy/process is a big problem. It could be done in stages and within smaller teams before the process is finally adopted by the whole organisation

Respondent 2
Software architectures differ with each project and application and the industry hasn't been able to come up with a standardised architecture to deal with all applications. Legacy codes and tools are one of the biggest challenges when it comes to migrating to different software architecture.

Respondent 3
I think most of the software architectures are tailored for specific needs and there is rarely any migration to different architecture which creates significant change with the system.

Respondent 4
Most businesses are concerned with a quick as possible time to market. Migration across software architectures raises this significantly. It can also often be hard to convince others that a different way of thinking can benefit them or the business as a whole.

Respondent 5
People of industry (not all) are unaware about it but it still requires some more clarification regarding TT and ET. i.e. how to apply TT in the system? How am I going to migrate? Do I really need to migrate? If my system is running perfectly without any huddles why should I need to migrate? (However all the comments may be useful/not-useful because I am not a very highly experienced person I have 2 years of experience of automation and control industry).

Respondent 6
I don't see migration away from interrupts in telecom. The problems associated with multi-core and parallel hardware need solution and migration more urgently than event-driven system challenges.

Respondent 7
I think that people working in the industry (I am speaking about the managers and big boss) tend to keep traditions. I think it is really a hard task to convince big managers to change their way of work that have been adopted for a number of projects (and years). Big managers have fear of switching to other options as their main goal is reaching time to market.

Respondent 8
Coping badly

Respondent 9
Thorough background knowledge of software architectures is required in order for successful migration. Some companies are not willing to explore on the new or

different architectures due to several factors: [1] cost - companies are not willing to buy new tools to support different architecture [2]lack of manpower - developers need to quickly adapt to new architecture. Due to current economic climate, companies are not willing to hire new people to explore on the newer or different architecture. [3] time to market - by adopting new architecture, time to market will be increased due to learning curve that developers need to endure.

Respondent 10
Every company has its own strategy.

Respondent 11
Plan migration and portability at design time by implementing firmware to be compliant with recognized company or industry wide standards such as AUTOSAR.

Respondent 12
The main challenge from my point of view is that the industry is very much plan-driven and nowadays you have to respond to change more quickly. So more agile approaches are needed to create the software. In many cases, we use subcontractors and communicating the domain knowledge and the architecture to them is troublesome. For me, it is not very important if the system is time-triggered or event-triggered. Currently we are developing in event-driven fashion and no plans to change that.

Respondent 13
Migration between software architectures is not easy in aerospace industry because of the cost involved in rigorous verification to prove the new changes are airworthy and getting the changes certified by authorities (European Aviation Safety Agency EASA).

Respondent 14
Understanding "tribal knowledge" not captured in the formal documents of a system.

Respondent 15
Present ET architecture is familiar to us , TT arch implementation new and no explanation on that and no support

Respondent 16
This migration requires expertise and good software development practices. Sometimes, this migration might not be that straight forward at all and might require scrapping the previous software design and developing it from scratch. I think the industry needs to be more educated on this problem.

Respondent 17
Migration between software architectures is a time consuming task and typically not interesting for engineers who are focused on implementing real world solutions. With the need to integrate embedded systems with other systems becoming more and necessary engineers are expected to know many software architectures particularly those centred around the Internet.

Respondent 18

I have just been involved in writing a proposal where the architecture has moved from co-operative to pre-emptive and now porting parts back to co-operative (interim legacy support). There's no easy way to set about doing this, especially as libraries expect pre-emptive threads. Generally the industry knows that the CPU isn't getting faster and that multi thread/process/processor is the way forward but that it is hard to do. Functional languages are suggested but that has never really taken off. Clever additions to the language are being tried, but it's always going to be hard - the emphasis is pre-emptive rather than co-operative or hybrid. I don't see that changing soon.

Respondent 19

Migration is very poorly done due to lack of knowledge and lack of good migration tools. The migration process itself is usually completely or partially manual, and error prone.

Respondent 20

For small systems, ET is widely used. For more complex systems, a RTOS is an easy solution.

Respondent 21

In my opinion, migrating software architecture from one to another requires substantial effort and cost, and often relies on experience engineers to do it manually. Companies are usually reluctant to invest such a large amount of money to do the software architecture migration, unless a good profit return is envisaged. Of course, a methodology that can simplify the migration process is desirable; since it will maximise the profit margin if software architecture migration is necessary.

Respondent 22

I really don't know the answer to this.

Respondent 23

Software architecture migration involves several challenges like resource constraints, hardware availability and dependency and I am not sure how industry is coping.

Respondent 24

Consider a company which is using event triggered architecture for last 10 years with well-trained staffs. It is bit difficult for them to migrate into TT within a short period. Challenges they may have, 1) Cost 2) Short period 3) Need to train current staffs or need to recruit experienced people.

Respondent 25

Industry is not ready to move from the classical ET architecture until and unless some really excellent tools help them to migrate easily. Certainly Patterns can be a help but need lot of automation in the process.

