

Elevating Design as a First Class Activity in the Development of Software-Intensive Systems

Insup Lee
Department of Computer and Information Science
University of Pennsylvania
lee@cis.upenn.edu

October 1, 2003

Software-intensive systems have been developed traditionally in an ad-hoc manner by practicing engineers and programmers. Knowledge of system behavior and functionality is contained in the mind of the domain-expert designer, and is only imperfectly captured and translated into system products by the engineer and programmer. Design based on mathematical modeling plays a critical role in other engineering disciplines, such as structural engineering. Although much progress in developing formalisms and tools based on them have been made during the last three decades, their use has been limited by the expense (i.e., time, computational resources, and expertise) required to employ them. The complexity of software-intensive systems has increased to a point where it is not possible to envision the development of high-quality software-intensive systems without using such techniques in the future.

We identify several challenges for making designs based on formal methods to become widely used in practice.

Eliciting formal models from informal requirements. The development of most systems starts with informal requirements that specify how the system (consisting of hardware and software) and the user or environment are expected to behave and their interactions. Research is needed to facilitate the elicitation of a design from such informal requirements (with appropriate NLP techniques and tools to facilitate the process). It is important for the elicitation process to be intuitive for the domain experts. Furthermore, the resulting requirements specification should be amenable to analysis and refinement during subsequent stages of the design process. Balancing these aspects is the first challenging problem since the goals are often contradictory. For example, formalisms that make the analysis easier are usually far removed from the problem domain. Furthermore, requirements start as informal description but need to be formalized to support analysis. Another challenging problem is how to facilitate the translation from informal description to formal specifications and to capture assumptions made by domain experts in informal requirements.

Design validation. The design must be verified and validated to ensure it is correct and consistent with its intended purposes. Such analysis can be done on the design artifacts (e.g., state machines, logical properties, etc.). There is much work done on how to validate that models captured in design artifacts are indeed the ones that are intended. Also, design should be used to validate implementation. The requirements, design, and implementation need to be related such that if one can show that the design satisfies the requirements and the implementation conforms to the design, then the implementation meets the requirements.

Multiple uses of design artifacts. One of the reasons why formal methods techniques are not used in practice is that formal models are only used at the design phase, but become useless down the development process. We need to explore ways to reuse the various design artifacts during the other development phases, such as coding and testing phases. The potential promising areas include the use of models for automatic test generation and code generation, as well as the use of logical properties proved at design time for run-time verification. We need to explore how to ensure that investments on design specification and analysis pay off directly in the final product development.

Sharing of design artifacts. We believe that it is easier (in theory) to share design models than code. This is because models are at higher level of abstraction than code, and thus, tied less with target platforms. There has not been much support and effort with open model (*a la* open source) development so far. Such an endeavor should help to elevate design based on formal methods into main-stream activities.

Composable designs. The notion of composition is necessary to deal effectively with designing of large complex systems. Thus, designs should be composable to facilitate reuse and sharing, as well as just to be able to put together a complex systems using simpler components. The composition of design can be either for homogeneous models or for heterogeneous models. Traditionally, formal method research has concentrated in composing specifications in the same modeling language or paradigm. To elevate the design to a high level, it is important that we start investigating how to compose designs with different purposes (e.g., one design model for the physical layout of a sensor network and another design model for the protocol used between sensor nodes) to determine interaction between different views and to understand the overall design.

Certification based on design. To be able to evaluate the quality of software-intensive systems, we need to develop a certification process based on sound scientific foundations. The certification can be done in two steps: first certify that a design has the right properties, and then, certify that an implementation conforms to the design. With proper scientific foundations, it should be possible to measure quantitatively how well a system meets its requirements.

To improve the quality of software-intensive systems, it is necessary to develop better design techniques and paradigms, where designs are amenable to formal modeling and analysis. We have described several challenging issues for supporting such design paradigms.

Insup Lee is Professor in the Department of Computer and Information Science at the University of Pennsylvania. His research interests include, embedded systems, real-time computing, formal methods, wireless network, and software engineering. He has developed programming concepts, language constructs, and operating systems for real-time systems. In recent years, he has developed specification, analysis, and testing techniques based on real-time process algebra (ACSR). In addition, he has developed a hierarchical specification language for hybrid systems (CHARON). Based on CHARON, he is currently developing techniques for automatic code generation and test generation. Furthermore, he is currently working in wireless network, especially power-aware protocols and security. He also has been developing the run-time monitoring and checking framework (MaC) that can be used to assure the correctness of a running system through monitoring and checking of safety and QoS properties. The prototype MaC system has been implemented in Java and is currently being ported to Real-Time Java.