System Architecture in a Mechatronics Perspective

De-jiu Chen
Mechatronics Lab
Machine Design, KTH
chen@damek.kth.se

Abstract

Architecture for mechatronics systems is an ongoing research at the Mechatronics Lab, KTH. This paper is the problem statement of current state-of-art study of the embedded real-time system architecture. It discusses some problems in developing the embedded real-time systems and maintains the importance of clear system architecture. To clarify the meaning of the term "architecture", a stringent definition of "system architecture" is described. This leads to the definitions of a set of subordinate domain architectures in the system. At last, the concepts of architecture based system development are introduced.

Keywords: system, mechatronics system, embedded system, real-time system, control system, robotic system, system architecture, system architecting.

1 Introduction

A MECHATRONICS (MECHAnics and elecTRONICS) system is a machine with intelligent behavior. To acquire this behavior, a mechatronics system is often a computer-based implementation of control theories and robotic theories. In the system, there are three subsystems, that is, a target environment or controlled system, a controlling system, and an operator. Since the controlling system has to control dynamics of the physical components in the target environment in a predictable and effective way, it is also called embedded (or dedicated) real-time system. Typical implementation fields for the embedded real-time system can be aircraft, spacecraft, road or rail vehicles, and robotics. Definitions of these systems are as the following.

- Embedded system: A computer system that forms an integral part of a larger system.
- Real-time system: A system for which the success of the execution does not only depend on the logical correctness of the results that are being produced, but also on the time at which the results are obtained [Stankovic91].
- System: A set of different elements so connected or related as to perform a unique function not performable by elements alone [Rechtin&Maier97].

2 What is System Architecture

Nowadays, the term "architecture" is often over-utilized or even misused. Originally, the word "architecture" has three meanings according to [Oxford95], i.e., the art and science of designing and constructing buildings; the design or style of a building or buildings; the nature and structure of a particular/computer system that determines the way it operates. Consequently, the word "system architecture" should be defined as,

- the art and science of designing and constructing systems;
- the design or style of a system or systems;
- the nature and structure of a particular system that determines the way it operates.

This shows clearly that architecture is not only a high-level design, or only the overall structure of the system.

The first definition indicates that system architecture is about the design issues; in

addition, it points out both science- and practice-based methodologies should be implemented to delimit and guide the system design. For example, the four most important methodologies in the process of architecting can be categorized as *normative*, *rational*, *participative*, and *heuristic* [Rechtin&Maier97]. The normative techniques are solution-based and prescribe solution structures as it "should be", i.e., as given in design handbooks; the rational techniques are method- or rule-based, i.e., as given in scientific and mathematical principles to be followed in arriving at a solution to a stated problem; the participative techniques are stakeholder based; the heuristic techniques are lessons based. The last two categories of techniques are used to handle the art/practice problem, i.e., non-analytical factors, which can not be analyzed analytically by the first two categories of techniques.

The second definition suggests that the system architecture is also about well-known, wide-accepted design decisions and generic structures. These mature problem solutions are often called styles and patterns, which are resulted by the first architecture definition and expressed according to the third architecture definition. By giving insight to the particular problem, they make designers having better apprehension of the inner nature of the system and hence make the design more effective and risk-free.

The third definition points out that the system architecture is about the behavioral and structural issues of a particular problem solution. Especially, it is about the organization of components and their interrelationship to solve a system design problem. Typical activities in this process can be generating and comparing (initial) solution alternatives. To make this possible, the representation of the system's behavioral and structural issues becomes essential.

Consequently, system architecting is the process of solving system design problem by organizing the system components and their interrelationships. It strives for fit, balance, and compromise among tensions of clients' needs and resources, technology, and multiple stakeholder interest [Rechtin&Maier97].

These three definitions complement each other and constitute the entire meaning of "architecture" and "architecting". That is, while the last definition addresses architecture's property for a particular problem solution, the first and the second one indicate its general features. Some existing system architectures and their position in the context of these architecture meanings are described in Table 1.

