

Software Certification for Distributed, Adaptable Medical Systems: Position Paper on Challenges and Paths Forward

Kurt Rohloff, Richard Schantz, Partha Pal, Joseph Loyall
BBN Technologies
Cambridge, MA, USA
{krohloff, schantz, ppal, jloyall}@bbn.com

Elements of previously vetted architectural constructs, design principles and algorithms, along with static and dynamic analysis, simulation, testing and instrumentation/logging have all historically contributed to certification arguments for safety-critical medical systems. Although certification arguments based on these aspects are appropriate and have been sufficient in the past, especially for previous monolithic, non-adapting software systems, these aspects of certification arguments will have to be updated in order to generate economically feasible approaches to certification arguments for advanced, distributed adaptive system architectures such as Plug and Play (PnP), multi-layer Quality of Service (QoS) management, peer-to-peer behavior and ad-hoc distributed interaction strategies for reconfigurable topologies. Updated certification arguments need to take into account system-wide, multi time-scale event structures with time-critical operations, the aggregation of synchronous and asynchronous computation systems, and multi-platform, distributed resource management issues in a cost-effective manner. Furthermore, for the pervasive, wide-scale architectures listed above, life-cycle issues need to be addressed so that beyond initial certification, the systems can be easily and inexpensively recertifiable as they are modified and adapted as system requirements that evolve over both very short and very large time scales (possibly years or decades). Recertification must be possible without incurring large additional costs, with the goal of being able to recertify on the fly for PnP operation.

In this position paper, we propose an analysis, architecture and design approach to specify and enforce certifiable behavior as a means for meaningful and economically feasible certification argument construction in the context of distributed, adaptable safety-critical software systems. The main components of our approach are:

1. Methods to identify and separate uncertifiable behavior based on system observables.
2. Extending interface standards to complement certification activities.
3. Methods for regulating component interaction.
4. Methods to dynamically, constrain behavior into localized, certifiable operating regions.

Additionally, to have maximum impact, future work on the analysis, architecture and design of certifiable distributed, adaptable medical systems must intelligently link augmentations of each of these analysis, architecture and design approaches with the traditional certification evidence such as simulation, testing and instrumentation/logging into one unified methodology capable of being easily adopted by governmental and industrial regulation agencies to enhance certification standards.

Our thoughts on approaches to design for certifiable, distributed, PnP and adaptable medical systems is informed by our experiences in designing, developing, building and fielding numerous highly adaptable prototype distributed software systems in diverse application areas. Several examples of relevant programs for which we recently developed adaptable distributed software system include the ARMS and DPASA programs funded by DARPA and the ICED program funded by the AFRL. For ARMS we developed an adaptive and reflective middleware system that manages the resource allocation and fault recovery of distributed computation processes. As part of the DPASA program we developed an adaptive survivability architecture to improve the resiliency and tolerance of distributed information systems against cyber attacks performed by a malicious adversary. For the ICED program we developed a QoS management system for dynamic information sharing environments.

Identify “Good” and “Bad” Behavior

As an initial step on the path towards the certifiability and acceptance of adaptive, distributed, real-time medical systems, one needs to be able to separate “good” dynamic behavior from “bad” dynamic behavior from a certifiability point of view. System adaptation is not always appropriate and there may be times when otherwise appropriate dynamic behavior is inappropriate. The certification of dynamic systems will need to establish the terms of evaluation for dynamic operation, and will need to establish that both “good” behaviors happen, while “bad” behaviors don’t, to a high degree of plausibility.

Beyond a simple binary analysis of “good” and “bad” appropriate for static systems, there is a need in dynamic systems to be able to express how “good” and “bad” evolves over time – for example mild cardiovascular stress may be appropriate during periods of light exertion, but cardiovascular stress is never appropriate when the patient is resting. This illustrates the need for time-varying expressions for the utility of dynamic system performance. These expressions for the utility of system performance could vary from simple rankings of observed system behavior to more complex expressions for evaluating the timing of conditional occurrences and orderings of system behaviors.

The more complex approaches to evaluating system behavior based on the timing and ordering of system events may be thought of similarly to formal logics classically used to evaluate system correctness. However, our proposed methods are intended to be more easily applicable in that they are intended to be driven by direct observations of system behavior rather than by the inferencing of underlying, unobserved system behavior that may not be directly related to system observables.

Based on a new, experimental approach for distributed systems behavior evaluation, a major research and development step in the fielding of distributed, certifiable real-time system would be the ability to automatically generate code based on expressions of the suitability of system behavior based on derived, experimental understanding of system behavior. This automatically generated code and/or the experimental models could be used as an additional input to our methodologies for component interaction control.

