Making Plug and Play Safe for Networked Medical Devices

Christopher D. Gill
cdgill@cse.wustl.edu
Center for Distributed Object Computing
Department of Computer Science and Engineering, Washington University, St. Louis, MO

Abstract

Building "plug-and-play" systems in which multiple devices can be integrated on the fly, and new devices can be added, removed, and modified dynamically during system operation, offers significant potential benefits for the medical community. However, due to the critical health and safety issues associated with medical devices, it is essential to maintain high confidence in the correct functioning of these systems, even as a running system of interoperating devices is reconfigured dynamically.

To achieve high confidence for these systems, especially when system correctness involves (1) ordering and timing constraints on system events, and (2) security of system data and actions, a new branch of systems research is needed. This paper proposes a research agenda to establish a foundational set of formally verified systems mechanisms, and formal tools through which they can be combined to support high-assurance interoperation of medical devices in the context of plug-and-play systems.

1 Introduction

Medical devices are rigorously designed to perform specific tasks, but to integrate them effectively, particularly for seamless plug-and-play interoperation among multiple devices, requires significant additional system engineering. Collecting data from a set of devices and displaying it *post hoc* in a common format is already readily achievable by commercially available laboratory information management systems. However, this level of integration falls well short of what is possible if outputs of some devices were connected to inputs of other devices *in real-time with rigorous assurances of timing and security of operation*. Furthermore, to achieve the flexibility and scale of integration needed, software must play a central role not only in the individual devices, but in their end-to-end integration.

Middleware, software that bridges multiple devices in a distributed system, is playing an increasingly important role across a variety of application domains. Significant research over the past decade has made middleware both modular and customizable through the development of state-of-the art object frameworks [1] and QoS-enabled component middleware [2]. Furthermore, to address the complexities raised by timing and other constraints outside the data and computational requirements of a system, research has focused on applying model-driven development techniques [3, 4] to middleware [5, 6], to allow *a priori* modeling and analysis of system properties.

Challenges: The three most important challenges that must be addressed to realize the benefits of middleware in the context of dynamic plug-and-play medical device systems with stringent timing and security constraints, are: (1) common low-level mechanisms from which a wide range of middleware architectures can be built, must be modeled using formal and rigorous techniques at a much finer granularity than has been done previously; (2) tools for synthesis of high-assurance middleware must be developed, which can perform analysis of those formal mechanism-level models, and then compose and configure middleware mechanisms themselves based on that analysis; (3) these tools and their constituent analysis techniques must be made applicable throughout the lifecycle of plug-and-play medical device systems, so that a suitable balance of timing, adaptability, and rigor can produce systems that are both high-performance and high-assurance. We call this approach "composable model-driven middleware".

Research needs: The three most important research needs to address the challenges described above are: (1) suitable mechanism-level abstractions must be selected and modeled, based on their suitability for a particular middleware domain and its associated timing, concurrency, security, and other constraints – we are working on this need in collaborative research efforts described in Sections 2 and 3; (2) tools, including libraries of models and the mechanisms they represent, must be designed and developed, verified via rigorous experimentation with realistic systems, and integrated with other existing development tools and methods, to promote transition of composable model-driven techniques into mainstream engineering practice for developing systems of medical devices; (3) the time and space costs of model checking must be addressed, particularly through hybridization of model checking with other analysis techniques, to allow analysis on-line at run-time as devices are added, removed, and modified dynamically.

