Research Issues on System Architecture for Mechatronics Systems

De-jiu Chen
Royal Institute of Technology, Mechatronics Laboratory, Department of Machine Design,
100 44 Stockholm, Sweden
chen@damek.kth.se

Abstract: The ARTES project no: A4-9805 aims to progress the state of the art on software architectures for complex control systems. A specific goal of the project is to provide a system architecture suitable for scalable, maintainable and reconfigurable mechatronics systems controlled by an embedded distributed computer system. This are requirements found by the legged locomotion systems that are being studied and used as a case study within the project. In this report, we briefly introduce the system characteristics and give introductory knowledge on the key research issues around system architecture.

Background

A mechatronics system embodies technologies from several engineering disciplines in the domains of mechanics, automatic control, computer software and hardware. In such a system, the major portion of system functionality is realized by adopting automatic control through computer software and hardware, as well as a variety of sensors and actuators. As this approach promises enhanced functionality, performance and flexibility, more and more traditional mechanical solutions in modern machinery are being improved or replaced by the mechatronics solutions. For example, *ABS* (Antilock Brake Systems) for the performance of the breaking system of a car or the *x-by-wire* flight controls in a modern passenger airplane.

There is inherent complexity in this multidisciplinary approach. First, the complexity arises from the multidimensional interdependency across the involved domain technologies (i.e., mechanics, electronics, automatic control, computer software and hardware). To complete a specific task or to satisfy a desired non-functional quality attributes (e.g., versatility, modifiability, maintainability, etc.), this interdependency needs to be correctly established and handled in the system. Second, the software makes the system structure and behavior less comprehensible and controllable. This is caused by the lack of physical constraints and the large "state-space" of software (e.g., the fine-grained codes and their complex interrelations). In contrast to the physical constraints in other engineering disciplines, there are only storage and (temporal) performance constraints on software. This gives tremendous design freedom of software; however, also makes the design more error-prone. As a mechatronics system has physical impact on its environment, software failure is often intolerable.

To facilitate the development and maintenance of the system, a variety of measures managing the complexity are needed. At this point, managing the software complexity becomes extremely important for the system functionality and dependability. In the following sections, we will list some key concepts on how to handle the complexity with the focus on software. The aim is to illustrate some important research issues around the system architecture.

An architecture-based design approach

To handle the complexity, the design must progress in a top-down and evolutionary manner. In a systematic top-down approach, the system is hierarchically decomposed into lower level constituent units and their relationships. With the abstractions, we can concentrate on the system temporal and spatial structures, which in turn determine the essential properties of the system. For software, it means that programming-in-the-large and the programming-in-the-small issues are separated. Meanwhile, the design should also progress evolutionarily in the sense that we map requirements on the system decompositions and remedy any inconsistency as the design goes on.

The above mentioned design approach is an architecture-based design approach. Although no consensus on the definition of system architecture has been reached today, we believe architecture is a high-level (i.e., abstract) representation of the logical, temporal and spatial structures of the system. It shows how the system properties in terms of functional and non-

functional quality attributes are determined by its constituent units and their relations. Working concretely with architecture for the system design implies the following issues.

System architectural description

One of the most important research issues around system architecture is the description of the system abstractions, i.e. system architectural description. It is the basis for all design activities, e.g., comprehension, communication quality prediction, trade-off, decision-making, manipulation, modification, reuses, etc.

The most fundamental requirement on the description is *completeness* in the sense that the described information should be sufficient for the purposes. For mechatronics software, the description should at least include the functional and behavioral/temporal aspects of the system. To facilitate the comprehension, the description should be based on the principle of *separation-of-concerns*. It means that the description should be given in multiple hierarchical layers and orthogonal views (e.g., the structural, functional, behavior, and reliability views, etc.).

The research of architectural description for mechatronics systems should focus on elaborating the needed information for the design, as well as integrating and improving legacy modeling techniques.

Software components

In practice, no design proceeds in a purely top-down manner. During the design, the designers have to use knowledge relating to the system implementation. Otherwise, a semantic gap between an architecture and its implementation may arise. For instance, they may consider similar control algorithms and software codes for quality prediction, and available off-the-shelf components (e.g., sensors, microprocessors, communication bus system, real-time operating system, drive units and actuators, etc.) for implementation. Thus, accessing bottom-up information is unavoidable in a top-down design approach and has impacts on the system architecture.

Fundamentally, a software component is a constituent unit of a software system. As the system exists in different hierarchical layers, there are also different classes of software components. For the real-time software, we may consider the software hierarchy runs from the automatic control solutions at the top level to the software implementation at the lower level (e.g., class and object in an Object-Oriented language). The constituent parts in these layers have multiple interrelations established by the adopted automatic control logic (e.g., required control data flow, desired timing and synchronization), the chosen software implementation (e.g., time consumption and shared timing resource), and the shared hardware resources (e.g., I/O, memory, processor). Thus, to support the design and to achieve flexibility and reuse, the interface of a software component should provide sufficient information so that these multiple interrelations can be established properly.

Some answers on how to classify the software components in the system and how to define their logical, structural and temporal contexts, as well as the required implementation resources and services, should be provided by the research.

Architectural principles and styles

During the design, the designers also use the knowledge from the engineering background (i.e., evaluation criteria, and mature solutions on partitioning/structuring, allocating/distributing, scheduling, etc.). Architectural principles and styles formulate the essential features of existing mature solutions of software, which can be compared with the principles and mature solutions in other well-established engineering domains. Under the circumstances, it is easier to reuse the existing solutions, to elicit hidden requirements in time, and to provide early quality predictions for the domain systems. Thus, research efforts are needed to identify and categorize useful architectural principles and styles.

Conclusions

To illustrate the key research issues around system architecture, we have briefly introduced the system characteristics and the architecture based system design approach. The fundamental problems needs to be tackled by the research are: what constitute the system software and how they interact with each other; how the system architectural description should be provided so that comprehension and analyses are facilitated; how to make software engineering more mature and constrained. A summarization of the project status can be found in the project homepage (http://www.damek.kth.se/~chen.html).