Respondent 26

At work currently we design from scratch for new processor architecture or new software architecture. Our control software design is primarily based on a time-triggered approach. However, user interface applications tend to be predominantly event-based. The challenge is usually in integrating these two approaches where they meet or need to interface.

Respondent 27

I believe that industry is moving towards Agile Model Driven Development (AMDD). (http://www.agilemodeling.com/essays/amdd.htm)

Respondent 28

It is a very mixed bag depending on the coders background. One big issue I find is a lot of people new to writing embedded C starting learning to code for PCs and so have never had to think about issues like CPU or memory resources and lack the understanding of the electronics in the microcontroller to debug issues. This can result in rather "heavy" and unreliable code (littered with ISRs which can create unpredictable behaviour).

PART E: REFERENCES

To acknowledge the efforts of all those who contributed to the completion of this research, references are provided and it has been considered that the accuracy of these references should be provided to the best of knowledge.

Adams, M., J. Coplien, et al. (1996). Fault-tolerant telecommunication system patterns. Pattern Languages of Program Design 2. Boston , MA, USA, Addison-Wesley 549-562.

Agerbo, E. and A. Cornils (1998). How to preserve the benefits of design patterns. Proceedings of the 13th ACM SIGPLAN Conference on object-oriented programming, systems, languages and applications, ACM 134-143.

Albert, A. and R. Bosch GmbH (2004). Comparison of event-triggered and time-triggered concepts with regard to distributed control systems. Proceedings of the Embedded World 2004. Nurnberg**:** 235-252.

Albert, A. and W. Gerth (2003). Evaluation and Comparison of Real-time Performance of CAN and TTCAN. Proceedings of the 9th International CAN in Automation Conference. Munich, Germany**:** 05-01 to 05-08.

Alexander, C. (1964). Notes on the synthesis of form, Harvard University Press.

Alexander, C. (1979). The timeless way of building, Oxford University Press.

Alexander, C., S. Ishikawa, et al. (1977). A pattern language, Oxford University Press.

Alexander, C., M. Silverstein, et al. (1975). The oregon experiment, Oxford University Press.

Allworth, S. T. (1981). Introduction to real-time software design, Macmillan.

Almeida, L., P. Pedreiras, et al. (2002). "The FTT-CAN protocol: Why and how." IEEE Transactions on Industrial Electronics 49(6): 1189 - 1201

ALTERA. (2012). "DE2-70 Development and Education board." Retrieved March 2012, from http://www.altera.com/education/univ/materials/boards/de2-70/unv-de2-70-board.html. .

Ambler, S. W. (1998). Process patterns, Cambridge University Press.

Appleton, B. (2000). "Patterns and software: Essential concepts and terminology." Retrieved 16th June 2011, from http://www.cmcrossroads.com/bradapp/docs/patterns-intro.html#FurtherInfo.

Aras, K. (2005). Empirical analysis of design patterns - A case study in COMPUCELL3D. Masters Thesis, University of Notre Dame.

Aras, K., T. Cicovski, et al. (2005). Empirical evaluation of design patterns in scientific application, Technical Report TR-2005-08, Department of Computer Science and Engineering, University of Notre Dame.

Arisholm, E. and D. I. K. Sjoberg (2004). "Evaluating the effect of a delegated versus centralized control style on the maintainability of object-oriented software." IEEE Transactions on Software Engineering 30(8): 521-534.

ARM (2012). Migrating from 8051 to Cortex Microcontrollers Application Note 237.

Audsley, N. C., A. Burns, et al. (1991). Real-time scheduling: The deadline monotonic approach. Proceedings of the 8th IEEE Workshop on Real-time operating Systems and Softwares, Atlanta, USA 133-137.

Ayavoo, D. (2006). The development of reliable X-by-Wire systems: Assessing the effectiveness of a simulation first approach. University of Leicester.

Ayavoo, D., M. J. Pont, et al. (2005). A Hardware-in-the-Loop testbed representing the operation of a cruise-control system in a passenger car. Proceedings of the 2nd UK Embedded Forum. Birmingham, UK, Published by University of Newcastle upon Tyne**:** 60-89.

Baker, T. P. and A. C. Shaw (1988). "The cyclic executive model and Ada." Real-time Systems Springer Netherlands 1(1): 7-25.

Balarin, F., L. Lavagno, et al. (1998). "Scheduling for embedded real-time systems." IEEE Design and Test of Computers: 71-79.

Barr, M. (1999). Programming embedded systems in C and C++, O'Reilly & Associates.

Barr, M. (2006) "Multitasking alternative and the perils of pre-emption." Embedded Systems Design, Issue January 2006.

Bate, I. J. (1998). Scheduling and timing analysis for safety-critical real-time systems. PhD Thesis, University of York.

Bate, I. J. (2000). "Introduction to scheduling and timing analysis." The use of Ada in real-time system 6.

Baurah, S. K., G. C. Buttazo, et al. (1999). Scheduling periodic task systems to minimize output jitter. Proceedings of the 6th International Conference on Real-time Computing Systems and Applications, RTCSA. Hong Kong**:** 62-69.