Roadmap: A variety of middleware and operating system abstractions are already widely used in the development of distributed real-time and embedded systems. Model-driven tools for middleware configuration and analysis have also been prototyped in research settings. In the near term, two main research efforts are needed to begin to apply these existing capabilities to the domain of plug-and-play medical devices: (1) existing model-driven middleware tools should be augmented with our fine-grained composable model-driven middleware techniques to support detailed analysis of concurrency, timing, and security properties, and highly customized synthesis of exactly (and possibly only) the mechanisms needed by each application; (2) these augmented tools must be evaluated rigorously and empirically for developing realistic systems of medical devices, though in the near term the context of evaluation should be limited to fairly static combinations of devices. Longer term research, *conducted in collaboration between the theory and systems research communities* is needed to address the hard problems of performing model analysis within scales of time and space that match the

constraints imposed by dynamic addition, removal, and modification of medical devices at the time scales needed in various medical contexts. One such collaboration is described in Section 2, in which incorporating focused protocols whose properties can be proven formally has the potential to reduce the state space that must be explored through model checking.

We now describe two specific areas of concern for plug-and-play medical device systems, (1) the assurance of timing properties and (2) the assurance of security for system data and actions, to help illustrate some of the challenges and potential solutions for composable model-driven middleware. Section 2 describes work on timing assurance being conducted in collaboration with Venkita Subramonian at Washington University, and Cesar Sanchez, Henny Sipma, and Zohar Manna at Stanford University, in which we are developing fine-grain models of composable middleware mechanisms and evaluating composed models through model checking and other principled analysis techniques. Section 3 describes work on integrated security of system data and actions being conducted in collaboration with Armando Migliaccio from the Universita degli Studi di Napoli Federico II (currently a visiting scholar at Washington University), Douglas Niehaus at the University of Kansas, and Ravi Sandhu at George Mason University, in which well-known techniques for access control can be integrated with new approaches to scheduling and with the techniques described in Section 2, to facilitate assurance of security of system data and actions.

2 Timing Assurance

Formal models have been used traditionally to uncover flaws in application design early in system development. However, to ensure those models continue to reflect the system being built these application-level models must then be elaborated and refined throughout the entire system development lifecycle. For example, as decisions regarding (1) the deployment of application components, (2) the topology of dependencies between components and devices, and (3) settings for policies and mechanisms at the middleware level are made, these decisions must also be modeled and the complete model re-evaluated for adherence to design constraints as each such decision is made.

Fine-grain models of middleware elements: We focus our modeling efforts on middleware abstractions that are sufficiently fine-grained for both (1) rigorous concurrency and timing analysis, and (2) widely used across different middleware implementations and configurations. The choice of abstractions to model also has implications for the cost and scalability of analysis. We are currently modeling canonical abstractions from ACE [1], as a suitable basis for building middleware.

Figure 1 illustrates a scenario in which application components are deployed across two interoperating devices. Each component is implemented as an event handler (EH1, EH2, and EH3). On each device, system events are dispatched to handlers by a "reactor", which is a middleware mechanism for dispatching events arriving from multiple sources. Client components external to the devices (for example, on PDAs used by healthcare personnel) can interact with the components on each device by sending events through its reactor.

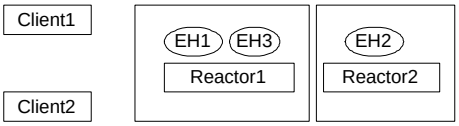

Figure 1. Component Deployment

Rigorous analysis of composable models: We have selected timed automata [7] as the formalism with which to build models of canonical ACE abstractions. We then use model-checking techniques [8] to detect potential constraint violations or deadlocks. Model checking tools such as UPPAAL [9] and IF-toolkit [10] are available for development, simulation, and analysis of timed automata models.

We now illustrate how timing properties of the system are affected by even such simple choices as the deployment of components and their interdependencies. Consider a scenario based on Figure 1, in which Client1 sends an event to event handler EH1 and Client2 sends an event to event handler EH3. Assume EH1 and EH3 must each perform a specific computation when invoked, and each return a result to the source of the event by a given relative deadline as illustrated in Figure 2. Whether or not the relative deadlines can be met depends on a variety of factors including the relative execution times and deadlines for the event handlers, the the relative order in which the reactor is allowed to dispatch events to them, and possibly other factors such as multi-threading within a reactor or thread scheduling policies enforced by the middleware and operating system.

