An Open Test Bed for Medical Device Integration and Coordination*

Andrew King, Sam Procter†
Dan Andresen, John Hatcliff‡, Steve Warren
Kansas State University
{aking,samuel3,dan,hatcliff,swarren}@ksu.edu

William Spees, Raoul Jetley
Paul Jones, Sandy Weininger
US Food & Drug Administration
{William.Spees,Raoul.Jetley,PaulL.Jones,
Sandy.Weininger}@fda.hhs.gov

Abstract

Medical devices historically have been monolithic units – developed, validated, and approved by regulatory authorities as stand-alone entities. Modern medical devices increasingly incorporate connectivity mechanisms that offer the potential to stream device data into electronic health records, integrate information from multiple devices into single customizable displays, and coordinate the actions of groups of cooperating devices to realize "closed loop" scenarios and automate clinical workflows. However, it is not clear what middleware and integration architectures may be best suited for these possibly numerous scenarios. More troubling, current verification and validation techniques used in the device industry are not targeted to assuring groups of integrated devices, and regulatory regimes have not yet been developed that allow manufacturers to bring systems of cooperating devices (each approved individually beforehand) to market.

In this paper, we propose a publish-subscribe architecture for medical device integration based on the Java Messaging Service, and we report on our experience with this architecture in multiple scenarios that we believe represent the types of deployments that will benefit from rapid device integration. This implementation and the experiments presented in this paper are offered as an open test bed for exploring a multitude of development, quality assurance, and regulatory issues related to medical device coordination.

1 Introduction

Most early medical devices were manually operated and incorporated electrical and mechanical control functionality. As medical-device complexity increased, analog circuitry was replaced or supplemented with digital technology, driving an increased reliance on embedded and high-level software for the implementation of critical device functionality. For example, early pacemakers included around 10,000 lines of code. Today, a modern pacemaker will host over 100,000 lines of code. Despite the increasing focus on medical device software, much of this code is still developed by engineers whose primary training is geared toward low-level hardware and firmware development – few are trained in state of the art software development technologies or formal quality assurance techniques. Conversely, challenges of medical device development have not been sufficiently exposed in the software engineering community, leading to a situation where innovations in software engineering and validation techniques are slow to be incorporated in medical device development. Additionally, device regulation approaches have been rigid by nature to promote patient safety, so regulation is naturally outpaced by rapid advances in software technology that can be quickly adopted as functional improvements in generic consumer electronics, broadening the gap between what is expected from devices and what can be realistically validated.

Historically, medical devices have been developed as monolithic stand-alone units. Current Verification and Validation (V&V) techniques used in industry primarily target single monolithic systems. Moreover, FDAs regulatory regimes are designed to approve single stand-alone devices. Currently, there are no guidelines in place for how the industry might bring to market a framework that provides new and extensible clinical functionality (in essence, creating "virtual devices") by utilizing an open system of cooperating medical device components from different vendors. This is driven by uncertainty regarding how one might regulate device collections when the full suite of device-device interactions is not fully known *a priori* [21, 20].

This state of affairs stands in direct contrast to the pervasive integration of and cooperation among computing devices in our world today, and it is quite clear that numerous clinical motivations exist to deploy systems of integrated and cooperating medical devices. Many devices marketed today already include some form of connectivity (serial ports, Ethernet, 802.11 or Bluetooth wireless) – typically used to unidirectionally log data/events from these devices. However, it is anticipated that medical systems will undergo a paradigm shift as device integration moves well beyond simple connectivity to provide functionality such as device

*SAnToS Tech Report 2008-02. *Submitted for publication.* This material is based upon work supported by the National Science Foundation under Grants # 0454348 and 0734204 and by the Air Force Office of Scientific Research.

†Author's current affiliation: University of Nebraska, Lincoln
‡Corresponding Author.

data streaming directly into patient electronic health records (EHRs), integration of information from multiple devices in a clinical context (*e.g.*, hospital room) into a single tailorable composite display, and automation of clinical workflows via computer systems that control networks of devices as they perform cooperative tasks. Indeed, companies including Cerner, CapsuleTech, Philips/Emergin, Sensitron, and iSirona are bringing to market infrastructure that facilitates streaming of device data into medical records. In addition, large-scale research projects such as the Medical Device "Plug and Play" Interoperability Program [13], funded by the U.S. Army's Telemedicine and Advanced Technology Research Center (TATRC), are developing standards and prototypes for systems of cooperating devices.

In summary, the technology exists to assemble many of types of medical systems that can substantially improve health care quality while lowering costs of medical care. It is, in fact, so easy to establish *ad hoc* networks and integrate devices that companies are rapidly pushing integration solutions into market, and increasing numbers of clinical technicians are "rolling their own" device networks. This state-of-affairs is dangerous because the V & V technology and regulatory processes to guarantee the safety and security of these systems is lacking.

Progress must be made on a number of fronts to address the challenges described above.

- Which middleware and integration architectures are candidates to support device integration across multiple interaction scenarios?
- Which programming models are suitable for rapid development, validation, and certification of systems of interacting medical devices?
- What V & V techniques are appropriate for compositional verification of envisioned medical systems, and how can the effectiveness of the techniques be demonstrated so as to encourage adoption among commercial vendors?
- Can existing regulatory guidelines and device approval processes that target single devices be (a) extended to accommodate component-wise approval of integrated systems and (b) established in a manner that encourages innovation and rapid transition of new technologies into the market while upholding a mandate of approving safe and effective technologies?
- What interoperability and security standards are necessary to encourage development of commodity markets for devices, displays, EHR databases, and infrastructure that can support low cost deployment of integrate systems and enable flexible technology refresh?

To facilitate industry, academic, and government exploration of these issues, we are developing an open Medical Device Coordination Framework (MDCF) for designing, implementing, verifying, and certifying systems of integrated medical devices. This paper reports on our experience of using publish-subscribe architectures and component-based model-driven development along with standards-compliant open-source code-bases in the initial development of this infrastructure. The specific contributions of this paper are as follows:

- We identify three clinical contexts in which device integration has the potential to be especially effective, and we summarize performance, development, and regulatory requirements of each.
- We propose a flexible publish-subscribe middleware architecture based on the Java Messaging Service (JMS) to support exploration of issues surrounding systems of integrated medical devices.
- We propose a model-based programming environment for rapid development of systems of integrated devices that we believe has the potential to support a new paradigm of regulatory oversight that can accommodate approval of device integration scenarios.
- We describe experiments that evaluate our infrastructure against the data formats, performance requirements, system functionality that we believe are representative of the requirements of envisioned clinical contexts for systems of integrated devices.

