Design Pillars for Medical Cyber-Physical System Middleware *

David Arney[1], Jeff Plourde[1], Rick Schrenker[1], Pratyusha Mattegunta[1], Susan F. Whitehead[1], and Julian M. Goldman[1,2]

1 **MD PnP Program, Massachusetts General Hospital**
Boston, MA, USA
info@mdpnp.org

2 **Harvard Medical School**
Cambridge, MA, USA
JMGOLDMAN@mgh.harvard.edu

Abstract

Our goal is to improve patient outcomes and safety through medical device interoperability. To achieve this, it is not enough to build a technically perfect system. We present here our work toward the validation of middleware for use in interoperable medical cyber-physical systems. This includes clinical requirements, together with our methodology for collecting them, and a set of eighteen 'design pillars' that document the non-functional requirements and design goals that we believe are necessary to build a successful interoperable medical device system. We discuss how the clinical requirements and design pillars are involved in the selection of a middleware for our OpenICE implementation.

1998 ACM Subject Classification J.3, D.2.1

Keywords and phrases medical device interoperability, ICE, ASTM 2761, 11073, clinical requirements, design pillars, requirements elicitation, validation

Digital Object Identifier 10.4230/OASIcs.xxx.yyy.p

1 Introduction

Medical device interoperability has the potential to reduce healthcare costs, improve patient outcomes and improve patient safety. Achieving interoperability requires that medical devices (including software applications) and other equipment share the same information model and communication protocol. This enables applications to work with any source of compatible data regardless of the manufacturer or specific device type. In this paper, we concentrate on the communication protocol aspect of interoperability, in particular the role of middleware.

A system using separate medical devices is a distributed system. There is a long and rich history of work in the field of distributed systems that can directly inform the development of interoperable medical cyber-physical systems (MCPS). One broadly accepted tenet of this work is that network architecture can be broken down into a number of layers; this is perhaps most commonly illustrated by the Open Systems Interconnection (OSI) Seven Layer Model. Breaking network architecture into these layers allows designing and reasoning about

* This publication was made possible by grant number 1U01EB012470 from NIH/NIBIB and award number W81XWH-12-C-0154 from Department of Defense US Army Medical Research and Materiel Command. Its contents are solely the responsibility of the authors and do not necessarily represent the official views of the NIH/NIBIB or the US Army or the Department of Defense.

Conference/workshop/symposium title on which this volume is based on.
Editors: Billy Editor and Bill Editors; pp. 1–8
OpenAccess Series in Informatics
OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

them independently - a transport layer can operate on many different network and data link layers that in turn can work with a multitude of physical layers. Middleware is software that implements some of these middle layers between an application and networking hardware. There are a great number of middleware implementations with widely varying capabilities that implement various subsets of the seven layers. Choosing an appropriate middleware for a particular domain is thus a complex undertaking that requires an understanding of what applications need and expect from the network.

Systems Engineering similarly has a long history with many lessons that can inform MCPS development. The most important lesson here is that user needs must be used to validate system designs. Broadly stated, technical requirements are used for verification (that "the system was built right") and user requirements are used for validation (that "you built the right system"). A technically flawless system that does not satisfy user needs will not be used.

In this paper, we present two types of user requirements and discuss how they can be used to validate middleware for MCPS. The two types of requirements are Clinical Requirements, which capture the needs of clinicians, clinical engineers, biomedical engineers, and other medically-oriented users and Design Pillars, which are broad-scope non-functional requirements that we and our team of collaborators have formulated over ten years of building MCPS implementations.

2 Implementations of ASTM F2761

Interoperable MCPS follow the architecture described in ASTM F2761-09(2013) [8]. This standard does not call out a specific middleware, but includes high-level requirements for any interoperable clinical environment. We aim specifically to support 'plug-and-play' interoperability, including the composition of devices that may be used as by applications as components of the integrated system. Other existing standards such as ISO/IEEE 11073 specify all layers of the network stack, from physical to application. We reuse parts of the 11073 family, notably the terminology set (11073-10101) and parts of the information model in our OpenICE implementation [13], adapting them as needed for use with a middleware.

