Elektrotechnik & Informationstechnik (2008) 125/11: 401–405. DOI 10.1007/s00502-008-0587-z

Applying time-triggered architectures in reliable embedded systems: challenges and solutions

M. J. Pont

Since the mid 1990s, researchers in the Embedded Systems Laboratory at the University of Leicester have developed a range of techniques and tools which support the creation and maintenance of reliable, resource-constrained embedded systems. Central to this work has been a focus on systems with a time-triggered architecture. This paper provides a review of some of this work.

Keywords: time-triggered architectures; reliable embedded systems

Anwendung von zeitgesteuerten Architekturen in zuverlässigen eingebetteten Systemen: Herausforderungen und Lösungen.

Seit Mitte der 1990er Jahre haben Forscher im Embedded Systems Laboratory der Universität von Leicester eine Reihe von Techniken und Werkzeugen entwickelt, die den Entwurf und die Wartung von zuverlässigen eingebetteten Systemen unter gleichzeitiger Berücksichtigung von limitierten Systemressourcen unterstützen, wobei der zentrale Fokus auf zeitgesteuerten Architekturen liegt. Der vorliegende Beitrag zeigt in einer Zusammenfassung wesentliche Aspekte dieser Arbeit.

Schlüsselwörter: zeitgesteuerte Architekturen; zuverlässige eingebettete Systeme

Received July 28, 2008, accepted August 6, 2008

1. Introduction

For the last 15 years, researchers in the Embedded Systems Laboratory at the University of Leicester (UK) have been carrying out research in the area of time-triggered (TT) embedded systems. This paper reviews some of the work which we have carried out in this area.

We begin by explaining what we mean by a TT design.

2. Time-triggered software architectures

Throughout this paper, we are concerned with the development of embedded systems for which there are two (sometimes conflicting) constraints. First, we wish to implement the design using a low-cost microcontroller, which has – compared to a desktop computer – very limited memory and CPU performance. Second, we wish to produce a system with extremely predictable timing behaviour: for example, we typically require that the interval between the start times (the ''jitter'') of key periodic tasks does not vary by more than l μs.

In order to minimise costs and maximise predictability in this type of application, we wish to keep the software architecture as simple as possible. Therefore, instead of a full RTOS, a simplified form of a scheduler is often used. For example, a cyclic executive (*Baker, Shaw, 1989; Locke, 1992*) is a form of cooperative (or ''non pre-emptive'') scheduler that has a ''time-triggered'' – as opposed to event-triggered – architecture. If 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 – provided that an appropriate algorithm is used – can maintain their low-jitter characteristics even when techniques such as dynamic voltage scaling (DVS) are employed to reduce system power consumption (*Phatrapornnant, Pont, 2006*).

Despite many attractive features, a TTC solution is not always appropriate; as *Allworth (1981)* has noted, ''*[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*''. We can express this concern slightly more formally by noting that if a system is being designed which must execute one or more tasks of (worst-case) execution time e and also respond within an interval t to external events then, in situations where $t < (e +$ execution time of the task that handles the event), a pure co-operative scheduler will not generally be suitable.

In such circumstances, it is tempting to opt immediately for a fully pre-emptive design with event-triggered tasks. However, there are other design options available, including – for example – ''TT hybrid'' and ''TT rate monotonic'' designs. In the case of a TT hybrid (TTH) design (*Pont, 2001*), the system designer creates a static schedule made up of (i) a collection of tasks, which operate co-operatively and (ii) a single – short – pre-empting task. In many designs, the pre-empting task will be used for periodic data acquisition, typically through an analogue-to-digital converter or similar device: such a requirement is common in, for example, a wide range of control systems (*Buttazzo, 2005*). In the case of a ''TT rate monotonic'' (TTRM) design (*Wang, Pont, 2008*), we implement a rate-monotonic architecture (*Liu, Layland, 1973*) – using a static schedule – and can still obtain very predictable system behaviour with limited resource requirements.