We are submitting this paper to the ICSE Experience Track because we seek to (a) directly engage the software engineering community with initial experience and challenge problems associated with this emerging paradigm of medical systems and (b) overcome community barriers that have previously inhibited interactions between the software engineering researchers/practitioners, industrial medical device developers, and government regulatory authorities. The contents of this paper should not be interpreted as an endorsement by the FDA of any particular technology, software infrastructure, or direction for regulatory policy. However, we expect experience with frameworks like the one presented here to provide science-based input to ongoing regulatory policy and standards development efforts. The infrastructure is available for public download at [12].

2 Clinical Contexts

Device integration/coordination can be beneficial in a number of contexts. However, requirements on computational resources, performance, data rates, information availability, safety, and programmability vary greatly across these contexts. In this section, we note the characteristics of three clinical device integration contexts (CDICs) that we believe are especially relevant.

2.1 Room-oriented Device Information Presentation

Concept: A typical hospital room in an intensive care ward hosts a number of stand-alone devices (see Figure 1). Many modern rooms are integrated with an EHR database to log clinical activities, lab data, treatment plans, and information

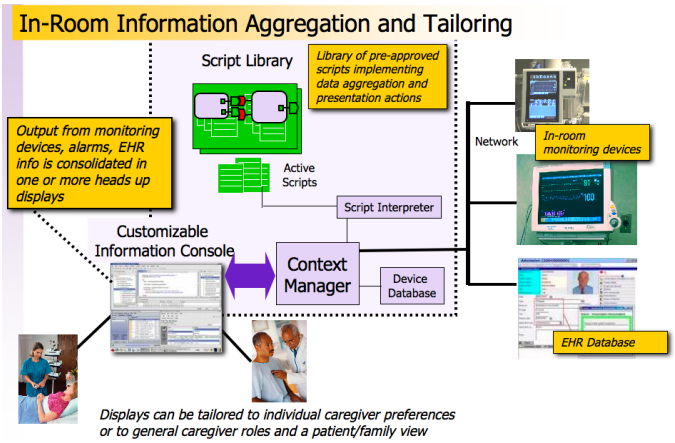

Figure 1. Integrated device display

for patient billing. Connections to drug dosing databases may also be available to facilitate correct drug dispensing.

In such contexts, a number of factors reduce efficiency, degrade the quality of the patient's encounter, and increase error likelihood. Each device in the room has its own user interface, and these interfaces are non-uniform – potentially leading to mental overload and confusion on the part of the caregiver. The devices are physically separated from each other (e.g., on different sides of the bed), and a caregiver must step from one device to another to read device displays or change settings. Different interfaces in different locations make it more difficult for clinicians encountering an alarm or some other critical event to rapidly gain situational awareness. Clinicians with different roles (e.g., the patient's doctor versus a nurse tasked with dispensing prescriptions) are likely to need different "views" or presentations of device information, and such reconfigurations must be carried out manually. Finally, each device provides data necessary to assess a patient's condition, but data are logged separately for each device, are not easily searchable, and do not always become part of the patient's EHR.

Integration Solution: Figure 1 presents a vision of how devices and medical information systems can be integrated to address these issues. Several vendors including Capsule Tech, Cerner, LiveData are now marketing integration frameworks that cover one or more aspects of this vision. In these integrated solutions, traditional medical devices are viewed as data producers that publish periodic or streaming data to several types of data consumers.

An EHR database serving as a data consumer allows information from individual devices to be integrated directly into the EHR. This aggregates device data into one place (simplifying record keeping), facilitates data searches (to speed diagnoses and track device performance), and facilitates discovery and tracking of statistical trends or anomalies that would otherwise go unnoticed by clinicians.

Another type of data consumer would be a single "heads up" display that takes information from multiple devices and an EHR database (in this case, acting as a data producer) and formats it on one or more large monitors near the patient bed. For example, Cerner's recently-released CareAware infrastructure uses IBM's Eclipse framework to aggregate data from multiple devices onto such displays. In CareAware, the Eclipse "view" mechanism is used to define one or more data views from a particular device and even to combine and integrate data coming from multiple devices [2]. The Eclipse "perspective" mechanism is used to define different collections of views that are tailored to the concerns of different caregivers.

Implementation Issues and Requirements: The device integration scenario above must support different data amounts and rates. For example, a pulse oximeter finger clip may yield only heart rate and blood oxygen saturation values that are updated as individual parameters every 10 seconds, yet an electronic stethoscope may need to stream data at 8 kilosamples per second over the network. The architecture must allow easy insertion of "data transformation" components that aggregate, filter, and transform data. The infrastructure must be supported by a programming and validation environment that (a) allows easy definition and composition of data producers, consumers, and transformers and (b) provides facilities for thorough validation and easy auditing so as to support regulatory oversight.

Deployment Strategies: This integration scenario may be implemented using a dedicated server for each room, or by a collection of one or more servers that support integration activities across an entire ward or suite of rooms. Thus, any integration framework that aims to support this CDIC should be evaluated to determine if can give acceptable performance as the number of rooms scales when deployed in a server-for-multiple-rooms configuration.

V&V and Regulatory Issues: When considering the viability of different architectures and deployment strategies, one must first assess conformance to basic safety, performance, and security requirements. In this context, safety and performance issues include the following: Can the infrastructure transfer data from producers to consumers at required rates while avoiding unacceptable latencies and jitter? Are data out-of-range and network babbling scenarios properly anticipated and addressed? Security issues center around the ability of the infrastructure to ensure that private data are unobservable and unalterable by unauthorized parties. In a shared server scenario, the most likely source of safety and security threats would be interference between activities in different rooms supported by the server or other network nodes. For example, heightened activity in one or more rooms might degrade the performance in another room. Data transmitted from one room may be leaked or observed by a party in another room. A server-per-room strategy would seem to have safety and security advantages (*e.g.*, no interference between activities in different rooms)

but would be subject to more difficult and costly maintenance (*e.g.*, more machines to maintain).

As stated in Section 1, almost any device-integration consideration raises a number of regulatory issues due to the fact that devices currently are approved by the FDA as stand-alone units. No claims are made about their potential integration with other devices. Device approval addresses a number of issues including assessment of human factors in the reading and understanding of device displays. Since one of the primary motivations for this integration scenario is to *redisplay* information produced by a device on an integrated display, any such redisplay should be faithful to the precision and presentation of data on the original device. Any filtering, transformation, or integration of data runs the risk of masking important data or alarms from the original device or leading to clinicians interpreting the data redisplay in a manner that differs from that of the original device display. In fact, for transformations that radically alter the original data, it is possible that FDA may interpret such a redisplay as a *new device* (albeit, a virtual device) that would be subject to FDA regulatory oversight.

