View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by OAR@UM

A Software Development Framework for Hardware Centric Applications – An Architectural Perspective

Dr. Ernest Cachia
Department of Computer Science
University of Malta
Email: ernest.cachia@um.edu.mt

Michael Bonello
Department of Computer Science
University of Malta
Email: michael_bonello@hotmail.com

Abstract

Throughout the history of Software Engineering, software development has been looked at from various perspectives, in terms of: usability, suitability for proposed problem, speed of development, relevance to real world scenarios, as well as in terms of the hardware that it needs to manifest itself in the real world. This paper delves deeper into the aspect of the actual core concept in software engineering: that of mapping software onto hardware[1], focused specifically on Hardware Centric systems (HCS), (systems where the hardware dictates to an influential level, the actual nature of the software); examining the various frameworks and concepts that exist for displaying this mapping from an architecture point of view, so as to establish if there is a need for a more complete and/or effective framework. It also proposes a roadmap proposal for a base architecture framework for the development of Hardware Centric applications, which will then be employed to determine if a suitable framework already exists.

Gentle Note of Introduction

One is to note that this paper focuses on HCS, meaning systems whose software is configured according to hardware structure. In this regard, examples of hardware components that can classify a system as Hardware Centric could be: digital or analogue I/O cards, AtoD and DtoA conversion cards, sampling cards, Programmable Logic Controllers (PLC) like devices etc… As a general term of reference, HCSs are mainly systems that have a substantial degree of PC interfacing.

1. Introduction

Historically, software was written for a particular hardware platform. However the current trend, which started with the advent of programming languages usable across different processors, (with compilers available for different processors), is pointing in the direction of having source code that is almost completely abstracted from the underlying hardware. [2]

This said, in HCSs, this indicates an apparent gap in the current methods used to actually develop software for hardware specific applications, such as PC interfacing, or PC based automation.

The actual base concept of mapping software to hardware is quite natural when one looks at the concept from a PC interfacing, or from a CPU based system-automation perspective.

Mainly, the software that runs on the actual CPU needs to map some of its inputs and outputs to actual physical devices connected to it that give real world interaction to the system. For example; a PC controlled cardboard box labeling system will have a flashing beacon which is activated when a box is placed on the conveyor that leads to the label applicator. For most information systems, one considers user input from a keyboard, mouse touch screen etc… while output is through a pc screen, printer etc… This leads to the adoption of abstraction in terms of hardware to software mapping, in that the actual software does not need to be fully aware of the underlying hardware. The underlying hardware just needs to satisfy some 'general' parameters that the software needs in order to operate successfully.[1]

Therefore one can reason that there are two extremes within the same concept, in one case the software needs to be fully aware of the hardware, while in the other the software is only generally aware of the underlying hardware.

A balance needs to be struck in between, and the actual marker on the balance scale will be affected by the type of the system being developed, as well as the scope in which the system is being developed.[3]

2. Styles of Design

Trends nowadays are pointing to various styles of system architecture design, which somewhat contrast with the styles of system design that were generally used for HCS. This mainly due to the fact that HCS were analyzed almost exclusively from the hardware limitations perspective, in terms of such parameters as: cycle times, sensors and actuator interfacing, heat buildup, power needs etc… Nowadays almost all information systems software design is centered on one or more of the following [2] [4] aspects:

2.1. Sharp Focus on System Architecture

System architecture can be considered the actual foundation of any system, due to the fact that it gives an overall vision to anyone working on the system in terms of the overall system functionality. This is essential for anyone that needs to take a calculated decision on anything regarding the system's design, development, maintenance and continuous improvement. This is because it gives an actual boundary in which these decisions can be taken. It also provides a model which describes how the system should evolve into the desired architecture, therefore assisting in project planning and project management.