Pont, Michael J., Dr., Embedded Systems Laboratory, University of Leicester, University Road, Leicester LE1 7RH, UK (E-mail: M.Pont@le.ac.uk)

Overall, our aim with a TT design can be summarised as follows: we wish to be able to determine – before the system begins executing – exactly what it will do at every moment of during which it is running. Such predictable behaviour offers clear advantages for many types of system, but it comes at a price. Throughout this paper, we will consider ways in which some of the challenges facing the developers of TT systems can be addressed.

3. The fragility of TT (design time)?

One key challenge with any form of TT architecture is what can be called the ''fragility'' of the design. By this we mean that changes to even a single task in a working design (for example, a slight increase in execution time) may require significant revisions to the schedule parameters.

In fact, this is just one aspect of a larger problem. If (for example) we implement a TTC or TTH design, a number of key scheduler parameters must be determined at design time (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). Where the parameter set does ensure that all tasks are scheduled, inappropriate decisions may still lead to unnecessarily high levels of task jitter and/or to increased system power consumption. It has been demonstrated in previous studies that the problem of determining these parameters is NP-hard, *Brucker, Garey, Johnson, 1977*; *Baker, Shaw, 1989*; *Tindell, Burns, Wellings, 1992*; *Xu, Parnas, 1992*; *Kopetz, 1997*; *Ekelin, Jonsson, 2001*; *Cucu, Sorel, 2004*; *Baruah, 2006*).

Effective solutions now exist for such problems. For example, we have recently described a solution which ensures that: (i) task constraints are met; (ii) power consumption is ''as low as possible''; (iii) a fully cooperative scheduler architecture is employed whenever possible (*Gendy, Pont, 2008a, b*). This approach does not attempt to perform an exhaustive search, but it provides results which are close to those obtained in a branch and bounds search, in a fraction of time (typically a few seconds).

4. The fragility of TT (run time)?

In Sect. 3, we considered the fragility of TT designs at design time. To ensure reliable behaviour, we also need to take steps to ensure that – if our design assumptions are not met at run time – the system behaviour is not compromised. In this context, information about Worst Case Execution Time (WCET) is of key concern. In any TT design, the WCET of every task must be known at design time. Unfortunately, as many researchers have observed (e.g. *Liu, Layland, 1973*; *Becker et al., 2003*; *Becker, Gergeleit, 2001*; *Domaratsky, Perevozchikov, 2000*; *Engblom et al., 2001*; *Gergeleit, Nett, 2002*; *Kirner, Puschner, 2003*; *Nett et al., 1996*; *Puschner, 2002*), determining the WCET of tasks is rarely straightforward.

Lack of knowledge about WCETs is a problem which faces the developers of many embedded systems (not just those based on TT designs). For example, 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''. Nonetheless, the fact that a TT architectures employ static scheduling means that a task overrun may have a serious impact on the system behaviour. For example, as Buttazzo has noted: ''[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'' (*Buttazzo, 2005*).

One simple solution to this problem is to err on the side of caution when employing WCET estimates, thereby reducing the chances that an overrun will occur. Typical ''safety margins'' used in this way are said to be around 20% (*Vallerio, Jha, 2003*). Such an approach is simple and can be effective, but it inevitably adds to costs.

In many cases, a better alternative is to be slightly more conservative when estimating WCET values (e.g. add 5% to accurate estimates) and then extend the scheduler (or add additional hardware) in such a way that (at run time) any overrunning tasks can be shut down, and/or the schedule can be adjusted. This can be done by employing some form of ''watchdog timer'' (e.g. *Ganssle, 2002*) in a ''scheduler watchdog'' design (e.g. *Pont, Ong, 2002*). Alternatively, greater control over the system behaviour can be obtained by using a ''task guardian'' (*Hughes, Pont, 2004*; *Hughes, Pont, in press*).