Name	Description	Coherence	Definitions
Guards [DeVa97]	**G**eneric **U**pgradable **A**rchitecture for **R**eal-time **D**ependable **S**ystems [DeVa97] • Providing a "generic" fault-tolerance architecture and components. • Providing methodology to combine components.	⭘	Def I
		⭘	Def II
		●	Def III
TTA	**T**ime-**T**riggered Computer **A**rchitecture [DeVa97]including • communication principles and communication interfaces of nodes; • communication protocol with clock synchronization and redundancy management; • 'Verification procedures' for certification of TTA products.	⭘	Def I
		⭘	Def II
		●	Def III
OSA-CA	**O**pen **S**ystem **A**rchitecture for **C**ontrols within **A**utomation **S**ystems: A reference architecture which details the functionality, structure and configurations of control applications in automation.	○	Def I
		⭘	Def II
		●	Def III
SIMP-LEX	A methodology to accomplish systematic fault-tolerance. [Sha et.al.96][Sha97]	○	Def I
		⭘	Def II
		⭘	Def III

Table 1-Practical architectures and their coherence with the architecture definition. (●-Strong, ⭘-Fair, ○-Weak)

3 Different Domain Architectures

The unique function of the embedded real-time system can be divided into a set of incorporating subordinate functions given by several subsystems. For example, in a robotic system, there are dynamics control functions, mission management functions and external communication functions. Then, all these functions are realized by software running on hardware platform in a computer system. Consequently, the entire system architecture can be hierarchically divided into several subordinate domain architectures for these incorporated functions.

For control of dynamics, the control theory should be implemented. How this theory provides methodologies, suggests mature solutions, and also structures the solution related to the early design for a particular control design problem, are considered as the architecture issues of the control system.

For more intelligent behavior other than the dynamics control, more functionality has to be provided by the system. This is typically illustrated in the robotic system, which should also be able to discriminate stimuli, classify features, recognize objects, and eventually create a symbolic representation of the sensed environment. This complex activity is called “perception process” [Buttazzo96], which suggests the (high-level) solution structure of this particular problem and affects then the software and hardware architecture of the entire system. Together with the methodologies and mature solutions related to the early design given by robotics, it constitutes the architecture issues of the robotic system.

Often, the system functionality or intelligence is realized or implemented by software. Software architecture is about the *structural issues* of a software system [Clements&Northrop98] at a more abstract level than algorithms and data structures. It describes and prescribes software components, roles and relationships of the components, early software design methodologies, styles/patterns and models. Although there are various definitions of software architecture, the bottom line is that software architecture is about structural properties of a system, including components, their interrelationship and principles about their use, which is the definition given by Garlan and Dewayne (1995) [Clements&Northrop98].

Clearly, software has to be mapped on hardware platform to be operational and the system has to interact with the environment by hardware components. Thus, how the hardware platform is structured affects the system characteristic. For instance, the implementation of embedded systems as distributed systems, i.e., multiprocesses on multiprocessors with inter-processor communication (IPC) links, is preferred for throughput /performance, cost-effectiveness, COTS products [Wolf94] and the nature of implementation.

Even though these domain architectures contribute to different system characteristics, they do interact with each other. For instance, in order to grant the modifiability/maintainability, usability and understandability of the entire system, the active perception process of a robotic system may have a layered software architecture. Since this solution may generate extra execution overhead, the performance and reliability offered by the software and hardware run-time platform in certain architecture may not be enough for supporting the dynamic characteristics of the controlled object anymore. Consequently, this software architecture becomes infeasible on the platform.

Therefore, these domain architectures have different priorities during the system architecting. The control or robotic system architecting forms actually an important stage before the computer system architecting can even begin since control functions are often the most important functionality of the mechatronics system and should be used to constrain other architectural or conceptual design works. Functionality

and performance should be the first class quality attributes.

4 System Development Process and Design Methodology

Developing and constructing an embedded real-time system can become a complicated and formidable challenge for engineers. This difficulty can be illustrated in the following points,

- First, the most fundamental requirement on an embedded real-time system is that the system must be able to provide logically and timely correct interactions with the environment, including the controlled object and the operator. Therefore, a set of functional and temporal application constraints on the system components and their interrelationship, e.g., periodicity, deadline, offset, jitter, precedence, exclusion, locality and communication latency, should be clarified. These application constraints constitute the functional and performance of the system quality attributes. Moreover, since these quality attributes are imperative, it defines also the system design space.
- Second, a number of subsystem functions, e.g., dynamics control functions, mission management functions, external communication functions, should be incorporated properly to provide desired system functionality.
- Third, when both software and hardware are involved, inappropriate usage might decrease the understandability, maintainability verifiability and even dependability of the system. For example, unconstrained "coding desire" can introduce unnecessary or failure-prone complexity in the system and keep the engineering cost high, despite the flexibility offered by software.
- Fourth, many stakeholders with different or even conflicting interests have to be "satisfied". Typical system quality attributes can be predictability, functionality, availability, reliability usability, modifiability/maintainability reusability, integrability, testability and verifiability. How to prioritize these quality attributes and find balance is an important high-level design issue.