2.2 Alarm Integration and Forwarding

Concept: Many medical monitoring devices produce alarm signals when device data fall outside of a prespecified range. Proper implementation of device alarms and clinical workflows to monitor/respond to alarms is a significant concern. The IEC 60601-1-8 standard [9] lists a variety of requirements for alarm categories, priorities, auditory and visual alarm signals, and vendor documentation of alarm logic and settings. It also defines the notion of a *distributed alarm system* that forwards alarm information for multiple devices to a central nurse station, database, or paging system.[1]

To combat problems of alarm false positives (the presence of many false positives can cause providers to ignore alarms or disable them), some researchers and vendors are producing "smart alarms" that use some form of sophisticated reasoning (*e.g.*, fuzzy logic) to improve accuracy. Substantially increasing connectivity between devices and health information databases as in Section 2.1 makes it possible to design alarms that consider not just readings from a single device but also readings from multiple devices or accumulated health histories. For example, rather than adopting a "one size fits all" approach to alarm threshold parameter settings, which may lead to false positives/negatives, an automatic threshold setting algorithm could consider the current patient body type, weight, etc. as well as large collections of patient histories to provide more accurate settings. Utilizing multiple device integration, MDPnP has considered simple use cases where alarms are only raised if two different devices (*e.g.*, pulse oximeter and a respiratory monitor) both show alarm conditions [7].

Integration Solution, Implementation, and Deployment: To support open experimentation with these emerging technologies, an integrated distributed alarm and device system must be able to forwarding alarm events that include information about the priority and source of the alarm (room, patient, and device) as well as *information signals* that provide data from monitoring devices (*e.g.*, an ECG waveform indicating ventricular fibrillation). Other requirements are similar to the CDIC in Section 2.1. Transformations on data streams must be easily programmable to support data/alarm correlation from multiple devices and access to health information databases.

2.3 Critical Care Device Coordination

Concept: The above CDICs consider unidirectional flows of information from devices (restricted to *producers* of data) to displays or medical databases. Relaxing this restriction to allow devices to be data/control consumers substantially increases risks but can provide tremendous benefits including increased precision, removal of the potential of human error in tedious and repetitive tasks, reduction of time in time-sensitive tasks, and cost reductions. In system concepts currently envisioned, *e.g.*, by MDPnP, risks and safety concerns are mitigated by having an automated or semi-automated agent control and supervise communication between devices, thus avoiding unexpected interactions that may arise in direct inter-device communications.

Example: An integrated X-ray/ventilator demonstration from MDPnP is an example of useful coordination that addresses the common problem of acquiring chest X-rays of patients on ventilators. To keep the lungs' movements from blurring the image, doctors must manually turn off the ventilator for a few seconds while they acquire the X-ray image, but there are risks in inadvertently leaving the ventilator off for too long. These risks can be minimized by automatically coordinating the actions of the X-ray imaging device and the ventilator: specifically, the ventilator can identify when the lungs are at full inhalation or exhalation (and thus experiencing minimal motion) so that the X-ray image can be automatically captured at the optimal point in time [7].

Various levels of automated feedback mechanisms that close the monitor-assess-treat loop are present in single devices today. Examples include so-called "smart pumps" that use monitored parameters (*e.g.*, blood pressure, heart rate, and blood glucose level), to control fluid infusion (*e.g.*, insulin) into the body. A framework that supports controlled device coordination can expand closed-loop capabilities from functionality hard-coded into single devices up to flexible and extensible functionality that spans multiple devices. Smart pumps have been helpful to assure precision infusions that reduce harmful over- or under-dosing of fluids. However, the executable behavior of smart pumps

[1] Philips/Emergin is a good example of a comprehensive distributed alarm system.

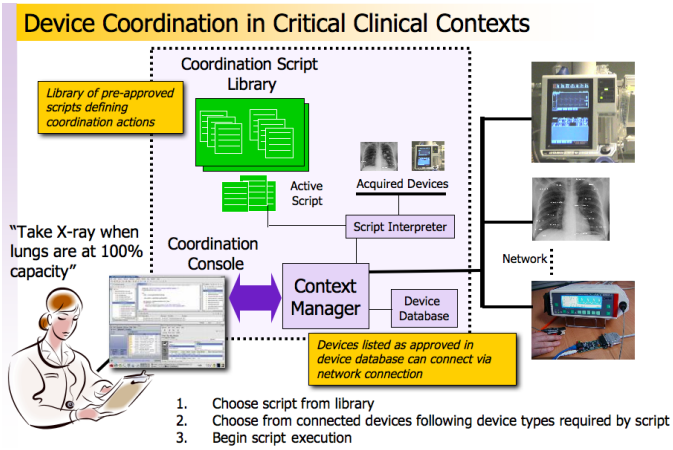

Figure 2. Critical care device coordination

is relatively complex compared to most modern medical devices. Recalls on such devices have occurred due to software-related problems. As closed loop concepts are scaled to multiple devices, frameworks like ours that facilitate open experimentation and assessment are crucial.

Integration Solution: Figure 2 presents a vision of the functional capabilities of a critical care device coordination framework. A collection of network-capable devices are connected to a coordination framework. A device database records the unique identifiers (*e.g.*, based on MAC address) and device drivers for devices that have been pre-approved for connection to the framework. A database of available coordination behaviors holds scripts written by experts that implement approved coordination protocols. A clinician that desires a particular coordination behavior chooses an appropriate script from the database. Each script contains a list of device types needed to carry out a coordination activity. During the script execution start-up phase, the script interpretation framework attempts to acquire devices that are currently on the network and allocated to the current clinical context (*e.g.*, present in the current room). The clinician assists with device acquisition (*e.g.*, by selecting from a list of available devices of the appropriate type). After a complete set of required devices has been selected and confirmed as available (a device may be restricted from participating in more than one script at a time if it is being subjected to automated control, while "read only" devices may be shared), the script interpreter begins execution of the coordination script. Script execution may proceed without intervention, or may stop to receive input from the clinician.

The device coordination framework functionality that we have sketched above is similar to functionality being proposed in the ICEMAN draft standard from the MDPnP group. Although we have not designed our framework to directly implement that draft standard, one of the goals of the experiments reported in this paper is to illustrate that publish/subscribe architecture is capable of supporting the functionality required by the standard.