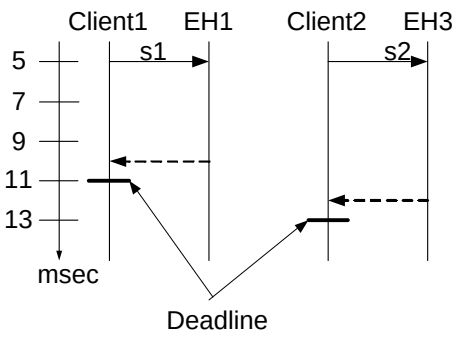

Figure 2. Client/Handler Interactions

Our models are designed to reflect such factors and to allow potential interactions between devices to be explored to determine conditions under which constraints may be violated, or to verify that the system as modeled is free of constraint violations. For example, we can analyze whether there are any deadline misses in the scenario above by checking temporal logic expressions in UPPAAL (*e.g.,* E◇ Client1.DeadlineMiss or Client2.DeadlineMiss – that is, is there any state of the system where the Client1 automaton is in its DeadlineMiss state or the Client2 automaton is in its DeadlineMiss state?).

If an expression evaluates to true, then the model checker can show a trace of the sequence of automaton transitions that led to the system state in which the expression became true. For example, if the reactor could dispatch events concurrently to EH1 and EH3, and just meet their deadlines, but when EH2 (on a completely separate device) was invoked it in turn also sent an event to EH1, a deadline miss in Client1 could occur if Client1 sent an event to EH1 just after EH2.

Reducing the cost of analysis: Tools such as IF-toolkit which allow dynamic introduction and removal of automata from the model also help to prune the size of the state space that must be checked at any point in the system lifecycle, compared to tools like UPPAAL that require all automata to be composed statically and thus require additional modeling (and thus checking) to emulate dynamic addition and removal of automata. Also, hybrid analysis techniques that apply other formalisms can reduce the state space that must be explored by model checking. For example, we have modeled deadlock avoidance protocols [11]. Whenever a request comes in to the reactor, the protocol determines whether or not a thread can be allocated to the incoming request using information about the dependency graph – if not the request must be delayed until allocating a thread to it would not risk deadlock. Although a formal proof that this protocol prevents deadlock [11] relieves the model checker of the need to test for deadlock, the protocol itself adds new sources of delay and thus must be modeled to check for deadline misses.

3 Security Assurance

Despite the importance of security of *both* system data and behavior, traditional system development approaches treat *access control* and *execution control* as separate areas of specification and enforcement, resulting in (1) rigid architectural boundaries across which it is difficult to integrate security policies and mechanisms, and (2) unanticipated "loopholes" that can weaken or even bypass security enforcement.

To assure security properties within the composable model-driven middleware approach described in Section 2, we have developed a novel security approach based on *integrated access, admission, and execution control*. Access control can increase security of system data by ensuring that event handlers must have appropriate permissions to be able read, remove, transmit, or modify any datum. However, for medical devices access control is only the first of several control decisions that must be made to ensure secure *execution* of those handlers. For example, once permission to send an event to a handler is granted, an *admission* decision must be made so that execution of the handler is not initiated if it will result in a violation of timing or other constraints. The run-time *scheduling* [12, 13] of handlers completes the sequence of decisions needed to ensure secure operation. Scheduling may enforce complex systems of timing and ordering constraints, for example to meet relative deadline [14] or progress [15] requirements.

Our approach is to extend the well-established role-based access control (RBAC) security model to include enforcement of admission and execution control decisions. As in the RBAC model, a role associates a set of permissions and other control-related data with a *group* of users so that the permissions for a group remain stable while users are allowed to join and leave groups dynamically. However, each role is then extended to include its own chain of decision functions for access, admission, and execution control, as illustrated in Figure 3.