Therefore, there are a lot of variables to be considered. As a matter of fact, complexity is the nature of the system, which is a complex composed of dynamically and statically interconnected parts. It is obvious that the more components and interrelationships exist in a system, the more complex the system becomes, and the more difficulties and problems can be expected in design and development.

The system development process often begins with the requirement acquisition and ends with the system delivery. This process can be described with three major steps [Gajski et. al. 94]: functionality specification, system design and component implementation. In the functionality specification, the entire system's desired functionality is defined and described. Then, the specified functionality is partitioned among allocated standard and custom components for more detailed component design. For embedded real-time systems, four major tasks can be identified in the design process [Wolf94]: *partitioning* the function, *allocating* those partitions to HW components, *scheduling* the times and *mapping* a generic functional description into an implementation on components. At last, each component's functionality is implemented as hardware or software.

No matter what development process model or life cycle model is utilized, the degree of design incoherence or the depth of iteration determines the cost of iterations. In fact, the reason for iterations is to assure the job quality of each stage and remedy eventual problem at earlier stages. A late found inconsistency with the requirements or unfeasibility of the system design is absolutely undesirable. Therefore, perspective, insight and integrity are required in the early phase of

the development process to avoid system level design problems and achieve the design target.

To get perspective, insight and integrity in the early design phase, systematic design methodologies and tools for embedded system design are needed. It should be able to,

- assure conceptual integrity by handling the complexity in different abstraction levels (with architectural models/views);
- provide perspective by discovering eventual system level inconsistencies, implementation or feasibility problems early in the design process and facilitating eventual modification and maintain tasks;
- give insight by facilitating utilization of earlier design decisions or experiences, modules and other COTS (Commercial Off-The-Shelf) products;
- assisting in keeping design integrity by constraining design works.

To handle the complexity, it is important to recognize the concept that a complex system can be progressively partitioned into smaller and simpler units – and hence into simpler problems – omits an inherent characteristic of complexity, the interrelationships/cohesion among the units (in certain abstraction levels) [Rechtin&Maier97][Cooling91]. For instance, the typical attempt to handle the complexity of the system is by modularization in mechanics, electronics and software, which leads to distributed and decentralized systems potentially allowing systems to be built from components; however, poor aggregation and partitioning during development can increase complexity [Rechtin&Maier97].

5 Architecture Based System Development

Architecture based system development of the embedded real-time system should strive to create the complex system from architecture. In this process, the system architecting is tightly connected with the requirement acquisition and the system specification. It uses both practice- and science-based techniques to balance and trade-off various functional and non-functional system quality attributes and classify solution alternatives by assessing feasibility, before the detailed system implementation even exists. Having good system architecture is a necessary (but not sufficient) condition for the final quality of the designed system.

A typical design process for embedded real-time systems can run sequentially from requirements acquisition and specification, initial solution architecture, system specification, design of HW&SW components, to integration and system testing with eventual iterations for modification and clarification. In the architectural level, which includes iterated requirements acquisition, initial solution architecture and system specification, the design has the following features.

- First, architectural components and interrelationships realizing the functionality prescribed by the control system or the robotic system should be defined after an initial requirement acquisition. Here, not only “problem-solution” based approach, but also “solution-problem” based approach should be adopted by means of architectural styles, patterns and principles. For a particular design problem, the similarities and variations compared to the established styles and patterns can be identified by domain analysis.
- Then, a properly represented initial solution architecture with architectural models/views for certain behavioral and structural properties forms a powerful basement for requirement refinement, trade-off analysis, feasibility assessment and architectural modifications with tools like ATAM[Kazman et al 98], SAAM[Bass&Clements&Kazaman98] and AIDA [Redell98].
- At last, this initial solution architecture constrains the following detailed implementation and keeps the design integrity.