Interface Standards

Today, to our knowledge, there are no meaningful, scalable techniques to certify a whole, composed network centric system, even when it is composed of certified parts using PnP interactions. This is due, in large part, to the fact that two composed elements, even when their independent behaviors are certified, might functionally interact in ways that cause one or both of them to exhibit unpredictable emergent behavior that violates their “certified” behaviors. A simple example is that of two high priority real-time components, each of which independently are shown to meet their real-time deadlines in isolation. When composed, they utilize shared computation and network resources needed by the other, in such a way that they can affect each other’s ability to meet its deadline.

A first step to managing component interactions for adaptable, distributed systems is the enforced use of interface standards, such as those for objects and components. This approach would impose some (limited) rigor on the functional interactions between elements. As long as system developers limit their development of independent elements to components or objects with well defined interfaces, and compose the system only through those interfaces, then some properties of the component interaction can be identified. These functional interfaces are insufficient for reasoning-based certification of our dynamic, distributed systems because components also interact through shared access to resources. Combined with dynamic resource management, these interactions are most often quite complex, difficult to completely and adequately specify, and even more difficult to certify.

An approach to developing advanced interface standards is thru methods to express the constraints/relations associated with component interaction. The constraints/relations/obligations (i.e., can do, cannot do, should do, must do or else-- so on and so forth) and their enforcement (static and dynamic) transcends the representation of system interaction. One step to expressing the constraints and relations for interface standards would be to develop an (or specialize an existing) extensible markup language for safety-critical component behaviors and interactions.

Beyond creating interface standards, an approach to certifiable composition is to create well-defined QoS, or resource, interfaces through a middleware system and program only to within the strict limits imposed by them, analogous to using functional interfaces. Such an interface would identify the resources used by a component (the obvious ones would be CPU and network, but other interfaces such as shared displays, memory footprints, etc. could also be considered) and identify the change in expectations as well as the behaviors of the component under various ranges of resource availability.

Component Interaction Control

Provisioning the resource management functionality as common middleware infrastructure enables us to more uniformly and more certifiably *control* the resources provided through these interfaces. Early on, we can use clearly partitioning resource allocation mechanisms, such as CPU reservations and network reservations (if available). These enable the dynamic resource manager to provide a clear set of

resources to each component (and component interaction), simplifying the reasoning and certification. A next step, which introduces more complexity to the interfaces and analysis, might look at priority based resource provisioning. Even there, priority lanes and admission control help bound the problem space.

A more general problem involves removing these constraint mechanism interfaces in favor of policy-driven application control that would automatically regulate component interaction and avoid system behavior that may be uncertified and/or uncertifiable. A promising idea in this space, related to our previously mentioned approach to dynamic system evaluation, is to use specialized, domain-specific, mutually composable sequential processes to express the certified behaviors of individual components of the system. The union of shared behavior in these processes would need to be able to interface with the behavior of the overall system (with respect to both functional behavior and resource usage), but would also need to be sufficiently easy to compose and perform computable operations with respect to the certifiability of the behavior of the overall system and evaluate experimentally.

Dynamic, Regulated Operation in Certifiable Configurations

An additional avenue to “on-the-fly” certification of highly reconfigurable medical systems would be to take an incremental online approach to system regulation that permits “certifiable” behavior to occur and facilitates system recovery if unforeseen events (such as partial system failures) occur that cause the system to enter an uncertified operating mode. For this approach, based on our previously mentioned approach to experimentally evaluating behavior, a regulator in the system could be continually run to identify “acceptable” and “unacceptable” variations of the system’s current configuration (based on some measure of utility). The regulator would attempt to prevent the system from entering into an “unacceptable” configuration and could be used as a kind of “fail-safe” governor for system behavior. If the system would ever exhibit “unacceptable” behavior due to partial system failure or the use of the system in an unproscribed manner, the governor would push the system to return into an acceptable operation configuration, or at least prevent the system from entering an unacceptable operation configuration.

As this procedure progresses, we would not automatically get full, continuous certification of the configuration space, but we would get directed coverage of large numbers of static certified configurations over the areas of highest interest/utility. Operation would be prevented from getting worse if things go wrong (and things always go wrong in complex systems over sufficiently large timeframes). This certification process could run in the background when the software is deployed. During operation, the software's configuration controller would only reconfigure to use configurations that have been designated as certified