Bautista-Quintero, R. and M. J. Pont (2008). "Implementation of H-infinity control algorithms for sensor-constrained mechatronic systems using low cost microcontrollers." IEEE Transactions on Industrial Informatics 16(4): 175-184.

BCC. (2012). "BCC research report number IFT016D. ." Embedded systems technologies and market Retrieved September

2012, from http://www.bccresearch.com/report/embedded-systems-technologies-markets-ift016d.html.

Beck, K., J. O. Coplien, et al. (1996). Industrial experience with design patterns. Proceedings of 8th International Conference on Software Engineering, Berlin, Germany 103-114.

Begole, B. (2011). Section Title: Background and capabilities of ubiquitous computing. Ubiquitous Computing for Business, Pearson Education Ltd.

Berczuk, S. P. and B. Appleton (2003). Software configuration management patterns: Effective teamwork, practical integration, Addison-Wesley Professional.

Bergin, J. (2000). Fourteen pedagogical patterns. Proceedings of the 5th European Conference on Pattern Languages of Programming. 11.

Berry, D. M. and W. F. Tichy (2003). "Comments on formal methods application: An empirical tale of software development." IEEE Transaction on Software Engineering 29(6): 567-571.

Birgisson, R., J. Mellin, et al. (1999). Bounds on test effort for event-triggered real-time systems. Proceedings of the 6th International conference on Real-time Computing Systems and Applications (RTCSA) 212-215.

Boassan, M. (1995). "The artistry of software architecture." IEEE Software 12(6): 13-16.

Bock, P. (2001). Getting it right: R&D Methods in Science and Engineering, Academic Press.

Borchers, J. O. (1999). Designing interactive music systems: A pattern approach. Proceedings of the 8th International Conference on Human Computer Interaction: Ergonomics and User Interfaces. 1**:** 276-280.

Bottomley, M. (1999). A pattern language for simple embedded systems. Proceedings of the Pattern Languages of Programming PLOP '99. Chicago, USA.

Bouyssounouse, B. and J. Sifakis (2005). "Current design practice and needs in selected industrial sectors." Embedded System Design: Lecture Notes in Computer Science 3436: 15-38.

Bozheva, T. and M. E. Gallo (2010). Framework of Agile Patterns. Proceedings of the 5th Workshop "From code centric to model centric: Evaluating the effectiveness of MDD (C2M:EEMDD)". Paris, France.

Brandberg, G. (2005). "Less Fear, More Patterns (Interview with Linda Rising and Mary Lynn Manns)." Retrieved 7th December 2010, from www.cs.unca.edu/~manns/interview.pdf.

Bril, R. J., E. F. Steffens, et al. (2004). "Best-case response times and jitter analysis of real-time tasks." Journal of Scheduling 7: 133-147.

Brown, W., R. Malveau, et al. (1998). "The software patterns criteria: Proposed definitions for evaluating software pattern quality." Retrieved July 19th, 2011, from http://www.antipatterns.com/orig/whatisapattern/.

Buschmann, F., R. Meunier, et al. (1996). Pattern oriented software architecture. Chichester, UK, John Wiley.

Buttazzo, C. G. (1997). Hard real-time computing systems predictable scheduling algorithms and applications, Kluwer Academic Publishers.

Buttazzo, G. C. (2005). "Rate Monotonic vs. EDF : Judgement day." Real-time Systems 29(1): 5-26.

Cain, B. G., J. O. Coplien, et al. (1996). "Social patterns in productive software development organisations." Annals of Software Engineering 2(1): 259-286.

Carver, J., L. Jaccheri, et al. (2003). Issues in using students in empirical studies in software engineering education. Proceedings of the 9th International Symposium on Software Metrics 239-249.

Case, J. M. and G. Light (2011). "Emerging methodologies in engineering education research." Journal of Engineering education 100(1): 186-210.

Cheng, A. M. (2002). Real-time systems, scheduling analysis and verifications, John Wiley a& Sons.

Chikofsky, E. and J. Cross (1990). "Reverse engineering and design recovery: A taxonomy." IEEE Software 7(1): 13-18.

Chung, E. S., J. I. Hong, et al. (2004). Development and evaluation of emerging design patterns for ubiquitous computing. Proceedings of the 5th Conference on designing interactive systems: processes, practices, methods and techniques, ACM, New York.

Claesson, v. and N. Suri (2004). TT$^{ET:}$ Event-triggered channels on a time-triggered base. Proceedings of the 9th IEEE International Conference on Engineering of Complex Computer Systems 39-46.

Cline, M. (1996). "The pros and cons of adopting and applying design patterns in the real world." Communications of the ACM 39(10): 47-49.

Cloutier, R. J. and D. Verma (2007). "Applying the concept of patterns to systems architecture." Journal of Systems Engineering 10(2): 138-154.

Cool, W. (1998). Personal Communication, Hillside Inc. Meeting. Vancouver, Canada.