Therefore, architecture based development does not only assure conceptual integrity and promote understandability of the complex system among the stakeholders by handling complexity, but also provide perspective to the all design process and gives insight by reusing earlier design decisions.

6 Conclusions and Further Research

The objective of this paper is to illustrate the importance of architecture for the design of complex embedded real-time systems in the context of the mechatronics system. Since a well-defined architecture brings perspective, insight and methodology in the early phase of the system design process, it forms the important base for requirement acquisition and refinement. Moreover, by means of architectural models/views for certain behavioral and structural properties at high abstraction level to handle the complexity of the system, it also assures the conceptual integrity among the stakeholders.

However, to carry out the system architecting including early feasibility assessment of architecture alternatives, it is required to know characteristics of the architecture components and interrelationships rooted from the control system or the robotic system, the hardware and software components (CPU, memory, process), operations of the network of components and the topology. This might be a hard work since some performance data can not be fully created at the architectural level. Therefore, further research should consider how to construct and represent the system architecture constrained by early performance analysis. When complete performance data can not be acquired, this early performance analysis should assist finding problems of unfeasibility and inconsistency in time in a more detailed and accurate analysis for changing the design.

7 References

[Bass&Clements&Kazaman98]
L. Bass, P. Clements and R. Kazaman, *Software Architecture in Practice*, ADDISON-WESLEY, ISBN 0-201-19930-3, 1998.

[Buttazzo96]
G. Buttazzo, *Real-time issues in advanced robotics applications*, Real-Time Systems, 1996, Proceedings of the Eighth Euromicro Workshop on, Page(s): 133 –138, ISBN: 0-8186-7496-2

[Clements&Northrop98]
Paul C. Clements, Linda M. Northrop, *Software Architecture: An Executive Overview*,Software Engineering Institute, Carnegie Mellon University, Technical Report CMU/SEI-96-TR-003, ESC-TR-96-003.

[Cooling91]
J.E. Cooling, *Software Design for Real-time Systems*, International Thomson Computer Press, 1991. ISBN 1-85032-279-1.

[DeVa97]
DeVa, *Guards-TTA joint selective open workshop*, 29 Sept - 1 Oct 1997, LAAS-CNRS Toulouse, France (Joint workshop report No. J1 Oct. 97)

[Gajski et.al. 94]
D.D. Gajski, F. Vahid, S. Narayan94, J. Gong, *Specification and Design of Embedded Systems*, Prentice Hall, 1994, ISBN 0-13-150731-1.

[Kazman et al 98]
Rick Kazman, Mark Klein, Mario Barbacci, Tom Longstaff, Howard, *The Architecture Tradeoff Analysis Method*, Lipson, Jeromy Carriere , 0-8186-8597-2/98, 1998 IEEE

[Oxford95]
Oxford Advanced learner's Dictionary of Current English, 5th edition, Oxford University Press, 1995. ISBN 0194314219.

[Rechtin&Maier97]
Eberhardt Rechtin, Mark W. Maier, *The Art of System Architecting*, ISBN 0-8439-7836-2, CRC Press, 1997.

[Redell98]
Redell, O., Modelling of Distributed Real-time Control Systems, An approach for Design and Early Analysis, Licentiate Thesis, KTH, Machine Design, TRITA_MMK 1998:9, ISSN 400-1179, ISRN KTH/MMK-98/9-SE.

[Sha et.al.96]
L. Sha, R. Rajkumar, M.Gagliardi, *Evolving Dependable Real-time Systems*, The Proc. Of the IEEE Aerospace Conference 1996.

[Sha97]
L. Sha, *Shifting the Computation Paradigm of Real-time Control Systems*, in SNAERT' 97 Preprints, Lund University of Technology, Sweden, August 21-22, 1997.

[Stankovic91]
J.A. Stankovic, *Real-time Operating System*, Real-time Computing-NATO ASI Series, Seies F: Computer and System Sciences, Vol. 127, 1991.

[Wolf94]
Wayne H. Wolf, *Hardware-Software Co-Design of Embedded Systems*, Proceedings of the IEEE, VOL. 82, NO. 7 July 1994.