Implementation Issues and Requirements: The CDIC of Figure 2 requires all of the functionality of the Device Information Integration CDIC of Figure 1. In addition, the component categories (data producer, consumer, and transformer) must be enhanced to include *coordination components* that can be thought of as simple automata with I/O that interact with devices and clinicians. The notion of a device component must also be enhanced to include interface ports that provide data input to or control of devices.

Due to the increased levels of criticality associated with device coordination (as compared to the earlier scenario focusing on device data presentation), the associated programming and validation environment must be able to support more rigorous notions of specification and validation for greater regulatory oversight.

Deployment Scenarios: In contrast to the CDICs that included possible deployment configurations in which shared servers supported pub/sub activities across multiple hospital rooms, we believe that, for this highly safety-critical context, it is important to reduce the risk of unanticipated and potentially harmful interference by allowing a server to support only a single room at a time. Improvements in verification technology that can provide high confidence of non-interference between multiple coordination activities may eventually lead to this restriction being lifted (in fact, one reason for our work is to provide a test bed that facilities research on and demonstration of such technology). However, for the present, the risks of potential harmful interference are best mitigated by using separate dedicated servers for each coordination activity.

3 Goals

Below we list the design goals for our MDCF.

1. Provide distributed networking middleware infrastructure that enables devices/displays/databases from different vendors to be integrated with minimal effort.
2. Provide payload capabilities that support common data formats used in the medical device and medical informatics domains.
3. Provide an architecture that enables tailoring, integrating, and transforming device information streams.
4. Support the requirements of realistic device integration contexts as described in Section 2.
5. Develop an architecture whose performance and application-level programmability scales gracefully as the number of integrated devices and computational resources (*e.g.*, server machines) increases.
6. Provide basic functionality needed for reliability including options for guaranteed message delivery, logging/auditing, and persistent storage of messages.
7. Support a script programming model that makes it easy to assemble new functionality from building blocks.
8. Use infrastructure that is freely available and open source (to enable academic research).

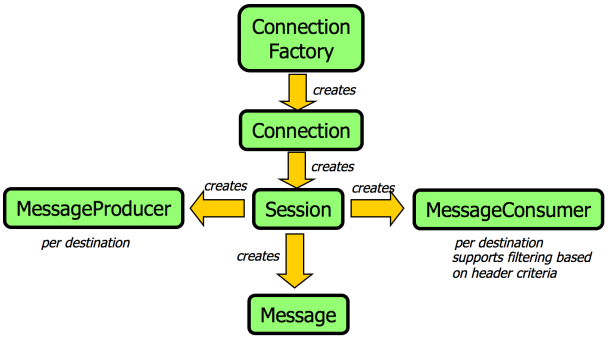

Figure 3. JMS primary objects

9. Use standards-based frameworks that are supported by enterprise-level implementations that can provide suitable performance in a realistic enterprise setting.
10. Because it will be difficult for academics to obtain real devices, support both real and simulated devices.

4 Architecture

4.1 MOM Foundation

The design of our core architecture is driven by practical realities of the clinical device integration contexts in Section 2, such as (a) flexible, dynamic information flow (frequently needing privacy), (b) heterogeneous systems, mechanisms, and needs, (c) many listeners, and many sources, and (d) time-critical, scalable performance. A message-oriented, publish-subscribe architecture with decentralized hubs, dynamic queuing, reliable message passing, and enterprise-grade deployment fits these criteria nicely. Thus, we chose to build upon a messaging-oriented-middleware (MOM) foundation. To address the goals of Section 3, we believe it is best to consider MOMs based on the Java Message Service (JMS) standard. JMS satisfies the criteria (a-d) above, while providing low-cost, open-source implementations for low barriers to entry and easy integration into research environments (Goal 8). In addition, there are multiple commercial enterprise-quality JMS implementations such as those found in IBM's WebSphere and Oracle's AQ products (Goal 9). JMS provides point-to-point or publish/subscribe topologies (addressing Goals 1 and 3), reliable or unreliable message delivery (Goal 6), and high performance (Goal 9). It enables distributed communication which is "loosely coupled, reliable, and asynchronous." In our application environment, its ability to pass simple data types as well as complex objects enables a clean integration with structured text standards such as HL7, as well as complex objects for seamless framework control (Goal 2).

Figure 3 presents the primary objects involved in JMS publish/subscribe communication. When a client wishes to originate a connection with a JMS provider, it uses the Java Naming and Directory Interface (JNDI) to locate a *Connection Factory* that encapsulates a set of connection-configuration parameters for the provider. The client then

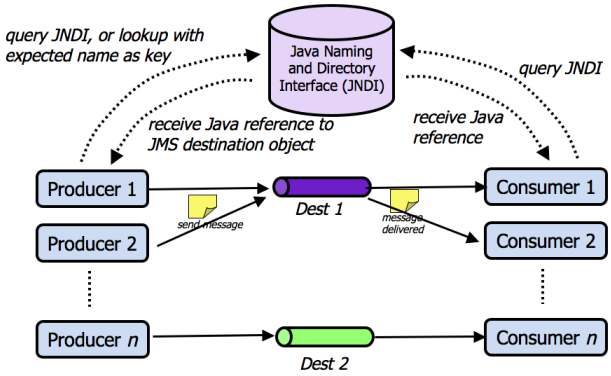

Figure 4. JMS destinations

uses the Connection Factory to create an active *Connection* to the provider (typically represented as an open TCP/IP socket between the client and the providers service daemon). In our architecture, clients will do all of their messaging with a single Connection. A Connection supports an *Exception Listener* that will be called when an connection fails (which we will use to handle situations in which a device unexpectedly disconnects in the middle of an activity). Once a connection is established, a client uses the connection to create a JMS *Session*.

Figure 4 illustrates that a JMS *destination* is an abstract entity to/from which a client publishes or receives a message. Destinations are located/retrieved via JNDI calls. A session serves as a factory for creating *MessageProducers* or *MessageConsumers* for a particular destination. To send a message, a client requests a session to create an empty message (of a particular type supported by JMS), the message contents are filled in, and a MessageProducer is called to send the message. To receive messages asynchronously (which is the method we will use in our framework), the client creates an object (a handler) that implements the MessageListener interface and sets that object as the listener for a particular MessageConsumer.

A session is a single-threaded context designed for serial use by one thread at a time. It conceptually provides a thread for sending and delivering messages for all message producers/consumers created from it, and it serializes delivery of all messages to all of its consumers.