5. Working with multi-processor designs

Time-triggered architectures can be applied very effectively in both single- and multi-processor designs. In the case of multi-processor designs, we have sought to demonstrate that a ''Shared-Clock'' (S-C) architecture provides a simple, flexible platform for many systems (*Pont, 2001; Ayavoo et al., 2007; Imran, Short, Pont, 2008; Muhammad, Pont, 2008*).

Our work in this area has – to date – had a focus on systems which employ the Controller Area Network (CAN) protocol at the physical layer (*Bosch, 1991*). CAN provides high-reliability communications at low cost, and most modern microcontroller families now have members with on-chip support for this protocol.

All of the S-C algorithms describe time division, multiple access (TDMA) protocols. The key idea behind these protocols is to synchronise the execution of tasks on the individual nodes by sharing a single clock source between the various processor boards (see Fig. 1).

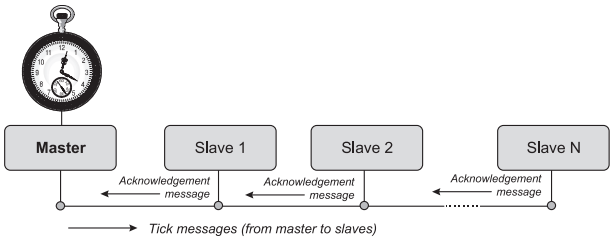

Fig. 1. Communication between Master and Slaves nodes in S-C architectures

In Fig. 1, we have a clock on the Master node that generates periodic timer interrupts (to drive the scheduler on this node). When a timer interrupt occurs, the Master node also generates a Tick message that is sent to the Slave nodes using the CAN protocol. The Slave nodes react to the Tick message by generating an interrupt (from the CAN hardware). This interrupt will in turn be used to drive the Slave's scheduler.

Besides synchronising the individual nodes on the network, TTC-SC1 (the first S-C protocol) is also responsible for detecting network and node failures. TTC-SC1 does this by having the Slave nodes return an ''Acknowledgement'' (Ack) message back to the Master node (Fig. 1). This way, the Master node will know the status of all its Slaves after one TDMA round. With each Tick message, the Master node identifies which Slave should return an Ack message by embedding that particular Slave ID in the data stream. Only the Slave with this particular ID will send an Ack message back to the Master. The Master will then check the status of this Slave, and send the next Tick message out with a new Slave ID.

CAN messages may be up to 8 bytes long. Only a limited overhead (typically up to 1 byte in each message) is required to support the TTC-SC1 protocol. This leaves up to 7 bytes/message for data transfers between the Master and Slave nodes.

Please note that in this version of the protocol (TTC-SC1), Slave-to-Slave communication is not permitted: all communication is direc-

ted via the Master node (through Tick and Ack messages). Some variations on this theme have also been reported (*Ayavoo et al., 2007*).

In a related programme of work, we have explored ways in which we can – to a large extent – automate the process of converting single-processor designs into an equivalent multi-processor (shared-clock) design (*Vidler, Pont, 2006*): this work is ongoing.

6. Specialised TT hardware

The techniques described in previous sections of this paper have involved creating software for industry-standard hardware platforms, such as the 8051 microcontroller (*Pont, 2001*), ARM processor (*Pont, Mwelwa, 2003*) or PC platform (*Pont et al., 2004*). Developing reliable applications using this approach can be effective, but there is a mismatch between generic processor architectures and time-triggered software designs. For example, most processors support a wide range of interrupts, while the use of a time-triggered software architecture generally requires that only a single interrupt is active on each processor. This leads to design ''guidelines'', such as the ''one interrupt per microcontroller rule'' (*Pont, 2001*).

As an alternative to such design guideliness, we have begun to explore ways of creating different forms of ''TT processor'' (*Hughes, Pont, Ong, 2005; Athaide, Pont, Ayavoo, 2008a, b*). This ongoing work is exploring both single- and multi-core designs.