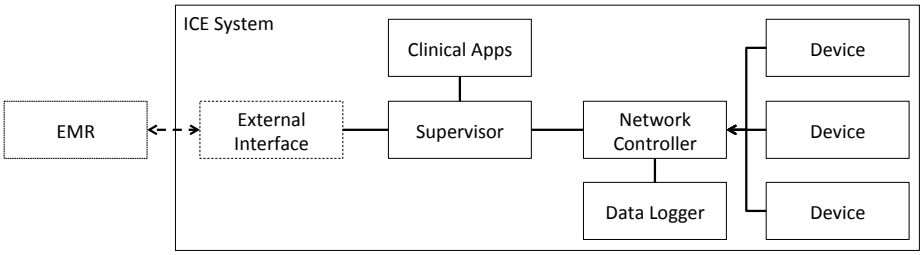

Figure 1 ICE Functional Architecture

ASTM F2761 defines a functional architecture for interoperable medical systems, illustrated in Figure 1. Clinicians use applications that interact with medical devices and that run on a Supervisor. Patient-connected medical devices and other equipment connect to the system via adapters (for legacy devices) or a built-in ICE Equipment Interface. The Network Controller ties together devices with the Supervisor, and sends data to the Data

Logger, which functions analogously to the black-box recorder of an airplane. The ICE system communicates with outside resources like an electronic health record (EHR) system, physician order entry system, or pharmacy system through External Interfaces. Though not shown in Figure 1, the patient and clinicians are key elements of the system, which is intended explicitly to improve patient safety and outcomes. Middleware can be used to implement this functional architecture by taking on responsibilities of the Network Controller and, to a lesser extent, the Supervisor. With a sufficiently capable middleware, almost all functions of the network controller are subsumed into the middleware.

Over the last 10 years we have built in our lab numerous prototype medical distributed systems [9] utilizing a variety of connectivity solutions. We started by using approaches built on web services such as SOAP and industrial systems like MODBUS to synchronize an X-Ray exposure with an anesthesia machine ventilator [3] [5]. This was followed by an infusion pump safety interlock built on a deterministic, hard real-time network implemented on custom FPGA hardware [4]. We have done extensive work on patient-controlled analgesia pumps including formal analysis of pumps [6], formal analysis of systems [7] [12] [2], and closed-loop control [14]. In addition to publications, these systems were presented at medical conferences including HIMSS, the American Society of Anesthesiologists annual conference, the Society for Technology in Anesthesiology annual meeting, and other venues. Feedback gathered from clinicians at these venues has gone into each iteration and been included in the clinical requirements and design pillars. This body of work forms the basis for claiming the validity of the design pillars in Section 3.

3 Design Pillars for Successful Interoperability

We have given the name Design Pillars to the set of non-functional requirements that summarize the approach that we believe is necessary to achieve safe, adoptable medical device interoperability. Other standards, including ISO 14971, IEC 60601, and FDA's guidance documents on risk management also include important guidance for interoperable systems. This list has a different focus, aiming to capture the normally unwritten goals and philosophy needed to achieve successful interoperability. These design pillars are a work in progress and we welcome additions, comments, and arguments about them.

No Silos Work toward a concrete implementation of the ICE standard has matured to a point where harmonization is required. Silos of interoperability work that cannot successfully interoperate are self-defeating. By identifying the most important characteristics of a middleware for ICE systems we can begin the process of selecting the most appropriate foundation for the platform. Future work can then proceed on a new generation of clinical applications that operate within the scope of this platform.

Open Source There must be an open source reference implementation. We will share our software code and documentation with the community, giving them the necessary tools to adapt and utilize our software, including commercial reuse. We will prefer dependencies (tools and software libraries) that are available with open source and little or no cost. We will demonstrate how a member of the broader community can easily make use of our work. Our development is done in an open repository. Documentation, the ticketing system, and bug tracking are all publicly visible. We encourage anyone interested to contribute to this effort. To demonstrate the feasibility of proposed solutions, prototype implementations are required. Such prototypes are most useful when they can be shared. This does not preclude closed-source implementations and commercialization once the conceptual use of a middleware to build a platform for ICE apps has been proven. An open source reference

implementation permits other implementers to perform testing and reuse code as appropriate.

Existing Standards Interoperability must be built on standards, utilizing existing software standards to the greatest possible extent. Where existing standards must be corrected, completed, or extended, the rationale must be documented.

Security Medical systems inherently touch human lives and private information. ICE implementations must be secure to the greatest extent possible. Security in this domain encompasses a tremendous range. Most relevant to middleware selection are the needs for identification, authentication, and authorization of connected devices, clinical users, and patients. Information in transit and at rest must be secured with appropriate use of encryption.

There is an apparent tradeoff between security and usability. Security features must not slow down or prevent urgent clinical use.

