Enhancing Dependability of Medical Software Systems

Oleg Sokolsky
Department of Computer and Information Science
University of Pennsylvania
Philadelpia, PA 19104-6389

April 15, 2005

Challenges to dependable design of medical systems. Medical software-based systems represent an important class of embedded systems. Along with many other classes of safety-critical embedded systems, such as avionics and automotive controllers, manufacturing, and mobile communication systems, medical systems are facing a number of challenges to their design process.

There are two major factors that complicate the design and implementation of software-based medical systems. First, the software complexity of medical systems has been increasing steadily as microprocessors become more powerful and demands on them grow along with the expectations of their capabilities. To mitigate the development cost of software, embedded systems are being designed to flexibly adapt to different environments. The requirements for increased functionality and adaptability make the development of embedded software complex and error-prone. Second, medical devices, like all other kinds of embedded systems are increasingly networked to improve functionality, reliability and maintainability. Networking makes embedded software even more difficult to develop, since composition and abstraction principles are poorly understood.

Modern software engineering methods that are increasingly applied in other application domains go a long way towards improving quality of software-centric systems. These methods are often based on modeling. Such modeling languages as the Unified Modeling Language (UML) and such development methodologies as Model-Driven Architecture (MDA) provide early discovery of design problems, help to ensure that independently developed components work well together, etc. Embedded systems benefit from model-based approaches as well. However, two features of medical systems, and embedded systems in general, stand in the way of realizing the full potential of model-driven development. On the one hand, correctness requirements for medical systems often include detailed behavioral requirements. Therefore, commonly used modeling languages, which tend to concentrate on structural properties of the system, fall shorts of the design needs. To make things even more complex, medical devices work in complicated, dynamically changing environments. Without adequate modeling of the environment, reliable model-based design of medical systems is impossible.

Second, the safety-critical nature of medical embedded devices requires a higher degree of validation than most regular software engineering processes provide. Therefore, verification and validation need to be at the core of the modeling technology for medical software systems design.

Therefore, we need to develop means of behavioral modeling for medical systems that would naturally support behavioral specifications and enable rigorous verification, and incorporate these methods into existing design processes. The quest for such modeling techniques and practices is probably the biggest challenge to the design of software-based medical systems.

Enabling technologies for rigorous medical system development. Verification of even the *design model* of complex embedded systems is intractable and furthermore, does not provide any guarantee that the *implementation* is correct. Model-based testing is a useful technique but does not provide guarantees about the correctness of the system. A third approach, model-based run-time verification has recently gained significant attention.

Run-time verification is applied directly to a running implementation and hence does not suffer from the drawback of analysis techniques which work on a design model of a system. In addition, because the goal of run-time verification is only to certify the current run of a system, this methodology does not suffer from the combinatorial explosion problem faced by approaches that attempt to exhaustively test a system. Methodology and tools for the run-time verification of software systems are being actively developed.

Medical systems must be designed to interact with complex physical processes in the human body, where some parameters of the system change continuously. In other words, they exhibit hybrid behaviors that are characterized by dynamics that are continuous and by mode transitions that are discrete. Hybrid systems combine continuous dynamics that have been well-studied in control theory and continuous mathematics with discrete communicating systems that have been extensively studied in computer science. We can use such a hybrid system model against which we will check the run-time behavior of our system. This presents challenges in determining the frequency at which to observe continuous variables, interpolating such variable values between observations, etc. In addition, physical systems develop faults and fail over time. We can use such fault and failure models in our hybrid model which will then have normal modes as well as various fault modes. The run-time monitor will be able to identify the mode the system is in and initiate corrective action (steering) when necessary.

We can also use run-time monitoring to diagnose the source of errors in the behavior of the system. A typical medical system will have many distinct components — sensors, actuators, physical devices, and software system, — and we would like to get a more precise diagnosis in monitoring such systems. In order to do this, we will need to introduce some fault-tolerance capability into the system of sensors and actuators deployed and perform checks on individual units against the aggregate.

Current research on dependability of medical software systems. Several directions of the current research in our group at the University of Pennsylvania seek to address the needs of dependable design of medical software systems. One such direction is the design of modeling languages for hybrid systems and tools that support development and analysis of models in these languages. In particular, we have defined a language for hierarchical hybrid systems called CHARON [2, 3]. CHARON supports state-of-the-art modeling concepts such as encapsulation, reuse, preemption, and hierarchy. We also implemented a toolset that can be used to develop models, perform their simulation, and apply formal verification to explore their properties [1]. The work on the CHARON toolset continues with the development of additional analysis techniques, in particular tools for test generation.

Another direction deals with the development of techniques for run-time verification of software-based systems. We have implemented a tool for monitoring and checking of complex systems, called MaC [4]. We use a formal language to specify behavioral requirements of a system. The tool then performs automatic instrumentation of the system to extract the necessary information. At run time, MaC observes the stream of events and checks that the observed behavior satisfies the requirements. We have applied MaC to validation of embedded systems [5]. However, in order to make MaC more useful in the design of medical software systems, we currently are exploring extensions of MaC that would allow us to monitor hybrid behaviors, and use hybrid systems models, such as CHARON models, as behavioral requirement specifications.

References

[1] R. Alur, T. Dang, J. M. Esposito, Y. Hur, F. Ivancic, V. Kumar, I. Lee, P. Mishra, G. J. Pappas, and O. Sokolsky. Hierarchical modeling and analysis of embedded systems. *Proceedings of the IEEE*, 91(1):11–28, Jan. 2003.

[2] R. Alur, R. Grosu, Y. Hur, V. Kumar, and I. Lee. Modular specifications of hybrid systems in CHARON. In *Proceedings of Hybrid Systems: Computation and Control, Third International Workshop*, volume 1790 of *LNCS*, pages 6–19. Springer-Verlag, 2000.

[3] R. Alur, R. Grosu, I. Lee, and O. Sokolsky. Compositional refinement for hierarchical hybrid systems. In *Proceedings of Hybrid Systems: Computation and Control, Fourth International Workshop*, March 2001.

[4] M. Kim, S. Kannan, I. Lee, O. Sokolsky, and M. Viswanathan. Java-MaC: a run-time assurance approach for Java programs. *Formal Methods in Systems Design*, 24(2):129–155, March 2004.

[5] M. Kim, I. Lee, U. Sammapun, J. Shin, and O. Sokolsky. Monitoring, checking, and steering of real-time systems. In *2^{nd} Workshop on Run-time Verification*, July 2002.