7. The design of TT systems using patterns

TT architectures match the needs of many embedded applications extremely well. However, there is a catch: designs using these architectures must be implemented with care. For example – as we noted earlier in relation to (pure) TTC designs – task durations must be brief if the real-time system performance is not to be impaired (e.g. *Allworth, 1981*). This concern is legitimate, and any TTC system which has been designed without due consideration of task durations is likely to prove extremely unreliable. However, there are a number of different techniques that may be employed in order to address this problem. For example: (i) By using a faster processor, or a faster system oscillator, we can reduce the duration of ''long'' tasks. (ii) By making use of an additional processor or CPU core, we can obtain a true multi-tasking capability. (iii) By using ''time out'' mechanisms, we can help to ensure that tasks complete within their allotted time. (iv) By splitting up long tasks (triggered infrequently) into shorter ''multi-stage'' tasks (triggered frequently), the processor activity can be more evenly distributed. In the right circumstances, each of these ideas can prove useful. However, such observations on their own do not make it very much easier for developers to employ time-triggered solutions. Instead, what is needed is a means of supporting the ''recycling of design experience'': specifically, we would like to find a way of allowing less experienced software engineers to incorporate features from previous (successful) designs in their systems.

In recent years, some developers have found that design patterns offer an effective means of achieving this form of ''design recycling''. Current work on patterns was inspired by Christopher Alexander and his colleagues (*Alexander, 1979*; *Alexander et al., 1977*). Alexander is an architect who first described what he called ''a pattern language'' relating various architectural problems (in buildings) to good design solutions. He defines patterns as ''*a three-part rule, that express a relation between a certain context, a problem and a solution*'' (*Alexander, 1979, p. 247*).

This concept of descriptive problem-solution mappings was adopted by *Ward Cunningham* and *Kent Beck (1987)* who used some of Alexander's techniques as the basis for a small pattern language intended to provide guidance to novice Smalltalk programmers. Cunningham and Beck's work was subsequently built upon by Erich Gamma and colleagues who, in 1995, published an influential book on general-purpose object-oriented software patterns (*Gamma et al., 1995*).

Over the last decade, the development of pattern-based design techniques has become an important area of research in the software engineering community. Gradually, the focus has shifted from the use, assessment and refinement of individual patterns, to the creation of complete pattern languages, in areas including telecommunications systems (*Rising, 2001*) and systems with hardware constraints (*Noble, Weir, 2001*).

In 1996, we began to assemble a collection of patterns to support the development of time-triggered software for embedded systems. The first versions of these patterns were used ''in house'', primarily for teaching and training purposes. We then began to publish and discuss the next versions of the patterns more widely (*Pont, 2000*) and not just at pattern workshops but also at more general technical conferences (see for example: *Pont, 1998*; *Pont, 1999*; *Pont et al., 1998*; *Pont et al., 1999*). Through this process, we obtained a great deal of useful feedback on the project, and refined the collection again. The end result was a set of more than seventy patterns, which we refer to as the Patterns for Time-Triggered Embedded Systems (PTTES) collection (*Pont, 2001, 2003*; *Pont, Banner, 2004*).

8. Ongoing work on TT patterns

We are continuing to do work in the area of pattern-based design. For example, while most of our previous work has explored TTC architectures, we are now beginning top explore patterns supporting TT systems with task pre-emption in greater detail (*Pont et al., 2007*; *Wang, Pont, Kurian, 2007*; *Wang, Pont, 2008*). In addition, we are exploring techniques which can be used to place greater control over the ways in which pattern-related code components are integrated (*Mearns, Pont, Ayavoo, 2008*)