Scalability and Extensibility ICE implementations must scale gracefully. A platform that enables a revolution in bedside devices must scale to support the next generation of devices. Therefore even while we're building concrete prototypes with the current generation of medical devices we must anticipate a newer generation of devices that we expect will furnish higher resolution data streams. Software simulation should be used for initial stress testing, testing on hardware may also be necessary. The platform, supporting current generation devices, should exhibit a great deal of underutilized capacity. ICE encompasses data and control at scales from the bedside to the globe and must support integration at these scales.

High Availability We are building software that must guarantee high availability. ICE supports the integration of multiple sources of patient data. Components that fail should be seamlessly replaced by redundant data sources or other components if they are available. Put another way, risk control measures need to take into account component malfunctions. ICE should support achieving single fault tolerance for applications. Dependability is availability plus reliability - i.e., it's there when you need it and won't break while you're using it.

Performance Performance is another key to acceptance by the clinical community. Sluggish performance may be inconsequential in the laboratory setting but a poorly performing system in the clinic consumes a critical resource; the clinician's time. Poor performance can also encourage clinicians to marginalize the system; isolating the threat to their workflow. ICE implementations must support dynamic detection and reporting of performance degradation.

Visibility of runtime configuration ICE implementations must surface the state of the system; for instance, the connection state of devices should be readily available to a user. When the system is in an undesirable state, for example lacking connectivity to a critical medical device, it is important that information be made available. A system operating with hidden states will never earn the confidence of clinicians, but neither will a system cluttered with unnecessary information. The platform must also allow for the plug-and-play assembly of medical devices and because of this the configuration at runtime is the only source of information about how the system is configured.

Generic Interface Each component will share its data representation in common. Software shared in common among components will mediate all communication.

External Connectivity ICE implementations must interface with external systems. Some examples of external systems include an EMR system, an eHealth eXchange (NwHIN), departmental systems (such as pharmacy), or network time protocol (NTP) servers inside or outside of the hospital or home.

Novel Applications ICE implementations must enable the development of novel applications that run within their frameworks. The point of ICE is to enable new clinical applications to improve patient outcomes and safety.

Clinical Scenarios Requirements for ICE implementations should be derived from publicly available clinical scenarios so that traceability of technical requirements can be maintained. Technical requirements must be linked to clinical requirements which are derived from clinical scenarios. Technical design will also be informed by those scenarios and linkages between design decisions and high-level clinical requirements must be documented.

Community Involvement Developers of ICE implementations must maintain awareness of developments in other large-scale initiatives and relevant standards bodies. The linkages between external developments and implementation design decisions must be explicitly documented. Findings should be shared back with Standard Development Organizations where possible.

Forensic Data Logging ICE implementations must create a credible log of all activity so that adverse events can be investigated in order to surface and trace root causes of faults in the distributed system. Every aspect of implementations must avoid any data pathways that may "sidestep" this logging (while balancing this with our need for scalability and security). Information known to bypass the data logger must be documented with a rationale.

Plug and Play Components can be added to and removed from the system at any time. The system must dynamically determine and monitor the presence of components. In the interests of security, scalability, and performance components may be refused by the system for various reasons but this refusal must be surfaced per 3. Applications must handle the disappearance of required data and control sources or sinks and the appearance of new sources and sinks gracefully.

Regulatory Pathway ICE implementations operate in a regulated space. The regulators vary geographically, but the need to demonstrate the safety and essential performance of ICE systems and components is universal. To achieve this, ICE implementations should be designed and implemented in such a way as to facilitate regulatory clearance. Following the other design pillars should ease regulatory burdens.

Industry Adoptability The goal of ICE to achieve dramatic improvements in patient outcomes and safety can only be met if such systems are commercially available. To this end, ICE implementations (particularly open source implementations) should facilitate commercial reuse. At the same time, common networking pieces such as data representations must be shared and developed in common.

Human Factors The user interface and other human factors issues need to be carefully designed and tested in realistic environments so that new hazards that are introduced are adequately controlled. For instance, when a device is operating as a component of a larger system, its front panel must display an indicator that it's under remote control.

4 OpenICE and Clinical Requirements

We have built an open source implementation of an interoperable medical device system based on the ICE standard. We call this implementation OpenICE, and it is available on SourceForge [13]. This implementation is built using the OMG standard middleware DDS[11]. Implementing using DDS has helped to inform our requirements, but our clinical requirements and design pillars are not tied to DDS. We have spent extensive time building implementations on a variety of platforms and we believe that DDS is a good (stable, well-supported, variety of implementations available, etc.) choice, but not the only choice. We hope that our approach is useful to anyone implementing an interoperable medical system on any platform.