Coplien, J. O. (2000). Software patterns, Copyright 1996 AT&T, Copyright 2000, Lucent Technologies, Bell Labs Innovations

Coplien, J. O. (2007). ""A pattern definition: Software patterns"." Retrieved December 2011, from http://hillside.net/component/content/article/50-patterns/222-design-pattern-definition.

Cottet, F. and L. David (1999). A solution to the time jitter removal in deadline based scheduling of real-time applications. Proceedings of the 5th IEEE Real-time Technology and Applications Symposium. Vancouver, Canada.

Cottet, F., J. Delacroix, et al. (2002). Scheduling in real-time systems, John Wiley & Sons.

Cunningham, W. (1987). The CHECKS pattern language of information integrity, Addison Wesley.

Darlington, K. (2000). The Essence of Expert Systems. Essex, England., Prentice Hall.

Dechering, P., E. Groenboom, et al. (1999). Formalization of a software architecture for embedded systems: A process algebra for SPLICE. Proceedings of the 32nd Annual Hawaii International Conference on System Sciences.

DeLano, D. E. (1998). Section title: Patterns Mining. The patterns handbook: Techniques, strategies, and applications. L. Rising, Cambridge University Press.

Dillman, D. A. (2007). Mail and Internet Surveys: The tailored design method, John Wiley and Sons.

Dornyei, Z. and T. Taguchi (2009). Questionnaires In Second Language Research: Construction, Administration, And Processing, Taylor & Francis.

Eakin, E. (2003). Architecture's irascible reformer. The New York Times.

Eckert, C., P. J. Clarkson, et al. (2004). "Change and customisation in complex engineering domains." Research in Engineering Design 15(1): 1-21.

Eden, A. H. (2000). Precise specification of design patterns and tool support in their application. Tel Aviv University.

Eloranta, V. P., J. Koski, et al. (2009). Software architecture patterns for distributed embedded control systems. 14th European Conference on Pattern Languages of Programming, EuroPLoP 2009. A. Kelly and M. Weiss. Irsee, Germany, CEUR Workshop Proceedings. 566.

Eloranta, V. P., J. Koski, et al. (2010). A pattern language for distributed machine control system, Tampere University of Technology, Department of Software Systems. Report 9.

Fricke, A. and M. Volter (2000). Seminars: A pedagogical pattern language about teaching seminars effectively. Proceedings of the 5th European Conference on Pattern Languages of Programs.

Gabriel, R. (1996). Patterns of software, Oxford University Press.

Gamma, E., R. Helm, et al. (1995). Design patterns: Elements of reusable object-oriented software, Addison Wesley.

Ganssle, J. (1992). The art of programming embedded systems. San Diego, Academia Press.

Gardner, K. M., A. Rush, et al. (1998). Cognitive patterns, Cambridge University Press.

Gear, C. W. (1973). Introduction to Computer Science. Chicago, SRA.

Gendy, A. K. and M. J. Pont (2007). Towards a generic "Single Path Programming" solution with reduced power consumption. Proceedings of the International Design Engineering Technical Conferences & Computers and Information in Engineering Conference IDETC/CIE 2007. Las Vegas, Nevada, USA**:** 65-71.

Gendy, A. K. and M. J. Pont (2008a). Automating the process of selecting an appropriate scheduling algorithm and configuring the scheduler implementation for time-triggered embedded systems. Computer Safety, Reliability and Security, Lecture Notes in Computer Science. M. Harrison. Newcastle upon Tyne, UK, Springer Berlin / Heidelberg.

Gendy, A. K. and M. J. Pont (2008b). "Automatically configuring time-triggered schedulers for use with resource-constrained, single-processor embedded systems." IEEE Transactions on Industrial Informatics 4(1): 37-46.

Gergeleit, M. and E. Nett (2002). Scheduling transient overload with the TAFT scheduler. Fall meeting of GI/ITG specialized group of operating systems.

Gower, B. (1997). Scientific Method: An historical and philosophical introduction. Chesham, Buckinghamshire, Ponting-Green Publishing Services.

Graaf, B., M. Lormans, et al. (2003). "Embedded software engineering: The state of the practice." IEEE Software 20(6): 61-69.

Hanmer, R. (2003). Pattern Languages. Naperville, Illinois, USA., Lucent Technologies.

Hanmer, R. (2007). Patterns for fault-tolerant software, John Wiley & Sons, Ltd.

Hanmer, R., Ed. (2009). Software patterns for reusable design.

Harrison, N. B. (2006). The language of shepherding: A pattern language for shepherds and sheep. Pattern Languages of Program Design 5. Boston, Addison-Wesley.

Hebert, D. (2007). "Best practices in control system migration." Retrieved 21st July 2011, from http://www.controlglobal.com/articles/2007/006.html.

Herzner, W., W. Kubinger, et al. (2005). "Triple-T (Time-Triggered Transmission)" A system of patterns for reliable communication in hard real-time systems. Proceedings of the 5th European Conference on Pattern Languages of Programming EuroPLoP 2005. Irsee, Germany.

Hevner, A., S. March, et al. (2004). "Design science in information systems research." MIS Quarterly 28(5): 75-105.