Like most of our previous work, the studies mentioned above focus on code creation. It is frequently argued that the costs of software creation are dwarfed in many projects by the costs of software maintenance (e.g. see *Bennett, Gold, Mohan, 2005*). This has encouraged some researchers to consider ways in which design patterns might be applied during software maintenance (e.g. see *Keller et al., 1999*; *Cinneide, Nixon, 2001*; *Huang et al., 2005*). In this vein, we have begun to explore ways in which can be used to support the maintenance of embedded systems which were created using patterns. To date, our particular focus has been on techniques which will allow us to exchange patterns in such a project – with minimal or no human intervention – after the code has been created and the system commissioned. By exchanging at this time, our aim is to identify the implementation of the pattern which we wish to change in the system code. Having done so, we aim to remove the relevant code and then substitute a suitable implementation of the replacement pattern (*Kurian, Pont, 2007*).

9. Examples of TT designs

We have provided detailed examples of systems created using TT designs in a number of papers. For example, we have described the implementation of a low-power medical (ECG) monitoring system (*Phatrapornnant, Pont, 2006*), a cruise-control system for a passenger car (*Short, Pont, 2008*), a controller for a brushless DC motor (*Hanif et al., 2008*) and a robust control system using an H_∞ algorithm (*Bautista-Quintero, Pont, 2008*).

10. From theory into practice

In 2005, the University of Leicester began the process of creating a new company to exploit the expertise in TT technology within the Embedded Systems Laboratory (ESL). The resulting company

(TTE Systems Ltd) now has a growing list of customers from across the world.

11. Conclusions

For the last 15 years, researchers in the Embedded Systems Laboratory at the University of Leicester (UK) have been carrying out research in the area of time-triggered (TT) embedded systems. This paper has provided an overview of some of the work which we have carried out in this area.

References

Alexander, C. (1979): The timeless way of building. Oxford University Press, NY.

Alexander, C., Ishikawa, S., Silverstein, M., Jacobson, M., Fisksdahl-King, I., Angel, S. (1977): A pattern language. Oxford University Press, NY.

Allworth, S. T. (1981): An Introduction to real-time software design. Macmillan, London.

Athaide, K. F., Pont, M. J., Ayavoo, D. (2008a): Deploying a time-triggered shared-clock architecture in a multiprocessor system-on-chip design. In: Proceedings of the 4th UK Embedded Forum (September 2008, Southampton, UK).

Athaide, K. F., Pont, M. J., Ayavoo, D. (2008b): Shared-clock methodology for time-triggered multi-cores. In: Stepney, S., Polack, F., McEwan, A., Welch, P., Ifill, W. (eds): Communicating Process Architectures 2008. IOS Press.

Ayavoo, D., Pont, M. J., Short, M., Parker, S. (2007): Two novel shared-clock scheduling algorithms for use with CAN-based distributed systems. Microprocessors and Microsystems, 31 (5): 326–334.

Baker, T. P., Shaw, A. (1989): The cyclic executive model and Ada. Real-Time Systems 1 (1): 7–25.

Baruah, S. K. (2006): The non-preemptive scheduling of periodic tasks upon multiprocessors. Real-Time Systems 32 (1–2): 9–20.

Bautista-Quintero, R., Pont, M. J. (2008): Implementation of H-infinity control algorithms for sensor-constrained mechatronic systems using low-cost microcontrollers. IEEE Transactions on Industrial Informatics 4 (3): 175–184.

Becker, L. B., Gergeleit, M. (2001): Execution environment for dynamically scheduling real-time tasks. RTSS 2001, 22nd IEEE Real-Time Systems Symposium, London, 2001.

Becker, L. B., Nett, E., Schemmer, S., Gergeleit, M. (2003): Robust scheduling in team-robotics. 11th Int. Workshop on Parallel and Distributed Real-Time Systems, Nice, France, 2003.

Bennett, K., Gold, N., Mohan, A. (2005): Cut the biggest IT cost. The Computer Bulletin 47 (1): 20–21.

Bosch, R. G. (1991): CAN specification version 2.0: Robert Bosch GmbH, Postfach 50, D-7000 Stuttgart 1, Germany.