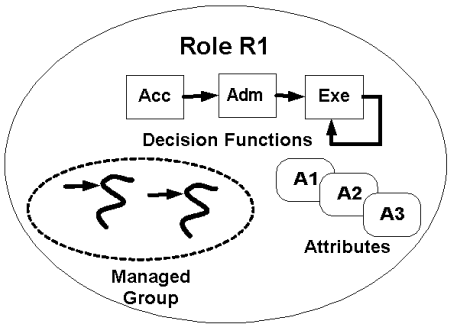

Figure 3. Integrated Role Structure

References

[1] D. C. Schmidt, "An Architectural Overview of the ACE Framework: A Case-study of Successful Cross-platform Systems Software Reuse," *;login:*, Nov. 1998.

[2] N. Wang, C. Gill, D. C. Schmidt, and V. Subramonian, "Configuring Real-time Aspects in Component Middleware," in *Proceedings of the International Symposium on Distributed Objects and Applications (DOA'04)*, (Agia Napa, Cyprus), pp. 1520–1537, Oct. 2004.

[3] J. Sztipanovits and G. Karsai, "Model-Integrated Computing," *IEEE Computer*, vol. 30, pp. 110–112, Apr. 1997.

[4] J. Liu, X. Liu, and E. A. Lee, "Modeling Distributed Hybrid Systems in Ptolemy II," in *Proceedings of the American Control Conference*, June 2001.

[5] J. Hatcliff, W. Deng, M. Dwyer, G. Jung, and V. Prasad, "Cadena: An Integrated Development, Analysis, and Verification Environment for Component-based Systems," in *Proceedings of the 25th International Conference on Software Engineering*, (Portland, OR), May 2003.

[6] G. Madl, S. Abdelwahed, and G. Karsai, "Automatic Verification of Component-Based Real-Time CORBA Applications," in *Proceedings of the 25th IEEE International Real-time Systems Symposium (RTSS 2004)*, (Lisbon, Portugal), IEEE, Dec. 2004.

[7] R. Alur and D. L. Dill, "A theory of timed automata," *Theoretical Computer Science*, vol. 126, no. 2, pp. 183–235, 1994.

[8] J. Edmund M. Clarke, O. Grumberg, and D. A. Peled, *Model checking*. MIT Press, 1999.

[9] G. Behrmann, A. David, and K. G. Larsen, "A tutorial on UPPAAL," in *Formal Methods for the Design of Real-Time Systems*, no. 3185 in LNCS, pp. 200–236, Springer–Verlag, 2004.

[10] M. Bozga, S. Graf, I. Ober, I. Ober, and J. Sifakis, "The IF Toolset," in *Formal Methods for the Design of Real-Time Systems*, Springer-Verlag LNCS 3185, 2004.

[11] C. Sanchez, H. B. Sipma, V. Subramonian, C. Gill, and Z. Manna, "Thread allocation protocols for distributed real-time and embedded systems," in *Submitted to FORTE 2005*, oct 2005.

[12] C. D. Gill, D. L. Levine, and D. C. Schmidt, "The Design and Performance of a Real-Time CORBA Scheduling Service," *Real-Time Systems, The International Journal of Time-Critical Computing Systems, special issue on Real-Time Middleware*, vol. 20, Mar. 2001.

[13] C. Gill, D. C. Schmidt, and R. Cytron, "Multi-Paradigm Scheduling for Distributed Real-Time Embedded Computing," *IEEE Proceedings, Special Issue on Modeling and Design of Embedded Software*, vol. 91, Jan. 2003.

[14] R. Gerber, W. Pugh, and M. Saksena, "Parametric Dispatching of Hard Real-Time Tasks," *IEEE Transactions on Computers*, vol. 44, Mar. 1995.

[15] T. Aswathanarayana, V. Subramonian, D. Niehaus, and C. Gill, "Design and performance of configurable endsystem scheduling mechanisms," in *Proceedings of 11th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2005.