An ideal middleware would support an abstract API that would permit many instantiations

on varying hardware and software platforms. DDS approaches this ideal in that the OMG standard specifies an API that may be implemented in many ways. ICE implementations have a wide range of requirements, for instance for timing and latency. Creating an abstract API that will support scalability, reliability, fault tolerance, and safety analysis is is not trivial, and we believe that this is a promising direction for research.

Our middleware choice was, and continues to be, driven by a combination of our design pillars and our clinical requirements. The pillars and clinical requirements drive the technical requirements that the middleware must meet for a particular system instantiation. We validate that a middleware is appropriate for use in building interoperable MCPS such as an ICE implementation by evaluating it against the design pillars and its capability to support the needs documented in the clinical requirements. By documenting and assessing user needs we gain assurance that the system we build will be suitable for use in its intended environment.

Our approach is to allow clinical focus groups [10] to suggest clinical scenarios, which are captured either in person or through our prototype clinical scenario repository [1]. These scenarios then suggest clinical requirements, such as the samples shown in Figure 2. These clinical requirements imply technical requirements which are implemented to build a concrete system such as OpenICE. We use the technical requirements to verify the implementation, the clinical requirements and design pillars to validate the implementation, document gaps, and iterate. This waterfall development style description is overly linear, and it's important to realize that design and implementation are likely to iterate rapidly.

Clinical scenarios may document a situation where patient outcomes or safety could be improved by the use of interoperable devices. It is vital that the set of scenarios also include situations where a technical integration failure or lack of interoperability leads to patient harm, as well as situations where interoperability leads to new hazardous situations. Scenarios can be reflect an actual or imagined sequence of events that happened, or they can be constructed from an imaginable sequence of events derived from what policies and guidelines exist to prevent. In this work, we concentrate on the use of clinical requirements and their influence on middleware selection, rather than the process of moving from clinical scenarios to clinical requirements or from clinical to technical requirements.

The clinical requirements primarily represent the interactions of the system, including constituent devices, with users including clinicians and the patient. Our clinical requirements have come from elicitation sessions, clinicians, hospital policies, existing documentation, ASTM F2761 Annex B, clinical care guidelines, nursing documentation, clinical specialists, incident reports, and other groups. Figure 2 contains a selection of clinical requirements that have direct implications for middleware selection. For instance, consider SCR2 "The ICE System shall notify users when it loses connectivity with any of its components." These clinical requirements are written from the perspective of the clinical user, who may have little or no knowledge of how the system works; they are a form of black box requirements. This requirement could be implemented in a wide variety of ways. There are no requirements stated for timing, for how the notification should happen, or for which component should do the notification. Such specializations of the requirement follow from specific use cases and specific implementations. The specialization of SCR2 will be very different for an ICE implementation intended to run only an application that sends data to an offline documentation archive versus an implementation intended to support running an application controlling a closed-loop infusion of a fast-acting drug. SCR2 and SCR1 may also raise the eyebrows of those experienced in real-time systems. The closest possible match may not be a very good match at all, which is why review is required, and any deviation may throw off

Figure 2 Sample Clinical Requirements

SCR1: The ICE system shall be aware of the required frequency / accuracy / reliability of the incoming data for each parameter based on clinical significance, and shall choose the closest available frequency / accuracy / reliability on the device and provide this information to the clinician for review.

SCR2: If the device connected to the ICE system is not capable of providing the required frequency / accuracy / reliability of the incoming data for each parameter based on clinical significance, the ICE system shall choose the closest available frequency / accuracy / reliability on the device and provide this information to the clinician for review.

SCR3: The ICE System shall notify users when it loses connectivity with any of its components.

carefully engineered timings. It is important to remember that these requirements capture clinical needs as voiced by clinicians. They are not technical engineering requirements, and they are subject to interpretation and change in building implementations. Validation that a given implementation satisfies the clinical requirement is inherently subjective. It is our intention in compiling these that they be relatively unambiguous and reflect clinical consensus. The clinical requirements shown in the examples are generic in the sense that they are meant to apply to all ICE systems.

5 Conclusion

We have presented an approach to validating middleware selection for MCPS using user needs as documented in design pillars and clinical requirements. We are using this approach in our development of our OpenICE implementation, and we believe that the user needs we document here will be useful for others who are working on medical device interoperability. Our design pillars are intended to support making interoperability a community activity. We want to be able to share interface code, test suites, and requirements with the whole interoperability movement, not just ideas.

Our goal is to improve patient outcomes and safety through interoperability. To achieve this, it is not enough to build a technically perfect system, even one that satisfies all of the clinical requirements. Our design pillars document the non-functional requirements and design goals that we believe are necessary to build a successful interoperable medical device system.