Figure 5 illustrates that the abstract structure of a JMS message is divided into three parts: a header containing values used by both clients and providers to identify and route messages, a properties section containing application-defined or JMS-provider-defined key-value pairs that provide additional metadata about the message, and the payload of the message. A number of these fields such as Destination, DeliveryMode, MessageID, Timestamp, and Redelivered are not set by the client but by the infrastructure layer as a message is transmitted. We use the Timestamp field to gather performance information reported on in Section 6. Other fields such as CorrelationID and ReplyTo are

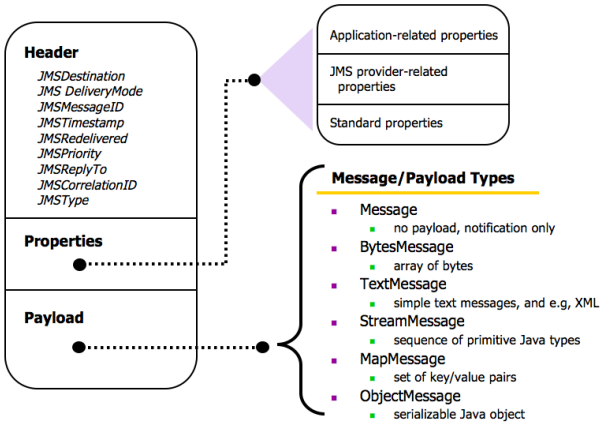

Figure 5. JMS message format

set by the client to guide responses to messages. We use CorrelationID to support the situation where we have multiple integration scenarios running on the same server. There are a few base administrative destinations (communication channels) that are shared among all running scenarios; each scenario sets a unique correlationID and watches for responses from the scenario administrator using the same ID.

Property values are set by the client prior to sending a message. When constructing a message consumer, a client can specify a filter expression that references fields in message headers and properties; only messages that pass the filter are delivered to clients. Thus, the primary purpose of message properties is to expose attributes for filtering. We currently use filtering only on header fields, but the property mechanim provides significant flexibility for enhanced functionality moving forward.

JMS provides a number of different formats for message payloads. We primarily use text messages (*e.g.*, HL7 and most other data) and object messages (*e.g.*, for DICOM images) (see Section 6).

4.2 MDCF Modules

We briefly summarize the structure of internal MDCF modules built in top of JMS (as seen in Figures 1 and 2), and refer the reader to the extended version of this paper for details [12].

A *device connection manager* listens on a dedicated JMS channel (stored in the JNDI) for messages indicating that a device desires to connect with the system. To simplify lower-level protocols and to facilitate the construction of mock devices, we assume that each device runs a JVM with a JMS client. Real devices that do not include an onboard JVM can be incorporated by attaching them to a JVM-capable adapter device.[2] Devices proceed through a authentication protocols that verifies that the connecting device is in a database (implemented using HSSQL) of approved devices and associated drivers (which provide API descriptions for interacting with each device).

Several consoles including a *maintenance console* (to allow installation of new device drivers, interaction scripts, *etc.*), *monitoring console* (that allows monitoring of events flowing through the infrastructure), and a *clinician console* that provides visualization of device and EHR data, invocation of and interaction with device coordination scripts. A *scenario manager* manages the life-cycle of scenario script executions including acquisition of devices needed in the script, creation of components and JMS channels to realize intercomponent communication, and tear-down of components and channels after script execution.

5 Programming Model

We anticipate that device integration scenarios will be implemented either by developers at a company that supplies an integration framework (who would find it advantageous to build up a collection of reuseable components or product lines to serve multiple customers) or by on-site clinical engineers (who may not be familiar with underlying middleware and network concepts). Thus, we have developed a component-based programming model that abstracts away the details of the lower-level infrastructure and facilitates rapid assembly of integration scenarios from reusable components (Goal 7).

The component model supports typed input/output event (asynchronous) ports with multiple categories of components, including data producers such as devices, data transformers that filter, coelesce or transform data streams, and data consumers that represent displays or data repositories. Some components may be both data producers and consumers, such as devices that may be controlled by others or health information databases.

We have built an integration scenario development environment in our Cadena framework [3]. Cadena provides component-based meta-modeling that enables us to define a domain-specific language of components for building device integration scenarios. Given a meta-model of the component language, Cadena generates a *component interface* editor that allows one to define component types and a *system scenario* editor that allows one to allocate and connect component instances to form an executable system. Cadena's rich type system allows one to define different type languages for component ports that capture specific properties of data communicated between components. Cadena provides a notion of "active typing" that continuously checks for type correctnesss as a system scenario is constructed in the graphical scenario editor.

Figure 6 shows a device integration scenario built in Cadena's scenario editor. Components corresponding to medical devices such as blood pressure and cardiac monitors appear on the right of the figure. Connections between components represent publish/subscribe relationships.

[2] Frameworks by Cerner and Philips/Emergin use similar adapters.

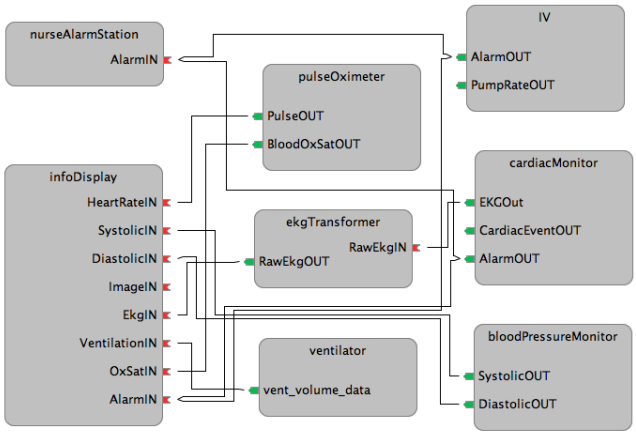

Figure 6. ICU scenario components

We have built a Cadena plugin that provides facilities akin to a very light-weight version of the CORBA Component Model (CCM). Given a Cadena type signature for an MDCF component, autocoding facilities generate a Java skeleton/container for the component. The skeleton contains all logic required by the framework to enable the component implementation to connect to the framework as a framework component (this includes automatically generating the logic for subscription assignment and publishing logic). The component developer then only needs to implement the "business logic" – the code that processes medical information (such as a data transformer or rendering routine) or device access logic (interaction with actual device sensor hardware).

Similar in spirit to CCM's deployment and configuration infrastructure, the plugin can also analyze a Cadena coordination scenario model and generate a MDCF specificiation file. The MDCF specification file consists of XML that describes the named component graph. The logical name of each component instance and the type of the component is present, as well as what inter-component connections exist. This information is used by the MDCF to locate the appropriate MDCF component class files and instantiate the coordination scenario.

