Design patterns – From Architecture to embedded software development

Farah Lakhani[1] and Nabiha Faisal[2]

[1,2] Department of Computer & Software Engineering, Bahria University 13, National Stadium Road Karachi, Pakistan

Abstract

Developing application software for embedded systems presents many challenges as a number of constraints need to be optimized such as strict timings, limited power usage and memory utilization. Also, successful embedded application design are the result of the combined inputs from various related disciplines such as control engineering and scheduling theory. In this regard, recycling design experience can play a substantial role by providing previously tested and proven techniques incorporated into new designs. This paper reports on the evolution of the concept of 'Design patterns' originated by analogy from building architectures, and now followed by a wide variety of diverse disciplines. In the field of embedded software development patterns have been found to be a useful adjunct to traditional development processes. Nowadays practitioners in the field of successfully implemented patterns are encouraged to build reliable architectures for the safety-critical applications either from scratch or to transform existing architectures into more predictable forms.

Keywords

Embedded systems, Software architecture, design patterns

1. Introduction

The concept of patterns applied in this research has its roots in architecture and the generic principles and techniques applied in that discipline have received global recognition in their successful transition to applications in software engineering. 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 specialised 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 exclusively in the heads of experts but is captured in a way that can be easily accessible and shared. This paper will present a detailed account on the evolution of patterns and their successful transfer into a wide and diverse variety of fields with a special focus on embedded software development. The organisation of the paper is as follows: Section 2 will present an account on the historical background of patterns. The adoption of patterns into diverse disciplines is discussed in Section 3 and a more detailed discussion on different aspects of patterns presented in Section 4. Section 5 describes the use of patterns for embedded software development and the contribution made by this research. Finally conclusions of the research presented in this paper are given in Section 6.

2. 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 [1]. Alexander and his colleagues published three pioneering texts [2-4] that laid the foundation of the use of patterns in the field of architecture. They produced a "pattern language" [3] 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" [4]. In pattern language, Alexander and his colleagues proposed 250 innovative and coherent patterns for designing and building homes, towns and cities etc. The various patterns in the

language can be combined in different ways to build a 'customised' solution.

3. 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 organizational management to poetry, but in particular – and in the context of this research – in the world of computer software design.

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 [5]. 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 [6] comprising of five patterns aimed at helping new programmers to design windows-based GUI applications using the 'Smalltalk' programming language. Later, in 1995 Erich Gamma and his colleagues 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 [7]. 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 introduced in [8] documented as 'best practices' for productive software development, and the collection of patterns for introducing new ideas into an organization [9]. 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 [10] and [11]. Patterns for telecommunication systems focus on improving the two unique characteristics of software-reliability and human factors. Examples include the works described in [12 -14] 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 [15], the software development process [16], cognition [17] and software configuration management [18]. Recent research work has proposed the use of design patterns for web application development [19] and for the location-based GPS application [20].

4. Broader aspects of Patterns

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 [7], patterns for fault-tolerant software [14] and design patterns for high availability systems [21]. 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. A generic concept of a design pattern is depicted in Figure 1. The idea mainly revolves around the existence of a problem and a solution. The problem is expanded in terms of its context, and relevant design forces which provide foundations for the solution. The solution then generates a few benefits, consequences and follow-on problems which could lead to the applicability of other patterns.

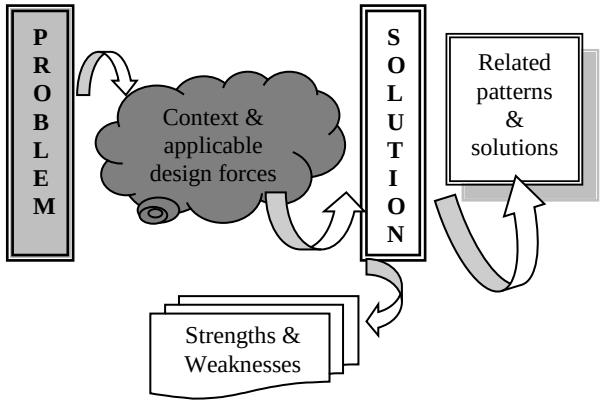

Figure 1: Illustrating the concept of a pattern

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 [22].

4.1 Pattern forms and Elements

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 1.

Table 1: Some well-known pattern forms

Pattern form	Description
Alexandrian form	This layout is used by Alexander in [2]
GoF form	Layout used to write the famous Gang of Four patterns [7]
Coplien form	This layout is followed by Coplien [8].
POSA form	This layout is used to write Patterns of Software Architecture [24].
PTTES form	This layout is used for Patterns for Time-Triggered Embedded Systems (PTTES) collection [27].

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

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. 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.