Acknowledgements Our team at the MD PnP Program has had the pleasure of working with many generous and brilliant collaborators. We would like to thank Sandy Weininger and Yi Zhang at FDA CDRH and our collaborators on the Quantum Medical Device Interoperability project, namely Insup Lee, Oleg Sokolsky, Andrew King and the rest of the PRECISE team at the University of Pennsylvania, John Hatcliff and the members of the Medical Device Coordination Framework project at Kansas State, Lui Sha and his team at UIUC, Tracy Rausch, Wayne Saari and the DocBox crew, Dick Moberg, and Mike Robkin and all of Anakena Solutions. We've all spent many hours hotly debating issues around medical device interoperability. This paper represents some of our perspective, which has been greatly improved by our collaborators input. Oversimplifications and omissions are our own. We're looking forward to the next round of discussion.

References

1 Diego Alonso, Jeff Plourde, Sandy Weininger, and Julian M. Goldman. Web-based clinical scenario repository (CSR). In Poster Presentation at the Society for Technology in Anesthesia Annual Meeting, 2014.

2 Rajeev Alur, David Arney, Elsa L. Gunter, Insup Lee, Jaime Lee, Wonhong Nam, Frederick Pearce, Stephen Van Albert, and Jiaxiang Zhou. Formal specifications and analysis of the computer-assisted resuscitation algorithm (cara) infusion pump control system. Software tools for technology transfer, 5(4):308–319, 2004.

3 D. Arney, J.M. Goldman, Insup Lee, E. Llukacej, and S. Whitehead. Use case demonstration: X-ray/ventilator. In High Confidence Medical Devices, Software, and Systems and Medical Device Plug-and-Play Interoperability, 2007, page 160, June 2007.

4 David Arney, Sebastian Fischmeister, Julian M. Goldman, Insup Lee, and Robert Trausmuth. Plug-and-play for medical devices: Experiences from a case study. Biomedical Instrumentation & Technology, 43(4):313–317, July 2009.

5 David Arney, Julian M. Goldman, Susan F. Whitehead, and Insup Lee. Synchronizing an x-ray and anesthesia machine ventilator: A medical device interoperability case study. In BIODEVICES 2009, pages 52 – 60, January 2009.

6 David Arney, Raoul Jetley, Paul Jones, Insup Lee, and Oleg Sokolsky. Formal methods based development of a PCA infusion pump reference model: Generic Infusion Pump (GIP) project. In HCMDSS-MDPNP '07: Proceedings of the 2007 Joint Workshop on High Confidence Medical Devices, Software, and Systems and Medical Device Plug-and-Play Interoperability, pages 23–33, Washington, DC, USA, 2007. IEEE Computer Society.

7 David Arney, Miroslav Pajic, Julian Goldman, Insup Lee, Rahul Mangharam, and Oleg Sokolsky. Toward patient safety in closed-loop medical device systems. In Cyber-Physical Systems (ICCPS 2010), April 2010.

8 ASTM F2761-09(2013). Medical Devices and Medical Systems - Essential safety requirements for equipment comprising the patient-centric integrated clinical environment (ICE) - Part 1: General requirements and conceptual model. http://www.astm.org/Standards/F2761.htm.

9 Julian M. Goldman, Mike Jaffe, Dave Osborn, and Sandy Weininger. The integrated clinical environment (ICE) standard (ASTM f2761-09) - the first ten years. In Poster Presentation at the Society for Technology in Anesthesia Annual Meeting, 2014.

10 Julian M. Goldman, Susan F. Whitehead, and Sandy Weininger. Eliciting clinical requirements for the medical device plug-and-play (MD PnP) interoperability program. In Anesthesia & Analgesia: Abstracts of Posters Presented at the International Anesthesia Research Society 80th Clinical and Scientific Congress, March 2006.

11 Object Management Group. Data distribution service (DDS). http://portals.omg.org/dds/, March 2014.

12 Andrew King, David Arney, Insup Lee, Oleg Sokolsky, John Hatcliff, and Sam Procter. Prototyping closed loop physiologic control with the medical device coordination framework. In 2nd Workshop on Software Engineering in Health Care SEHC 2010, May 2010.

13 MDPnP Interoperability Program. OpenICE software repository. http://mdpnp.org/MD_PnP_Program____OpenICE.html, March 2014.

14 Carl F. Wallroth, Julian M. Goldman, Jurgen Manigel, Dave Osborn, T Roellike, Sandy Weininger, and Dwayne Westenskow. Development of a standard for physiologic closed loop controllers in medical devices. In Poster Presentation at the World Congress of Anesthesiology, 2008.