We believe that the use of sophisticated architectural types and component encapsulation can help in constructing assurance cases for integration scenarios. Use of component technology helps prevent unanticipated interference between components by insuring that components only interact through explicitly declared ports. The strong typing in the Cadena modeling environment reduces the possibility of programming errors.

6 Experiments

Two categories of experiments were designed to evaluate the viability of the framework: baseline experiments and clinical scenario experiments. *Baseline performance* experiments use simple producer/consumer configurations to measure the raw performance of the framework as it propagates data representative of clinical contexts. *CDIC experiments* use device/display component configurations that correspond to the clinical integration contexts presented in Section 2 to assess the ability of the framework to support typical usage modes.

Three categories of data were considered in our experiments: device data (point data and streaming data from monitoring devices), alarm events (relatively infrequent anomaly events published by devices), and medical informatics data (relatively infrequent and large data sets corresponding, *e.g.*, to patient record data, drug dosing information, and medical images). Parameter settings (*e.g.*, the rates at which device data are published) are set to account for perceived worst case assumptions (maximum system requirements). For example, given a source device such as an electrocardiograph, a data update rate of once every 50 ms is considered frequent enough for a physician's data display to appear as if the data arrive in real time, so the data transfer and display process will not affect the quality of the associated clinical assessment. Other types of sensor data (*e.g.*, blood pressure, heart rate, or blood oxygen saturation) can arrive much more infrequently. In our experiments, we will simply assume that devices publish information at a minimum interval of once every 50 ms. Low latency is important for device and alarm data, but less so for informatics data.

Tests were performed on a single server representing the anticipated minimum machine configuration likely to be encountered in an enterprise-grade hospital information system (HIS) setting. We used a Sun Fire X4150 server with dual 2.8 GHz quad-core Xeon processors, 8 GB of RAM, a local 250 GB hard disk, and a gigabit Ethernet connection to the network fileserver. The server runs Linux 2.6.23, Java 1.5.0_13-b05, and OpenJMS 0.7.7-beta-1. OpenJMS was configured for non-persistent messaging unless otherwise noted. We observed that the current openJMS internal software architecture produced strongly asymmetric results; we expect other JMS implementations to provide more balanced performance. All results were averaged over multiple runs.

6.1 Baseline Performance

These experiments were designed to measure the throughput of the framework for single-step propagation (from a data producer to data consumer) given different types and sizes of clinical data. Performance was measured as a function of the numbers of producers/consumers under different connection topologies (fan-in/fan-out of producer/consumer relations).

6.1.1 Data Types and Connection Topologies

Three types of data were considered: simple event notifications, Health Level 7 (HL7) messages, and DICOM image

data.

Simple Event Notifications: Having minimal or no payload, these support the alarm notification of Section 2.2 (little or no payload), control instructions such as the X-ray activation of Section 2.3 (little or no payload) as well as many forms of device data such as the heart rate notification of Section 2.1 (small payload). To simulate messages of this type, we use JMS ByteMessages with a payload of 10 bytes.

HL7: Health Level 7 is a messaging standard for the electronic exchange of medical information. HL7 messages use a text format (frequently XML-based) to structure medical data, health record queries, and data from health records. Although theoretically unlimited in size, these message typically range between several hundred and several thousand bytes. Our base experiments use three sample HL7 messages from the CDC Immunization Record EXchange (iREX) project [10], where messages range in size from 313 bytes to 4312 bytes. The small 313-byte message is an HL7 patient vaccine record query message. The medium 2227-byte message contains a fragment of a patient record that notes adverse reactions to vaccinations (a VAERS record). The large 4312-byte message is also a VAERS record, but with more vaccination events noted.

DICOM: The DICOM image exchange and storage format supports high resolution digital images tightly coupled with patient information. For instance, a DICOM file or message will typically contain a digital image (JPEG or RLE/TIFF format), a header containing the patient name or identification, and other metadata such as image dimensions, format, color depth, manufacturer/software version, etc. [6, 11] For our experiments, we use sample DICOM data from [1]: "CR-MONO1-10-chest" (379 kB), "MR-MONO2-16-knee" (130 kB), and "MR-MONO2-12-shoulder" (70 kB).

Connection Topologies: The base experiments evaluate the framework with components arranged in basic topologies that are likely to appear in real-world CDICs. These topologies consider that some devices, databases, or displays (*e.g.*, a nurse's station display) may be shared within and across different scenarios. The topologies relating producers to consumers include 1 to 1, 1 to 50, 1 to 100, 50 to 1, 100 to 1. In each topology, producers operate at "full throttle" – emitting messages in a loop as fast as the infrastructure can handle them.

6.1.2 Baseline Experimental Results

Both message size and connection topology affect the rate at which messages will move through the framework. Larger messages take longer to marshall/unmarshall, which reduces the rate at which the system can move messages. Interestingly, throughput is greatly affected by the connection topology. Increasing the number of producers will not increase the message throughput nearly as much as increasing

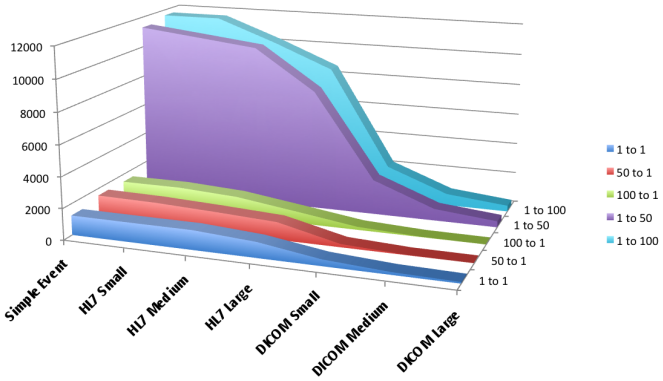

Figure 7. Message throughput

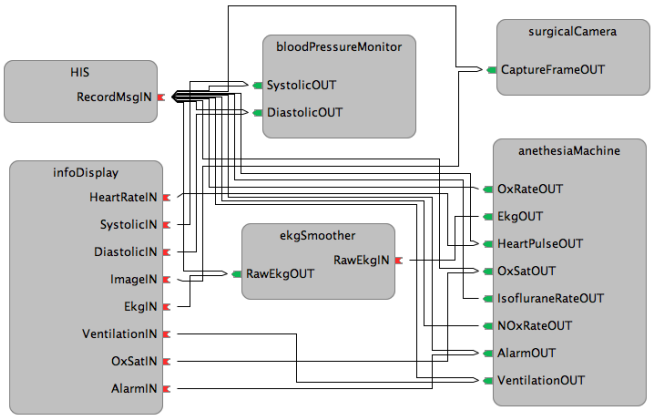

Figure 8. OR scenario components

the number of consumers. We suspect that this is because the JMS provider maintains a queue of pending messages that is shared between the provider's worker threads. In the case of many producers, many different messages can arrive at the message queue at the same time, and some resource contention can occur. When the number of consumers is scaled, the system merely has to remove one message from the queue and copy it to as many worker threads as system resources allow.