Brucker, P., Garey, M. R., Johnson, D. S. (1977): Scheduling equal-length tasks under treelike precedence constraints to minimize maximum lateness. Mathematics of Operations Research 2 (3): 275–284.

Buttazzo, G. C. (2005): Rate monotonic vs. EDF: Judgement day. Real-Time Systems 29 (1): 5–26.

Cinneide, M. O. and Nixon, P. (2001): Automated software evolution towards design patterns. International Workshop on Principles of Software Evolution (IWPSE), Association for Computing Machinery, Vienna, Austria: 162–165.

Cucu. L., Sorel, Y. (2004): Non-preemptive multiprocessor scheduling for strict periodic systems with precedence constraints. In Proc. 23rd Annual Workshop of the UK Planning and Scheduling Special Interest Group, PLANSIG'04, Cork, Ireland, Dec. 2004.

Cunningham, W., Beck, K. (1987): Using pattern languages for object-oriented programs. Proc. of OOPSLA'87, Orlando, Florida.

Domaratsky, Y., Perevozchikov, M. (2000): Highly dependable time-triggered operating system. Dedicated Systems Magazine, Oct.–Dec. 2000: 77–84.

Ekelin, C., Jonsson, J. (2001): Evaluation of search heuristics for embedded system scheduling problems. In: Proc. Int. Conf. Principles and Practice of Constraint Programming, Paphos, Cyprus, 2001: 640–654.

Engblom, J. A., Ermedahl, A., Sjoedin, M., Gubstafsson, J., Hansson, H., et al. (2001): Worst-case execution-time analysis for embedded real-time systems. Journal of Software Tools for Technology Transfer 4 (4): 437–455.

Gamma, E., Helm, R., Johnson, R., Vlissides, J. (1995): Design patterns: Elements of reusable object-oriented software. Addison-Wesley, Reading, MA.

Ganssle, J. (2002): The Art of programming embedded systems, Academic Press, San Diego, USA.

Gendy, A. K., Pont, M. J. (2008a): Automatically configuring time-triggered schedulers for use with resource-constrained, single-processor embedded systems. IEEE Transactions on Industrial Informatics 4 (1): 37–46.

Gendy, A., Pont, M. J. (2008b): Automating the processes of selecting an appropriate scheduling algorithm and configuring the scheduler implementation for time-triggered embedded systems. Proc. of The 27th Int. Conf. on Computer Safety, Reliability and Security (SAFECOMP08), 22–25 September 2008, Newcastle upon Tyne, UK.

Gergeleit, M., Nett, E. (2002): Scheduling transient overload with the TAFT Scheduler. GI/ITG specialized group of operating systems, Berlin, 2002.

Hanif, M., Pont, M. J., Ayavoo, D. (2008): Implementing a simple but flexible time-triggered architecture for practical deeply-embedded applications. In: Proc. of the 4th UK Embedded Forum (September 2008, Southampton, UK).

Huang, H., Zhang, S., Cao, J., Duan, Y. (2005): A practical pattern recovery approach based on both structural and behavioral analysis. Journal of Systems and Software 75 (1–2): 69–87.

Hughes, Z. H., Pont, M. J. (2004): Design and test of a task guardian for use in TTCS embedded systems. In: Koelmans, A., Bystrov, A., Pont, M. J. (eds): Proc. of the 1st UK Embedded Forum (Birmingham, UK, October 2004): 16– 25. Publ. by Uni. of Newcastle upon Tyne [ISBN: 0-7017-0180-3].

Hughes, Z. M., Pont, M. J., Ong, H. L. R. (2005): The PH Processor: A soft embedded core for use in university research and teaching. In: Koelmans, A., Bystrov, A., Pont, M. J., Ong, R., Brown, A. (eds): Proc. of the 2nd UK Embedded Forum (Birmingham, UK, October 2005): 224–245. Published by University of Newcastle upon Tyne [ISBN: 0-7017-0191-9].