Hughes, Z. M. and M. J. Pont (2004). Design and test of a task guardian for use in TTCS embedded systems. Proceedings of the 1st UK Embedded Forum, Published by University of Newcastle**:** 16-25.

Hughes, Z. M. and M. J. Pont (2008). "Reducing the impact of task overruns in resource-constrained embedded systems in which a time-triggered software architecture is employed." Trans Institute of Measurement and Control 30(5): 427-450.

IEC. (2012). "International Electrotechnical Commission, Functional Safety and IEC 61508." Retrieved August 2012, from http://www.iec.ch/functionalsafety/.

Jeffay, K., D. Stanat, et al. (1991). On non pre-emptive scheduling of periodic and sporadic tasks. 12th IEEE Symposium of Real-time Systems. San Antonio, Texas, IEEE Computer Society Press**:** 129-139.

Jerri, A. J. (1977). "The shannon sampling theorem: Its various extensions and applications A tutorial review." Proceedings of the IEEE 65(11): 1565-1596.

Jezequel, J.-M., M. Train, et al. (2000). Design Patterns and Contracts, Addison-Wesley.

Juristo, N. and A. M. Moreno (2001). Basics of Software Engineering Experimentation, Kluwer Academic Publishers.

Kalinsky, D. (2001). Context Switch. Embedded Systems Programming.

Kalinsky, D. (2002). Design patterns for high availability. Whitepaper/Advanced Course Reading Assignment, D. Kalinsky Associates.

Kawamura, Y., T. Yamazaki, et al. (2008). Network processing on an SPE core in cell broadband engineTM. Proceedings of the 16th IEEE Symposium on High Performance Interconnects**:** 119-128.

Khazanchi, D. (1996). A philosophical framework for the validation of information systems concepts. Proceedings of the Americas Conference on Information Systems. Phoenix, AZ**:** 755-757.

Khazanchi, D., J. D. Murphy, et al. (2008). Guidelines for evaluating patterns in the IS domain. Proceedings of the MWAIS 2008

Khazanchi, D. and I. Zigrus (2007). A systematic method for discovering effective patterns of virtual project management.

Kim, D., R. France, et al. (2002). Using Role-Based Modeling Language (RBML) to characterize model families. Proceedings of the 8th International Conference on Engineering of Complex Computer Systems, IEEE Computer Society**:** 107-116.

Kirner, R. and P. Puschner (2003). Discussion of misconceptions about WCET analysis. Proceedings of the 3rd Euromicro International workshop on WCET Analysis**:** 61-64.

Kitchenham, B. A., S. L. Pfleeger, et al. (2002). "Preliminary guidelines for empirical research in software engineering." IEEE Transactions on Software Engineering 28(8): 721-734.

Kopetz, H. (1991). Event-triggered versus time-triggered real-time systems. Proceedings of the Workshop on Operating Systems of the 90s and Beyond 87-101.

Kopetz, H. (1993). "Should responsive systems be event-triggered or time-triggered?" IEICE Transactions on Information and Systems E-76-D(11): 1325-1332.

Kopetz, H. (1997). Real-time systems design principles for distributed embedded applications, Kluwer Academic Publishers.

Kopetz, H. and G. Bauer (2002). The time-triggered architecture. IEEE Special Issue on Modelling and Design of Embedded Software 112-126.

Krasner, J. (2010). "Critical issues confronting medical device manufacturers, their CEOs, CFOs, managers and developers, EMF guide for medical device developers." Retrieved November, 2011, from http://www.researchandmarkets.com/reports/1226357/critical_issues_confronting_medical_device.

Krippendorff, K. (2004). Content Analysis An Introduction to Its Methodology, Sage Publicatioons.

Kubinger, W. and M. Humenberger (2004). Concept of time-triggered image acquisition for embedded control applications. Proceedings of the 10th IEEE Real-Time and Embedded Technology and Applications Symposium, Toronto, Canada.

Kumar, R. (2005). Research Methodology: A step-by-step guide for beginners, Sage Publications Ltd.

Kurian, S. and M. J. Pont (2005). Building reliable embedded systems using abstract patterns, patterns, and pattern implementation examples. Proceedings of the 2nd UK Embedded Forum, Birmingham, UK, Published by University of Newcastle upon Tyne 36-59.

Kurian, S. and M. J. Pont (2006). Restructuring a pattern language which supports time-triggered co-operative software architectures in resource-constrained embedded systems. Proceedings of the 11th European Conference on Pattern Languages of Programs (EuroPLoP 2006). Germany.

Kurian, S. and M. J. Pont (2007). "Maintenance and evolution of resource constrained embedded systems created using design patterns." Journal of Systems and Software 80(1): 32-41.

Labrosse, J. (1999). MicroC/OS-II: The real-time kernel, CMP Books.

Labrosse, J. (2000). Embedded systems building blocks, CMP Books.