2.2. Component Based Architecture and Design

The value of actually designing the system architecture in terms of components ties in with the Rapid Application Development concept (below) as well as with the concept of manageable portions of logic within the system, (the principle of divide and conquer). Therefore component based architecture is a mechanism that will help in separating concerns, in that each component will be responsible for particular functions within the system. This is beneficial in terms of both development (developers will be challenged with more manageable portions of logic at a time) as well as in terms of testing, (as testers can test one component at a time), reducing the perceived complexity of the system.[4]

2.3. Rapid Application Development

Rapid application development is the major trend in modern business applications. It is based on the development concept of providing what is needed by the business in small iterations with each iteration building on past iterations. The advantages of this approach is multifaceted in that, while providing the business side of the project with insight into what is actually happening within the project, the technical side of the project has the flexibility to incorporate new business needs in subsequent iterations. [5]

2.4. Architecture and Design Testing

The concept of actually testing the architecture and design that was submitted by the system architects / system designers is very beneficial by allowing crucial flaws to be checked early within the system development life cycle therefore saving both time and money for the client. [6][3]

3. Problem Definition

The question that this research effort aims to answer is if there is a software development framework, already developed, that can suitably model the varying level of software to hardware mapping, in terms of software architecture mapping and software design, for HCSs. It should be stressed that the intention is on finding a framework that can be adapted to different levels of abstraction, according to the specific hardware being used. The framework would also need to comply with the existing styles of system design as mentioned in the preceding section.

4. Historical Trends in HCSs Architecture

As will be mentioned later on in this paper, historically, hardware that was needed for PC interfacing applications needed to be built ad hoc, for various reasons, namely:

- High cost of off the shelf (OTS)hardware
- OTS hardware being bound to particular hardware architecture (ex: Z80 platform, Motorola 6800/0 command set etc…)
- OTS hardware did not fully address system functionality.

Another long-standing problem was due to the actual approach that was taken when building a HCS. The hardware almost completely dictated the functionality of the software. This was also due to the fact that the PC interfacing hardware that was available was limited in terms of functionality and versatility.

Finally, a problem that afflicted most realms of software engineering at the time, was the concept of piecemeal software development. This is when the software architecture was built piece by piece, with no notion of initial overall architectural vision that will, at later stages, guide system design and development. [8]

5. Current Trends in HCSs Architecture

If one was to look at the current trends in HCSs, one would note that various architectural models and system modularisation concepts have been introduced and maintained in a bid to standardise certain core interfacing concepts. Architectures such as those based on the MODBUS standard [9] (emerging from PLC device communication), standardise

the actual architecture of HCSs up to a certain extent, by providing a basic starting point off which the actual application specific architecture can be built. Other similar architectures, albeit focused on different aspects of the actual system to be constructed, are developed and used by major embedded system component production houses. Such organisations see a need to standardise their products in such a way that as system architecture built using one manufacturer's products can be easily translated to another's without major architectural overhauls. Such architecture driven concepts gave rise to the I^2C harmonise[10], as well as other standards that standardize various architectural and design aspects of HCSs.

6. Business Needs Mapping and Technological Restriction Mapping

Traditional embedded / electronics oriented applications had various deficiencies in terms of the actual mapping of the business needs of the clients[11] in relation to the actually developed solution. This was mainly brought about by the lack of availability of needed hardware, as well as the relevant bundled software drivers. Coupled with this, one needs to consider the fact, that if custom hardware needed to be built, one needed time to design and develop this hardware, as well as time to test it in isolation. Mainly due to the fact that such hardware is custom built for the first time for a particular application.[12] Therefore, one can say that the real problem was that either the available hardware did not match the business requirements fully in most cases, or that the custom hardware needed to be built as a stop-gap solution to this.

With the advent of more sophisticated and versatile PC interfacing devices, most of which came with bundled low-level drivers, this timeliness constraint in terms of building customised hardware was minimised. Also minimised was the need to test the hardware in isolation, as this OTS hardware would have, in most cases, been rigorously tested by its manufacturer. The afore mentioned traditional gap between business requirements and the actual available hardware, was also minimised considerably due to the increasingly business oriented market that supplies such hardware. Such hardware is nowadays, infused with a multitude of business / engineering workshops that manufacturers organize, specifically to bridge this gap.