Hughes, Z. M., Pont, M. J. (in press): 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.

Imran, S., Short, M., Pont, M. J. (2008): Hardware implementation of a shared-clock scheduling protocol for CAN: a pilot study. In: Proc. of the 4th UK Embedded Forum (September 2008, Southampton, UK).

Keller, R. K., Schauer, R., Robitaille, S., Page, P. (1999): Pattern-based reverse-engineering of design components, Proc. – Int. Conf. on Software Engineering: 226–235, IEEE, Los Angeles, CA, USA.

Kirner, R., Puschner, P. (2003): Discussion of misconceptions about worst-case execution-time analysis. 3rd Euromicro Int. Workshop on WCET Analysis, 2003.

Kopetz, H. (1997): Real-Time Systems, Design Principles for Distributed Embedded Applications, Kluwer Academic.

Kurian, S., Pont, M. J. (2007): Maintenance and evolution of resource-constrained embedded systems created using design patterns. Journal of Systems and Software, 80 (1): 32–41.

Liu, C. L., Layland, J. W. (1973): Scheduling algorithms for multiprogramming in a hard real-time environment. Journal of the ACM 20 (1): 40–61.

Locke, C. D. (1992): Software architecture for hard real-time systems: Cyclic executives vs. Fixed priority executives. The Journal of Real-Time Systems 4: 37–53.

Mearns, D. D. U., Pont, M. J., Ayavoo, D. (2008): Towards Ctt (a programming language for time-triggered embedded systems). In: Proc. of the 4th UK Embedded Forum (September 2008, Southampton, UK).

Muhammad, A., Pont, M. J. (2008): Synchronising tasks in wireless multi-processor environments using a shared-clock architecture: A pilot study. In: Proc. of the 4th UK Embedded Forum (September 2008, Southampton, UK).

Nett, E., Streich, H., Bizzarri, P., Bondavalli, A., Tarini, F. (1996): Adaptive Software Fault Tolerance Policies with Dynamic Real-Time Guarantees. WORDS 96, IEEE Second Int. Workshop on Object-oriented Real-time Dependable Systems, Laguna Beach, California, U.S.A, 1996.

Noble, J., Weir, C. (2001): Small Memory Software. Addison Wesley.

Phatrapornnant, T., Pont, M. J. (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. (1998): Control system design using real-time design patterns. Proc. of Control '98 (Swansea, UK), September 1998: 1078–1083.

Pont, M. J. (2000): Designing and implementing reliable embedded systems using patterns. In: Dyson, P., Devos, M. (eds): EuroPLoP '99: Proc. of the 4th European Conf. on Pattern Languages of Programming and Computing, 1999. ISBN 3-87940-774-6, Universitätsverlag Konstanz.

Pont, M. J. (2001): Patterns for Time-Triggered Embedded Systems: Building Reliable Applications with the 8051 Family of Microcontrollers. Addison-Wesley/ACM Press. ISBN: 0-201-331381. Available for download from: http://www.tte-systems.com/books/pttes/.

Pont, M. J. (2003): Supporting the development of time-triggered co-operatively scheduled (TTCS) embedded software using design patterns. Informatica 27: 81–88.

Pont, M. J., Ong, R. H. L. (2002): Using watchdog timers to improve the reliability of single-processor embedded systems: Seven new patterns and a case study. In: Hruby, P, Soressen, K. E. (eds): Proc. of the First Nordic Conf. on Pattern Languages of Programs, 2002: pp. 159–200.

Pont, M. J., Mwelwa, C. (2003): Developing reliable embedded systems using 8051 and ARM processors: Towards a new pattern language. Paper presented at the Second Nordic Conf. on Pattern Languages of Programs, (VikingPLoP ''2003''), Bergen, Norway, September 2003.

Pont, M. J., Banner, M. P. (2004): Designing embedded systems using patterns: A case study. Journal of Systems and Software 71 (3): 201–213.