6.2 Critical Care Device Coordination

We begin the CDIC experiments with the Device Coordination Context discussed in Section 2.3. Due to its safety-critical nature, this context has stronger real-time requirements. As discussed in Section 2.3, we expect that hospitals or critical care providers will use a dedicated server for each operating or critical care room, and the server will run one scenario instance at a time.

For this experiment, we imagine an operating room equipped with the following medical equipment networked to the MDCF: an anesthesia machine with an integrated ventilator and electrocardiograph (*e.g.*, an Ohmeda Modulus CD/CV) plus a blood pressure cuff. The operating room is also equipped with a large heads-up display that renders

device data streams. In this scenario, we also incorporate a software component, a `Transformer`, that preprocesses the electrocardiogram data stream prior to the stream's rendering on the physical display. See Figure 8 for a graphical depiction of this scenario's logical components.

Mean latencies of the informational messages are excellent - typically 1 ms. Each producer generates one data message on its output ports once every 50 ms (small numerical data messages that denote current sensor state, or a 50 ms subsection of a continuous waveform). Alarm events are updated once every 5 seconds.

Although this experiment does not represent an explicit coordination activity, it is clear from the performance discussion that our infrastructure would also be able to support critical care coordination activities such as those discussed in Section 2.3 when OpenJMS is used as the JMS provider and persistent messaging is disabled. Enabling persistent messaging increases the mean latency to 5 ms, but the peak latencies rise significantly, (in this case the peak latency was 7.42 seconds), indicating that OpenJMS may not be appropriate for some critical care scenarios when persistent messaging is enabled.

Mode	*Mean*	$\% < 50ms$	$\% > 2 \times mean$
Non-Pers.	1ms	99.99	1.0
Persistent	5ms	99.62	0.7

Table 1. Message latencies - OR scenario

6.3 Integrated Displays and Alarms

This experiment combines both the Room-Oriented Device Information Presentation (Section 2.1) and the Alarm Processing (Section 2.2) CDICs. It demonstrates the ability of the MDCF to scale to ward level and still meet appropriate quality-of-service standards.

In this scenario, we imagine a large ICU ward with multiple rooms – each equipped with a blood pressure cuff, cardiac monitor, intravenous medicator, pulse oximeter, and ventilator. Each of these devices produces one or more data streams or alarm events (see Figure 6 for details). Each room is equipped with a configurable in-room, heads-up display that renders these data streams. The ward is equipped with a nurse's station display, which subscribes to all alarm events generated by any of the individual room's devices. This experiment replicates the scenario 1 - 100 times and aggregates all alarm messages to one nurse's station instance.

As can be seen from Figure 9, the framework easily scales to 20 rooms. Even when managing 20 rooms, the maximum observed latency for any system message is 227 ms. The vast majority of the messages are transmitted much more quickly. At 50 rooms, the mean latency remains good, but the maximum observed latency has increased to 3 seconds (the spread of latencies has also increased, as can be seen by the increase in standard deviation). At 100 rooms, the maximum observed latency has grown to 4 seconds, but most latencies are still within allowable bounds.

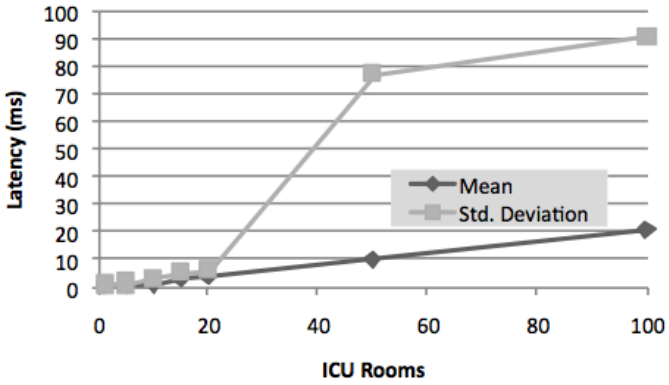

Figure 9. ICU latencies

7 Related Work

The ground-breaking MDPnP effort [13] has inspired many aspects of our work. MDPnP focuses on standards development targeted at critical care device coordination, including device connectivity protocols and a conceptual architecture of an Integrated Clinical Environment (ICE) that manages device connections and oversees inter-device coordination activities (which roughly corresponds to our CDIC of Section 2.3). The MDPnP effort follows other TATRC-funded initiatives that highlighted the need for standards-based interoperability architectures [16, 19, 5]; these efforts complement other medical device plug-and-play initiatives [4, 14].

MDPnP has not yet focused on performance/functionality evaluations of potential middleware infrastructures nor on implementing programming models for coordination. Our work complements the MDPnP effort by (a) exploring how a standard MOM framework (JMS) can provide the functionality and performance necessary to implement an ICE, (b) illustrating how the same framework can be used to provide scalable performance for additional CDICs (Sections 2.1 and Sections 2.2), (c) proposing a high-level component-based programming model for specifying integrated device behaviors, and (d) providing an open implementation to facilitate further experimentation.

Hofmann's Masters thesis [8] provides an excellent in-depth presentation of technical issues considered by MDPnP. Hofmann defines a meta-model to describe medical devices and a communication protocol to enable plug-and-play connectivity for compliant devices. A service-oriented system was implemented that can pair application requirements with device capabilities based on the ICE-MAN device meta-model. The system was tested (focusing on protocol correctness, not scalable performance) with simulated medical devices. Although our framework im-

plements protocols to connect devices with pre-installed drivers, we have not focused on true "plug-n-play" dynamic discovery/loading of drivers. It would be natural to incorporate Hofmann's proposed strategies into our infrastructure.

Several academic projects, including previous projects at KSU [21, 20, 22] have investigated interoperability standards for home-health and body-area networks. This work focuses primarily on consumer-oriented devices and networks as opposed to clinical use cases and enterprise-oriented architectures. In the coming months, KSU researchers plan a larger-scale demonstration that would integrate home health networks with the infrastructure described in this paper to support remote monitoring in a large assisted-living complex.

