Partitioning for a Concept-Based SoC Design to Address Product Evolution

Waseem Ahmed[1], Adrus Mohamad Tazuddin[2], Wong Ming Ming[3], Liaw Tzy Renn[4] and Douglas Myers[5]
Curtin University of Technology, Sarawak Campus, Miri, Malaysia
[1]waseem@curtin.edu.my, {[2]ultimate_cloud, [3]classic_phone, [4]sim_liaw}@hotmail.com,
[5]Curtin University of Technology, Perth, Australia
d.myers@exchange.curtin.edu.au

Abstract- **To address changes encountered during product evolution in SoCs, a concept based design methodology based on UML 2.0 is presented in this paper. A system specification given in the form of UML 2.0 sequence and class diagrams is segregated into concepts with a Generic Adaptive Layer (GAL) around them. GALs offer generic adaptability to existing concepts and change localization during product evolution. The major advantage of using the presented methodology is an improvement in the development time of the system during both the design and the maintenance phases. Experiences from using three categories of concepts are presented.**

I. INTRODUCTION

The last decade has seen a substantial increase in the number and variety of embedded systems, which range from the cheap and simple devices used in consumer electronic appliances to the more expensive and sophisticated devices used in industrial robotics and aviation.

There have been three noticeable trends in embedded systems. The first two are an increase in size and complexity of the embedded devices. Some of the contributing factors for this increase in size and complexity are application convergence, higher density of silicon chips which subsequently allow more functionality to be put on them, increased software-to-hardware ratio, Multi-Processor System-On-Chip (MPSoC), distributed embedded systems and Network-on-Chip (NoC). The third noticeable trend is product evolution. Embedded systems, particularly those in the consumer electronics domain, exhibit an extremely high turnover in terms of product releases. A company that spends 6-18 months developing a new product from scratch, incrementally changes this product for its subsequent releases. These changes range from simple bug fixes to more complex porting of the application to a different operating system or implementation platform. These systems, in general, have many such changes imposed on them. To retain or increase their market share, a system provider has to cater to these changing demands in a timely manner or face market loss.

The common approaches to address increase in size and complexity have been reuse, product lines, component-based design and the use of visual design paradigms like UML and MDA. Systematic product evolution, however, has been a problem. Evolution involves maintenance of code and is not easy if no pro-active maintenance measures have been taken at the design phase. For a system to be useful beyond its first release, it has to be malleable and accommodative to both internal changes and changes imposed by the external environment.

In terms of functionality, the difference between consecutive releases of the system may not be large. But the development effort that goes into performing adaptive, corrective or preventive maintenance on the system [1] for the next release or version is substantial. This effort is further complicated if the base architecture of the system is not coherent to the developers performing maintenance of the system. The preservation of the base architecture of the system is dependent on the way the initial partitioning of the system has been performed. The role of a partitioning algorithm in the codesign of mixed hardware-software systems is to either maximize or minimize a cost function based on a set of quality constraints like performance, size and energy consumption. While attaining its intended objectives it, however, results in a breakdown of the architecture. The resultant code is fragmented, unstructured, unmanageable and incomprehensible to anyone outside the core design group and the initial base architecture is lost after partitioning and cannot be easily obtained though reverse engineering. Partitioning is extremely detrimental to product evolution and works against the good design principles of software engineering and reuse.

This necessitates a strategy that addresses system design more holistically. A concept based architecture that uses generic adaptive layers (GAL) has been proposed in this paper that improves reuse and modularity while taking a pro-active approach to maintenance in the design phase.

The rest of the paper is organized as follows. Section II compares the presented methodology to existing similar approaches. Section III describes the GAL. Section IV describes the need for a Concept Repository and its maintenance. This is followed by the section on Concept Identification and the automation of GAL generation. Section VI describes the case study followed by conclusions in section VII.

II. RELATED WORK

There have been various approaches to address the