4.2 Pattern languages

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. 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'. Codifying good design practice helps to distil and to disseminate experience, and this helps others avoid frequently encountered development traps and pitfalls [23].

A repository of design patterns in an organisation can play an important role in replacing a loss of expertise or as an expert system. In this way patterns can help an organisation to "back up" key skills from a team of expert designers and provide a cheaper availability of the solutions. 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.

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" [24]. In the words of Alexander [2] "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 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.

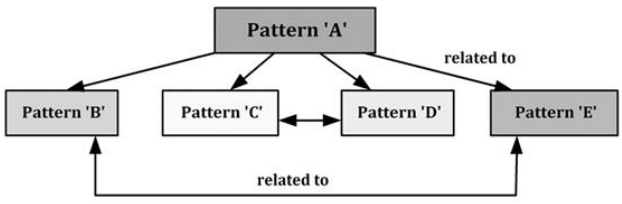

Figure 2: Illustrating an example structure of a pattern language

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.

4.3 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. This process is also called reverse-architecting. Several metaphors that were proposed such as hunting, fishing, harvesting, paleontology or archaeology and mining to describe the process of discovering and documenting patterns are discussed in [25]. 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.

Numerous international conferences on the Pattern Languages of Programming 'PLoP' 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. 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.

5. Patterns and Embedded Software

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, whereas embedded applications are built and tested in simulated environments and the generated executables are then transferred onto the target processor. Also designing successful designs for safety-critical embedded applications requires input from various other disciplines (shown in Figure 3) such as control theory, operations research, operating systems and Algorithms etc.

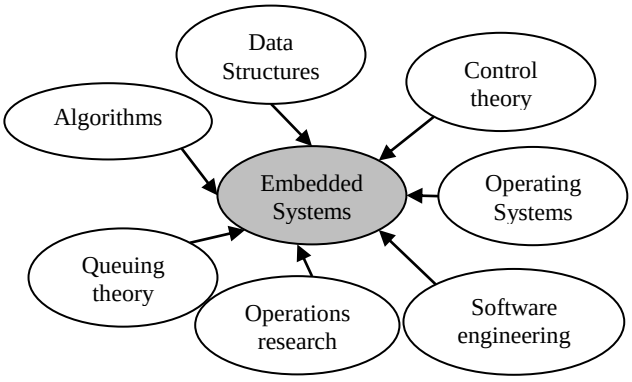

Figure 3: Disciplines that impact on embedded system engineering

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.

5.1 Patterns for system construction

A summary of some of the previously introduced pattern collections for embedded software development found in literature are given below:

1. A pattern language for designing simple embedded applications is introduced [26] and is based on a framework named 'The Carousel'. The basis of this framework is the well-known 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.

2. A huge collection of patterns for designing time-triggered embedded systems (called the 'PTTES' collection) [27]. 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.

3. A system of patterns for reliable communication in hard real-time systems called "Triple-T (Time-Triggered Transmission)" is also proposed [28]. 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.

4. A pattern language for distributed machine control system [29] emerged when the working engineers 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.

5. Other well-known examples of pattern languages in various domains of interest to control engineers are: patterns for concurrent and networked objects [30], communication patterns [13] and patterns for fault tolerant software [14].

5.2 Patterns for System Migration

A common requirement of all the pattern collections mentioned above is in providing techniques for designing systems from the scratch. However, pattern collection which could assist in alterations of existing architectures or migrating between different architectures were somewhat neglected. To fill this gap, a recent research was conducted on the use of design patterns for the migration between different architectures of embedded applications in order to improve the system reliability [31-32]. The focus of this work is on systems with "time-triggered" (TT) architecture the main alternative to which is an "event-triggered" (ET) architecture. These have already been distinguished by various research studies [33-36]. We further differentiate between these two alternatives as shown in Figure 4 below by defining static and dynamic variants of TT and ET systems. We assume that in a static TT system - we always know (i) when the next interrupt will occur, and (ii) exactly what the system will do in response to this interrupt. At the other extreme, we have dynamic ET system: in such designs we assume that (i) we never know when the next interrupt will occur, and (ii) that we do not know exactly what the system will do in response to this interrupt.

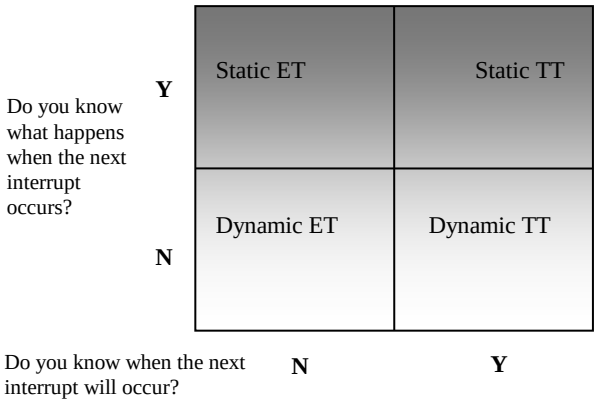

Figure 4: Distinguishing ET and TT designs

The research began with the hypothesis "Sets of rules, techniques and processes can be found which will be encapsulated in "pattern form" and which will let users apply a time-triggered approach in a wider range of systems".