Lakhani, F., A. Das, et al. (2009a). Towards a pattern language which supports migration of system from event-triggered pre-emptive to time-triggered co-operative software architectures. Proceedings of the 14th European Conference on Pattern Languages of Programs (EuroPLoP 2009), Irsee, Germany, Published by CEUR F2-1 to F2-25.

Lakhani, F., A. Das, et al. (2009b). Can we support migration from event-triggered to time-triggered architectures using design patterns? Proceedings of the 5th UK Embedded Forum. Leicester, UK, Published by University of Newcastle upon Tyne**:** 62-73.

Lakhani, F., A. Das, et al. (2010c). Improving the reliability of embedded systems as complexity increases: supporting the migration between event-triggered and time-triggered software architectures. Proceedings of the 15th European Conference on Pattern Languages of Programming (EuroPLoP'10). Irsee, Germany, ACM Press**:** 22:21 - 22:17.

Lakhani, F. and M. J. Pont (2010a). Using design patterns to support migration between different system architectures. Proceedings of the 5th IEEE International Conference on Systems of Systems Engineering (SoSE). Loughborough, UK**:** 1-6.

Lakhani, F. and M. J. Pont (2010b). "Code balancing" as a philosophy for change: Helping developers to migrate from event-triggered to time-triggered architectures. Proceedings of the first UK Electronic Forum. New Castle, UK, Published by Newcastle University.

Lakhani, F. and M. J. Pont (2012a). Applying design patterns to improve the reliability of embedded systems through a process of architecture migration. Proceedings of the 9th IEEE International Conference on Embedded Systems and Software (ICESS 2012). Liverpool, UK., IEEE Computer Society**:** 1563 - 1570.

Lakhani, F. and M. J. Pont (2012b). "Empirical studies for the assessment of the effectiveness of design patterns in migration between software architectures of embedded applications." ISRN, Journal of Software Engineering.

Lakhani, F., H. Wang, et al. (2011). Supporting the migration between 'event-triggered' and 'time-triggered' software architectures: A small pattern collection intended for use by the developers of reliable embedded systems. Technical Report ESRG 2011-09-01, University of Leicester.

Leung, J. Y. T. and J. Whitehead (1982). "On the complexity of fixed-priority scheduling of periodic, real-time tasks." Performance Evaluation 2(4): 237-250.

Li, Q. and C. Yao (2003). Real-time concepts for embedded systems, CMP Books.

Liu, C. L. and J. W. Layland (1973). "Scheduling algorithms for multiprogramming in a hard real-time environment." Journal of the ACM 20(1): 46-61.

Liu, J. (2000). Real-time systems, Prentice Hall.

Locke, D. (1992). "Software architectures for hard real-time applications: Cyclic executives vs. fixed priority executives." Real-time Systems 4(1): 37-53.

Maaita, A. (2008). Techniques for enhancing the temporal predictability of real-time embedded systems employing a time-triggered software architecture. PhD Thesis, University of Leicester.

Maaita, A. and M. J. Pont (2005). Using 'Planned Pre-emption' to reduce levels of task jitter in a time-triggered hybrid scheduler. Proceedings of the 2nd UK Embedded Forum. Birmingham, UK: 18-35.

Madisetti, V. J., Y. K. Jung, et al. (1999). "Re-engineering legacy embedded systems." IEEE Design and Test of Computers 16(2): 38-47.

Mak, J. K. H., S. T. Choy, et al. (2003). Precise specification to compound patterns with ExLePUS. Proceedings of the 27th Annual International Conference on Computer Software and Applications 440-445.

Manns, M. L. and L. Rising (2004). Fearless change: Patterns for introducing new ideas, Addison Wesley.

Mapelsden, D., J. Hosking, et al. (2002). Design pattern modeling and instantiation using DPML. Proceedings of the 40th International Conference on Tools Pacific: Objects for internet, mobile and embedded applications. Sydney Australia, Australian Computer Society, Inc.

Marti, P. (2002). Analysis and design of real-time control systems with varying control timing constraints, Technical University of Catalonia.

Martin, T. (2005). The insider's guide to the Philips ARM7 based microcontrollers, Coventry, Hitex, UK, Ltd.

Matassa, L. M. (2011). Power PC to Intel architecture migration, Intel Corporation.

Mikkonen, T. (1998). Formalizing design patterns. Proceedings of the 20th International conference on Software engineering. Kyoto, Japan, IEEE Computer Society.

Millett, S. (2010). Professional ASP.Net Design Patterns, Wiley Publishing Inc.

Mosley, D. (2006). When to migrate legacy embedded applications. Proceedings of the International Conference on Ada, SIGAda'06, ACM. 26.

Mwelwa, C. (2006). Development and assessment of a tool to support pattern-based code-generation of time-triggered embedded systems. PhD Thesis, University of Leicester.

Mwelwa, C., M. J. Pont, et al. (2006). Rapid software development for reliable embedded systems using a pattern-based code generation tool. Society of Automotive Engineers (SAE) World Congress. Detroit, Michigan, USA.

Neil, T. (2012). Mobile Design Pattern Gallery, O'Reilly Media Inc.