Pont, M. J., Li, Y., Parikh, C. R., Wong, C. P. (1999): The design of embedded systems using software patterns. Proc. of Condition Monitoring 1999 (Swansea, UK), April 12–15, 1999: 221–236.

Pont, M. J., Norman, A. J., Mwelwa, C., Edwards, T. (2004): Prototyping time-triggered embedded systems using PC hardware. In: Henney, K., Schutz, D. (eds): Proc. of the Eighth European Conf. on Pattern Languages of Programs (EuroPLoP 8), Germany, June 2003: 691–716. Published by Universitätsverlag Konstanz. ISBN 3-87940-788-6.

Pont, M. J., Kurian, S., Wang, H., Phatrapornnant, T. (2007): Selecting an appropriate scheduler for use with time-triggered embedded systems. Paper presented at the 12th European Conf. on Pattern Languages of Programs (EuroPLoP 2007).

Puschner, P. (2002): Is WCET analysis a non-problem? Towards new software and hardware architectures. 2nd Intl. Workshop on Worst Case Execution Time Analysis, Vienna, Austria, 2002.

Sheikh, I., Short, M., Pont, M. J. (2008): Hardware implementation of a shared-clock scheduling protocol for CAN: A pilot study. In: Proc. of the 4th UK Embedded Forum (September 2008, Southampton, UK).

Short, M., Pont, M. J. (2008): Assessment of high-integrity embedded automotive control systems using Hardware-in-the-Loop simulation. Journal of Systems and Software 81 (7): 1163–1183.

Tindell, K., Burns, A., Wellings, A. (1992): Allocating hard real-time tasks: An NP-hard problem made easy. Real-Time Systems 4 (2): 145–165.

Vallerio, K. S., Jha, N. K. (2003): Task graph extraction for embedded system synthesis. Proc. 16th Int. Conf. on VLSI Design concurrently with the 2nd Int. Conf. on Embedded Systems Design, 2003: 480–486.

Vidler, P. J., Pont, M. J. (2006): Computer assisted source-code parallelisation. In: Gavrilova, M., Gervasi, O., Kumar, V., Tan, C. J. K., Taniar, D., Laganà, A., Mun, Y., Choo, H. (eds): Proc. of the IEE Int. Conf. on Computational Science and its Applications (Glasgow, May 8–11, 2006), Part V. Lecture Notes in Computer Science (LNCS), 3984: 22–31.

Wang, H., Pont, M. J. (2008): Design and implementation of a static pre-emptive scheduler with highly predictable behaviour. In: Proc. of the 4th UK Embedded Forum (September 2008, Southampton, UK).

Wang, H., Pont, M. J., Kurian, S. (2007): Patterns which help to avoid conflicts over shared resources in time-triggered embedded systems which employ a preemptive scheduler. Paper presented at the 12th European Conf. on Pattern Languages of Programs (EuroPLoP 2007).

Xu, J., Parnas, D. L. (1992): Pre-run time scheduling processes with exclusion relations on nested or overlapping critical sections. 11th IEEE Int. Phoenix Conf. Computers and Communications, Scottsdale, AZ, USA, 1992: 774–782.

Author

Michael J. Pont
holds a B.Sc. from the University of Glasgow (UK) and a Ph.D. from the University of Southampton (UK). He worked at the University of Southampton and then University of Sheffield before joining the University of Leicester in 1992. He is currently a Reader in Embedded Systems and Head of the Embedded Systems Laboratory at the University of Leicester; he is also Founder and CEO of TTE Systems Ltd. Michael's main research focus is on the development of techniques and tools which support the design and implementation of embedded systems: he is particularly interested in the links between system architecture and key characteristics such as temporal predictability and power consumption. Michael is author or co-author of more than 100 technical papers, and is the author of three books. He is a Member of the IEEE, a Member of the SAE, a Member of the IET and a Member of the BCS.