Beginning from this point, research described in this paper 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. Finally, the 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 by demonstrating the applicability of the proposed patterns on real applications in laboratory experiments [37], and by conducting controlled experiments with a target audience of users [38]. It is beyond the scope of this paper to explain the details of these assessment studies. However results obtained from both studies suggest that design patterns can be successfully

implemented not only in the construction process, but they may also assist designers and developers in reducing the complexities involved in migrating between different architectures.

The association map for the proposed PMES pattern language is shown in Figure 5.

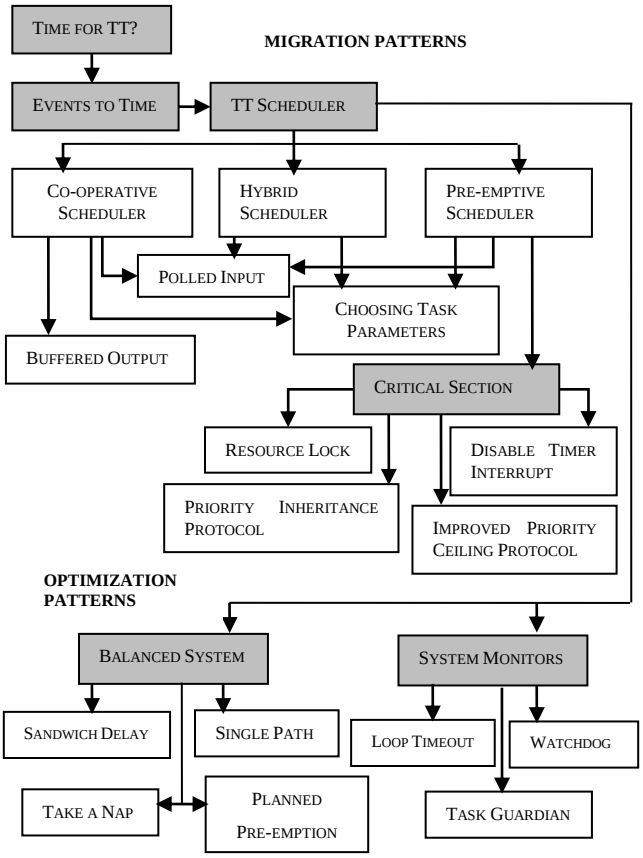

Figure 5: Association map for the patterns to support migration between architectures.

6. Conclusion

In this paper we have reported the successful technology transfer of what is called 'design patterns' originally developed for building architectures into various diverse disciplines with a special focus on software development for embedded applications. We have also introduced our novel contribution of the formation of a pattern language which could facilitate developers who are involved in altering design architecture in order to improve system reliability.

References

[1] Alexander, C. *Notes on the synthesis of form*, Harvard University Press, 1964.

[2] Alexander, C., S. Ishikawa, et al. *A pattern language*, Oxford University Press, 1977.

[3] Alexander, C., M. Silverstein, et al. *The oregon experiment*, Oxford University Press, 1975.

[4] Alexander, C. *The timeless way of building*, Oxford University Press,1979

[5] Beck, K., J. O. Coplien, et al. Industrial experience with design patterns. *Proceedings of 8th International Conference on Software Engineering*, Berlin, Germany (1996) 103-114.

[6] Cunningham, W. *The CHECKS pattern language of information integrity*, Addison Wesley, 1987.

[7] Gamma, E., R. Helm, et al. *Design patterns: Elements of reusable object-oriented software*, Addison Wesley, 1995.

[8] Cain, B. G., J. O. Coplien, et al. *Social patterns in productive software development organisations*. Annals of Software Engineering (1996) 2(1): 259-286.

[9] Manns, M. L. and L. Rising *Fearless change: Patterns for introducing new ideas*, Addison Wesley, 2004.

[10] Bergin, J. Fourteen pedagogical patterns. *Proceedings of the 5th European Conference on Pattern Languages of Programming*, 2000.

[11] Fricke, A. and M. Volter, A pedagogical pattern language about teaching seminars effectively. *Proceedings of the 5th European Conference on Pattern Languages of Programs*, 2000.

[12] Adams, M., J. Coplien, et al. *Fault-tolerant telecommunication system patterns*. Pattern Languages of Program Design 2. Boston , MA, USA, Addison Wesley (1996) 549-562.