NI. (2012). "National Instruments." Retrieved March 2012, from http://www.ni.com/labview/release-archive/2010/.

Nissanke, N. (1997). Real-time systems, Prentice Hall.

Obermaisser, R. (2005). Event-triggered and time-triggered control paradigms, Springer.

OED. (2012). "Oxford English Dictionary." Retrieved April 2012, from http://www.oed.com.

Oest, O. N. (2008). "Migrating complex embedded systems." Military Embedded Systems Retrieved 30th July 2011, from http://www.mil-embedded.com/articles/id/?3375.

OLIMEX2378. (2012). Retrieved May 2012, from http://www.olimex.com/dev/lpc-2378stk.html.

Ortega-Arjona, J. L. (2009). Architectural patterns for parallel programming: Models for performance estimation, VDM Verlag.

Ortega-Arjona, J. L. (2010). Patterns for parallel software design, John Wiley and Sons.

Park, J., M. Ryu, et al. (2006). "Rapid performance re-engineering of distributed embedded systems via latency analysis and k-level diagonal search." Journal of Parallel and Distributed Computing 66: 19-31.

Peterson, R. A. (2000). Constructing Effective Questionnaires, Sage Publications Inc.

Petter, S., D. Khazanchi, et al. (2010). "A design science based evaluation framework for patterns." ACM SIGMIS Database 41(3).

Phatrapornnant, T. and M. J. Pont (2006). "Reducing jitter in embedded systems employing a time-triggered software architecture and dynamic voltage scaling." IEEE Transactions on Computers 55(2): 113-124.

Pont, M. J. (2001). Patterns for Time-Triggered Embedded Systems, ACM press.

Pont, M. J. (2002). Embedded C, Addison-Wesley.

Pont, M. J., S. Kurian, et al. (2008). Selecting an appropriate scheduler for use with time-triggered embedded systems. Proceedings of the 12th European Conference on Pattern Languages of Programs, Irsee, Germany, Universitätsverlag Konstanz.

Pont, M. J., Y. Li, et al. (1998). The design of embedded systems using software patterns. Proceedings of Condition Monitoring. Swansea, UK**:** 221-236.

Prechelt, L. and B. Unger (1998). A series of controlled experiments on design patterns: Methodology and results. Proceedings of Softwaretechnik '98 (Softwaretechnik-Trends) 53-60.

Prechelt, L., B. Unger, et al. (2002). "Two controlled experiments assessing the usefulness of design pattern documentation in program maintenance." IEEE Transactions on Software Engineering 28(6).

Prechelt, L., B. Unger, et al. (2001). "A controlled experiment in maintenance, comparing design patterns to simpler solutions." IEEE Transactions on Software Engineering 27(12): 1134-1144.

Puschner, P. (2002). Is WCET analysis a non-problem? Towards new software and hardware architectures. 2nd International Workshop on Worst Case Execution Time Analysis, Vienna, Austria.

Puschner, P. (2003). The single-path approach towards WCET-analysable software. IEEE International Conference on Industrial Technology 699-704.

Puschner, P. and A. Burns (2002). Writing Temporally Predictable Code. Proceedings of the 7th International Workshop on Object-Oriented Real-time Dependable Systems**:** 85-91.

Ramamritham, K. and J. Stankovic (1994). Scheduling algorithms and operating systems support for real-time systems. Proceedings of the IEEE 55-67.

RapidiTTy. (2009). "RapidiTTY tool set for Rapid development of embedded systems." Retrieved June 2012, from http://www.tte-systems.com/products.

Rising, L., Ed. (1998). The patterns handbook: Techniques, strategies and applications, Cambridge University Press.

Rising, L. (1999). Patterns: A way to reuse expertise. IEEE Communications Magazine.

Rising, L., Ed. (2001). Design patterns in communication software, Cambridge University Press.

Robson, C. (2002). Real world research: A resource for social scientist and practitioner, Blackwell Publishers.

Scarlett, J. J. and R. W. Brennan (2006). Re-evaluating event-triggered and time-triggered systems. Proceedings of the IEEE Conference on Emerging Technologies and Factory Automation**:** 655-661.

Scheler, F., M. Mitzlaff, et al. (2007). Towards a Real-time Systems Compiler. Proceedings of the 5th International Workshop on Intelligent Solutions in Embedded Systems (WISES 07)**:** 62-75.

Scheler, F. and W. Schroder-Preikschat (2006). Time-triggered versus Event-triggered: A matter of Configuration? Proceedings of the MMB GI/ITG Workshop on Non-Functional Properties of Embedded Systems, Nuremberg, Berlin VDE Verlag.

Scheler, F. and W. Schroder-Preikschat (2010). The RTSC: Leveraging the migration from event-triggered to time-triggered systems. Proceedings of the13th IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing**:** 34-41.

Schlindwein, F. S., M. J. Smith, et al. (1988). "Spectral analysis of doppler signals and computations of the normalized first moment in real-time using a digital signal processor." Medical and Biological Engineering & Computing 26: 228-232.