As noted earlier, several companies have recently brought device integration frameworks to market that target aspects of the CDIC functionality described here. The earliest of these were from device manufacturers that focused only on integrating devices that they themselves produced. Others, like Philips/Emergin, have targeted specific tasks, such as alarm management for a large class of devices from multiple vendors. To the best of our knowledge, these are closed proprietary implementations, and none of these targets the CDIC of critical care device coordination (which is the primary motivation for our work).

8 Assessment and Conclusions

Based on these experiences, we assess the ability of the JMS-based publish-subscribe architecture to satisfy the goals presented in Section 3.

What Worked Well: We believe the architecture/infrastructure presented here is an excellent candidate for enterprise-quality frameworks that aim to integrate devices, medical informatics systems, and multi-faceted displays in clinical contexts such as those presented in Section 2 (Goals 1 and 4). Industrial development trends and government coordination efforts indicate that the CDICs described here represent promising directions for expanding functionality of systems of integrated devices. We found JMS message types sufficiently rich to support the information exchanges required by our CDICs (Goal 2).

Our experiments indicate that the infrastructure can provide the scalability necessary to support realistic deployments in clinical environments (Goal 5). We were pleased that solid performance could be achieved with open source implementations; commercial enterprise JMS implementations can be expected to provide additional performance and reliability improvements (Goal 9).

The ability of a loosely-coupled component-based programming model to support rapid development of integration scenarios from building blocks (Goal 7) and flexible tailoring and transforming of information streams (Goal 3) coincides with our past experience using component-based development in embedded applications (*e.g.*, avionics systems). Our progress on this issue can substantially benefit other efforts such as MDPnP, since those efforts have previously focused on lower-level device protocol and integration issues as opposed to higher-level programming models.

The resulting open code base for our framework represents substantial progress toward a goal of providing resources to the academic and industrial research communities to facilitate further research and consensus-building on topics central to the emerging paradigm of systems of integrated and cooperating medical devices (Goal 8).

What Did Not Work Well: In our initial implementation efforts, we struggled to obtain acceptable performance when using JMS persistent message delivery (which routes all messages through a database) to provide recovery after a server malfunction. While we believe this mechanism can be useful to provide reliability (recovery would be especially important, *e.g.*, in the Critical Care Device Coordination CDIC in Section 2.3), it may be less important in other contexts such as the Room-Oriented Device Information Presentation CDIC in Section 2.1. Our experiments show that persistent delivery does indeed give acceptable performance for the Critical Care CDIC. Our experiments were carred out with an open-source HSSQL implementation of the database component. We anticipate that other database implementations or commercial database/JMS implementations would improve performance.

Where Improvements Can Be Made: We aim to significantly expand this initial, small set of mock devices (Goal 10) using publicly available databases of device data streams [15]. Our previous efforts in crafting wearable and ambulatory sensor-based devices that utilize interoperability standards [22, 18, 17] has guided us in the design of connectivity interfaces and device connection protocols for our framework. However, more implementation work is needed to harden these capabilities and to incorporate actual devices in the JMS-based. To validate our approach, we are incorporating several devices donated to KSU by Cerner.

References

[1] S. Barre. DICOM images – Sebastian Barre respository. `http://www.barre.nom.fr/medical/samples/`.

[2] Cerner CareAware. `http://www.cerner.com/public/Cerner_3.asp?id=29157`, 2008.

[3] A. Childs, J. Greenwald, G. Jung, M. Hoosier, and J. Hatcliff. CALM and Cadena: Metamodeling for component-based product-line development. *Computer*, 39(2):42–50, February 2006.

[4] Continua health alliance. `http://www.continuaalliance.org`, 2008.

[5] R. L. Craft. Toward technical interoperability in telemedicine. *Telemedicine and e-Health*, 11(3):384–404, June 2005.

[6] DICOM homepage. `http://medical.nema.org/`.

[7] K. Grifantini. “plug and play” hospitals: Medical devices that exchange data could make hospitals safer. MIT Technology Review, July 9, 2008, July 2008.

[8] R. Hofmann. Modeling medical devices for plug-and-play interoperability. Master’s thesis, MIT, June 2007.

[9] Alarm systems - general requirements, tests and guidance systems in medical electrical equipment and medical electrical systems. IEC 60601-1-8, 2003.

[10] CDC Immunization Record EXchange (irex) project. `http://www.dt7.com/cdc/`.

[11] W. B. Jr, S. Horii, F. Prior, and D. V. Syckle. Understanding and using DICOM, the data interchange standard for biomedical imaging. *Journal of American Medical Informatics Association*, 4(3):199–212, May 1997.

[12] Medical Device Coordination Framework (MDCF) – Kansas State University. `http://mdcf.projects.cis.ksu.edu/`.

[13] Medical device ”plug-and-play” interoperability program. `http://mdpnp.org/`, 2008.

[14] *Medical Device Communications: VuSpec*. IEEE, Piscataway, NJ, Feb. 2005. IEEE 11073 Standards.

[15] Physiobank medical device data stream repository.

[16] *The Role of Technology in Reducing Health Care Costs*. Sandia National Laboratories, Albuquerque, NM, Oct. 1996. SAND60–2469, DOE Category UC-900, Unlimited Release.

[17] D. Thompson and S. Warren. A small, high-fidelity reflectance pulse oximeter. In *2007 Annual Conference and Exposition, American Society for Engineering Education*, June 2007.

[18] S. Warren, D. Andresen, D. Wilson, and S. Hoskins. Embedded design considerations for a wearable cattle health monitoring system. In *2008 International Conference on Embedded Systems and Applications (ESA ’08)*, July 2008.

[19] S. Warren, R. L. Craft, R. C. Parks, L. K. Gallagher, R. J. Garcia, and D. R. Funkhouser. A proposed information architecture for telehealth system interoperability. In *Toward An Electronic Patient Record 99 (TEPR 99)*, pages 187–194, May 1999.

[20] S. Warren and E. Jovanov. The need for rules of engagement applied to wireless body area networks. In *Proceedings of the IEEE Consumer Communications and Networking Conference*, pages 979–983, Jan. 2006.

[21] S. Warren, J. Lebak, J. Yao, J. Creekmore, A. Milenkovic, and E. Jovanov. Interoperability and security in wireless body area network infrastructures. In *Proceedings of the 27th Annual Conference of the IEEE EMBS*, pages 3837–3840, Sept. 2005.

[22] J. Yao, R. Schmitz, and S. Warren. A wearable point-of-care system for home use that incorporates plug-and-play and wireless standards. *IEEE Transactions on Information Technology in Biomedicine*, 9(3):363–371, Sept. 2005.