7. Architectural Restrictions Issues in HCSs

Not particularly pertaining to HCSs, but still worth a mention in the context of their influence on the actual extended research which this paper is a precursor to, are the issues of architectural restrictions.

As mentioned earlier, issues of architectural restrictions are not solely bound to HCSs, although, several limiting factors do exist, which may influence the actual structure of an architectural mapping / development framework applicable to a particular scenario.

Different applications have different architectural restrictions. A short summary of the main architectural grievances that could afflict a HCS could be the following:

- Lack of a standardised architectural modus operandi in using the desired hardware i.e. standardisation of how

particular hardware should be used
- Bridging of programming mentalities between custom built software and OTS hardware controlling software
- A decision making process to decide between custom built hardware or OTS hardware
- Architectural completeness vis-à-vis cost of development and time to market concepts

8. What a Proposed Framework Should Address

An adequate framework should:

- Address the concept of adaptable abstraction in terms of software to hardware mapping. This because one would not actually know the level of detail that is included with OTS hardware, i.e. the drivers or ancillary software that is bundled with the hardware

- Satisfy the needs of modern system design i.e. should focus on architecture, system architecture and design testing, application of rapid
- application development and the system should be componentized, as outlined in section 2 (above).

- Cover the full software development life cycle (preferably based on an adapted RAD life cycle)

- Cope with Hardware centric architecture restrictions, as mentioned in the previous section

- Adapts to the business perspective of the system as well as the business requirements that the system is intended to address

- Provides adaptability in terms of the actual universe of discourse in which various HCSs could be built

9. References

[1] G. Mittal, D. Zaretsky, T. Xiaoyong, P. Banerjee, “An Overview of a Compiler for Mapping Software Binaries to Hardware” IEEE Transactions on Very Large Scale Integration (VLSI) Systems Volume 15, Issue 11, Nov. 2007 Page(s):1177 - 1190

[2] Richard N. Taylor and Andre van der Hoek “Software Design and Architecture: The once and future focus of software engineering” *IEEE Future Of Software Engineering, 2007*.

[3] Björn Hartmann, Scott R. Klemmer,Michael Bernstein, Leith Abdulla, Brandon Burr, Avi Robinson-Mosher, Jennifer Gee “Reflective Physical Prototyping through Integrated Design, Test, and Analysis” *UIST’06, October 15–18, 2006, Montreux, Switzerland*

[4] Johan Fredriksson, Massimo Tivoli, Ivica Crnkovic “A Component based Development Framework for Supporting Functional and Nonfunctional Analysis in Control System Design” *ASE 2005. Novermber 7-11 2005, Long Beach, California*

[5] Alan Howard “Rapid Application Development: Rough and Dirty or Value-for-Money Engineering” *Communications of the ACM October 2002/ Vol 45, No. 10* pp 27-28

[6] Einar Landre, Harald Wesenberg, Jorn Olmehim “Agile Enterprise Software Development Using Domain-Driven Design and Test First” OOPSLA 2007 October 21-25 2007, Montreal, Quebec, Canada pp. 983-985

[7] Paolo Ciancarini “On the Education of Future Software Engineers” ICSE 2005, May 15-21, 2005, St. Louis. Missouri, USA

[8] Fuqing Yang and Hong Mei “Development of Software Engineering: Co-operative efforts from academia, government and industry” ICSE 2006, May 20-28, 2006, Shanghai, China

[9] Modbus-IDA “Modbus Application Protocol Specification V1.1b” *http://www.Modbus-IDA.org* December 28, 2006

[10] “The I2C-Bus Specification” Philips Semiconductors Version 2.0 January 2000 doc or: 9398 393 40011

[11] Betty H.C. Cheng and Joanne M.Atlee “Research Direction in Requirements Engineering” *IEEE Future Of Software Engineering, 2007*.

[12] F. Vermeulen, F.Catthoor, D. Verkest, H. De Man “Extended Design Reuse Trade-Offs in Hardware-Software Architecture Mapping” *CODES 2000, San Diego, CA USA ACM 2000 1-58113-268-9/00/05*