Schmidt, D. (1995). "Using design patterns to develop reusable object-oriented communication software." Communication ACM 38(10): 65-74.

Schmidt, D., M. Stal, et al. (2000). Pattern-oriented software architecture Volume 2: Patterns for concurrent and networked objects, John Wiley and Sons.

Seaman, C. B. (1999). "Qualitative methods in empirical studies of software engineering." IEEE Transactions on Software Engineering 25(4): 557-572.

Sha, L. (2004). "Real-time scheduling theory: A historical perspective." Real-time Systems 28(2): 101-155.

Sha, L., R. Rajkumar, et al. (1990). "Priority inheritance protocols: An approach to real-time synchronization " IEEE Transactions on Computers 39(9): 1175-1185.

Shaw, A. C. (2001). Real-time systems and software, John Wiley & Sons Inc.

Shepard, T. and M. Gagne (1990). A model of F18 Mission Computer software for pre run-time scheduling. Proceeding of the 10th International Conference on Distributed Computing Systems. Paris, France**:** 62-69.

Short, M. and M. J. Pont (2005). 'Hardware in the Loop' simulation of embedded automotive control systems. Proceedings of the IEEE International Conference on Intelligent Transportation Systems (IEEEITSC 2005)**:** 426-431.

Short, M., M. J. Pont, et al. (2008). "Assessment of performance and dependability in embedded control systems: Methodology and case study." Control Engineering Practice 16: 1293-1307.

Short, M., M. J. Pont, et al. (2008). Exploring the impact of task pre-emption on dependability in time-triggered embedded systems: A pilot study. Proceedings of the Real-time Systems - Euromicro Conference on, ECRTS, 2008, IEEE**:** 83-91.

Silverman, D. (2011). Interpreting Qualitative Data, Sage Publications.

Sloss, A. N., D. Symes, et al. (2004). ARM system developer's guide: Designing and optimizing system software, Morgan Kaufmann.

Sproull, N. L. (1995). Handbook of research methods: A guide for practitioners and students in the social sciences, (2nd edition), The Scarecrow Press, Inc.

Straub, D. W. (1989). Validating instruments in MIS research. MIS Quarterly. 13**:** 146-169.

Strebelow, R. and C. Prehofer (2011). Work in Progress: Evaluation of parallel design patterns for message processing systems on embedded multi-core systems. Proceedings of the 1st workshop on Systems for Future Multi-core Architectures EuroSys 2011, Slazburg, Austria.

Taibi, T. (2007). Design Pattern formalization techniques, IGI Publishing.

Taibi, T. and D. C. Ling Ngo (2003). "Formal specification of design patterns - A balanced approach." Journal of Object Technology 2(4): 127-140.

Taibi, T. and F. Taibi (2006). Formal specification of design patterns and their instances. Proceedings of the IEEE International Conference on Computer Systems and Applications, IEEE Computer Society: 33-36.

Tindell, K. W., A. Burns, et al. (1994). "An extendible approach for analyzing fixed priority hard real-time tasks." Real-Time Systems 6: 133-151.

TTE. (2012). "TTE Systems Ltd." Retrieved April 2012, from http://www.tte-systems.com/technology.

Turley, J. (2003) "Motoring with microprocessors." EE Times.

Turley, J. (2009). Gaming the system -- High-end networking on the cell processor. Embedded.com.

UM10211 (2009). LPC 23XX User manual Rev. 03.

Unger, B. and W. F. Tichy (2000). Do design patterns improve communication? An experiment with pair design. Proceedings of the International Workshop on Empirical Studies of Software Maintenance.

Vahid, F. and T. Givargis (2002). Embedded System Design: A Unified Hardware/Software Introduction, John Wiley & Sons, Inc.

Vlissides, J. (1996). Seven Habits of Successful Pattern Writers, C++ Report, November/December 1996.

Vlissides, J. (1997). "Patterns: The top ten misconceptions." Retrieved 22nd June 2012.

Vokac, M. (2004). On the practical use of software design patterns. PhD Thesis, University of Oslo, Norway.

Wang, H., M. J. Pont, et al. (2007). Patterns which help to avoid conflicts over shared resources in time-triggered embedded systems which employ a pre-emptive scheduler. Proceedings of the 12th European Conference on Pattern Languages of Programs (EuroPLoP 2007). Irsee, Germany.

Ward, N. J. (1991). "The static analysis of safety-critical avionics control system" in air transport safety. Proceedings of the Safety and Reliability Spring Conference.

Xu, J. (2003a). "On inspection and verification of software with timing requirements." IEEE Transactions on Software Engineering 29(8): 705-720.

Xu, J. (2003b). "Making software timing properties easier to inspect and verify." IEEE Software 20(4): 34-41.

Xu, J. and D. Parnas (1993). "On satisfying timing constraints in hard real-time systems." IEEE Transactions on Software Engineering 19(1): 70-84.

Xu, J. and D. Parnas (2000). "Priority scheduling versus pre run-time scheduling." The International Journal of Time-Critical Computing Systems 18(1): 7-23.