[13] Rising, L., Ed. *Design patterns in communication software*, Cambridge University Press, 2001.

[14] Hanmer, R. *Patterns for fault-tolerant software*, John Wiley & Sons, Ltd, 2007.

[15] Borchers, J. O. Designing interactive music systems: A pattern approach. *Proceedings of the 8th International Conference on Human Computer Interaction: Ergonomics and User Interfaces* (1999) 1**:** 276-280.

[16] Ambler, S. W. Process patterns, Cambridge University Press, 1998.

[17] Gardner, K. M., A. Rush, et al. Cognitive patterns, Cambridge University Press, 1998.

[18] Berczuk, S. P. and B. Appleton, *Software configuration management patterns: Effective teamwork, practical integration*, Addison-Wesley Professional, 2003.

[19] Md. Umar Khan, Dr. T.V. Rao. *XWADF: Architectural pattern for improving performance of web applications*, IJCSI, Vol 11, Issue 2, No 2, March 2014.

[20] David Gillibrand, Khawar Hameed. *The use of design patterns in a location based GPS application*, IJCSI, Vol 8, Issue 3, No 1, May 2011.

[21] Kalinsky, D. *Design patterns for high availability* Whitepaper/Advanced Course Reading Assignment, D. Kalinsky Associates, 2002.

[22] Petter, S., D. Khazanchi, et al. *A design science based evaluation framework for patterns*. ACM SIGMIS Database (2010) 41(3).

[23] Jezequel, J. M. Train, et al. *Design Patterns and Contracts*, Addison-Wesley, 2000.

[24] Buschmann, F., R. Meunier, et al. *Pattern oriented software architecture.* Chichester, UK, John Wiley, 1996.

[25] DeLano, D. E. *Section title: Patterns Mining. The patterns handbook: Techniques, strategies, and applications*, Cambridge University Press, 1998.

[26] Bottomley, M. *A pattern language for simple embedded systems. Proceedings of the Pattern Languages of Programming PLOP '99*. Chicago, USA, 1999.

[27] Pont, M. J. *Patterns for Time-Triggered Embedded Systems*, ACM press, 2001

[28] Herzner, W., W. Kubinger, et al. "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.

[29] Eloranta, V. P., J. Koski, et al. Software architecture patterns for distributed embedded control systems. *Proceedings of 14th European Conference on Pattern Languages of Programming, EuroPLoP 2009.* A. Kelly and M. Weiss. Irsee, Germany, CEUR Workshop Proceedings. 566.

[30] Schmidt, D., M. Stal, et al. *Pattern-oriented software architecture Volume 2: Patterns for concurrent and networked objects*, John Wiley and Sons, 2000.

[31] Lakhani, F., A. Das, et al. 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 2010). Irsee, Germany,* ACM Press**:** 22:21 - 22:17.

[32] Lakhani, F. and M. J. Pont. 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.

[33] Kopetz, H. Event-triggered versus time-triggered real-time systems, 1991. *Proceedings of the Workshop on Operating Systems of the 90s and Beyond 87-101*.

[34] Albert, A. and R. Bosch GmbH. Comparison of event-triggered and time-triggered concepts with regard to distributed control systems. *Proceedings of the Embedded World 2004.* Nurnberg**:** 235-252.

[35] Kopetz, H. "Should responsive systems be event-triggered or time-triggered?" IEICE *Transactions on Information and Systems* (1993) E-76-D(11): 1325-1332.

[36] Scheler, F. and W. Schroder-Preikschat 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, 2006.

[37] Lakhani, F. and M. J. Pont. 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.

[38] Lakhani, F. and M. J. Pont . "Empirical studies for the assessment of the effectiveness of design patterns in migration between software architectures of embedded applications." ISRN, Journal of Software Engineering, 2012.

Dr. Farah Lakhani received her Bachelor's and Master's degrees in Computer Engineering from NED University Karachi and her PhD in Embedded Systems Engineering from the University of Leicester UK. She is currently working as Assistant Professor in the Computer & Software Engineering department at Bahria University Karachi Pakistan. Dr. Farah has published around 10 publications in various international conferences, forums and journals. Her area of research interests are software architectures for modern embedded applications, reliability issues for safety-critical applications and software engineering paradigms.

Nabiha Faisal received her B.E degree in Computer & Information Systems Engineering from NED University of Engineering & Technology, Karachi, Pakistan, her MEngg degree in Computer Architecture & System Design from NED University of Engineering & Technology, Karachi, Pakistan. Her research interests include Embedded Systems, Artificial Intelligence and Computer Graphics. She is currently working as an Assistant Professor in the Department of Computer & Software Engineering at Bahria University Karachi Campus, Pakistan.