Detecting Interoperability Failures in Interoperable Medical Device Systems

Krishna K. Venkatasubramanian*, Jin-Hyun Kim†, Oleg Sokolsky†, Vasiliki Sfyrla‡, Eugene Vasserman§, Insup Lee†
*Worcester Polytechnic Institute, kven@wpi.edu
†University of Pennsylvania, {jinhyun,sokolsky,lee}@cis.upenn.edu
‡Unafiliated, vasiliki.sfyrla@gmail.com
§Kansas State University, eyv@ksu.edu

***Abstract*—This paper addresses the problem of high-assurance operation for medical cyber-physical systems built from interoperable medical devices. Such systems are different from most cyber-physical systems due to their plug-and-play nature: they are assembled as needed at a patients bedside according to a specification that captures the clinical scenario and required device types. Due to the integration of disparate medical devices, at patient-side, such interoperable medical device systems (IMDS) are prone to interoperability failures. We define interoperability failures as the observance of unsafe behaviors within the IMDS despite the fact that all individual components in the IMDS are working as expected. In this work we provide an overview of the importance of detecting interoperability failures and our overall approach to detecting such failures once they are assembled on-demand at run-time.**

I. Introduction

Interoperable medical device systems (IMDS) are medical cyber-physical systems that can *integrate information from multiple clinical sources in a context-sensitive way* to guide patient care or prevent common critical mistakes [9]. IMDS are essentially *dynamically composed* cyber-physical systems. They are fundamentally different from the traditional statically-composed systems (e.g., automotive or avionic systems) and architectures (e.g., integrated modular avionics [11]), where the constituent hardware and software entities (and their platforms) are known at design time and do not change over the life-time of the system. On the other hand every instance of IMDS can be different. IMDS typically consists of two main entities: *medical devices* involved in the treatment of patients and *controller applications* (henceforth referred to as (*medical apps*) that coordinate these devices. The medical apps run on a middleware *coordinator* platform that consists of a routing agent called the *network controller* and an app execution environment called the *supervisor*. The coordinator provides the infrastructure to enable the medical devices to work together to provide patient care as defined by the app.

The work thus far in implementing interoperability of medical devices has focused on functional interoperability, that is, making devices coordinate despite the differences in the app's expectations and the device platform's capabilities with focus on *regulatory* issues [5] and safety analysis of plug-and-play medical systems [8], [2], [1]. Efforts in the standards community such as ASTM2761 [3], MDPnP [10], and Continua [4], and 11073 [7] are also trying to address this problem. In this work, we are focusing on a complementary issue. *We want to move from the more "traditional" functional interoperability to enabling fault-tolerance during the interoperability.*

The current regulatory regime requires us to view systems of interconnected devices that collaborate, as a single medical device and assure its safety as a whole [5]. As with any device or system, safety assurance requires us to demonstrate the following two goals: (1) the operation of the system is safe in the nominal operating mode, i.e., in the absence of any faults, and that (2) any faults that may take the system outside of the nominal operation are detected and properly reported to the system operator (in this case, clinician or clinical engineer). The first of these goals is performed by analyzing the medical apps for compliance with the clinical scenario and is outside the scope of this paper. The second goal implies that the dynamically composed medical systems have to be capable of detecting and locating emergent operational failures at run-time. Though operational failures can occur for many reasons in IMDS, in this work, we focus particularly on **interoperability failures**.

II. Interoperability Failures in IMDS

We define *interoperability failures in IMDS* as the observance of unsafe behaviors within the IMDS despite the fact that all individual components in the IMDS are working as expected. Typically such failures occur due to mismatch of assumptions between the various components in the IMDS.

> **Example:** Imagine a closed-loop patient controlled analgesia system, where a patient is given pain medication using an infusion pump. Based on their current blood oxygen levels, the volume of the medication infused is continuously and automatically controlled by a specifically designed controller app. The controller app issues a ticket to infusion pump to allow infusion to happen for a specific time-interval. Now, as app developer and the infusion pump manufacturer may not be the same entity, they might have differing assumptions about how the other operates. Therefore the app may assume that anytime it issues an infusion ticket, the infusion

pump has no active infusion ticket on it. However, the infusion pump may have no such assumptions. Therefore, if it receives a new ticket when an old one is still active, it may just simply add on the duration of the new ticket to the remaining duration of the currently active ticket (or even override the original ticket) and keep infusing for unsafe duration of time. Here, none of the individual IMDS components failed, but an unsafe over-infusion occurred anyway. Interoperability failures occur in a manner that is undetectable within the IMDS' exception handling infrastructure and require a global view of the operation of the IMDS instance. ■

Leveson et al. have tackled interoperability failures in the context of statically composed systems [6]. We believe that our work is complementary to that of Leveson's work and in some instance goes beyond the assumptions of her work. If we were dealing with statically composed systems, we can easily use STPA to identify and address interoperability issues. With IMDS, we are dealing with dynamically composed systems whose base components, i.e., medical devices and controller app are not known until deployment time. This means, knowledge of interoperability failures is not sufficient to prevent their occurrence, we need to leverage the knowledge of interoperability failures and build monitors to detect their occurrence while the IMDS is operational. Our aim in this project is to develop techniques to automate the process for detecting interoperability failures in such dynamically composed IMDS.

III. Approach for Detecting Interoperability Failures in IMDS

The on-demand nature of IMDS imposes difficulties in the ensuring fault-tolerance as we do not know the environment of deployment (i.e., medical devices and medical apps). In this regard, we take a specification-based simultaneous top-down, bottom-up approach to detect interoperability failures in IMDS. Our approach consists of the follow three building parts:

1. To develop the interoperability failure model, we start with a verification of the app (controller + device models) using assume-guarantee reasoning, and prove that the whole app is safe as long as the assumptions are satisfied. This would ensure that the assumptions are sufficient and nothing important is missed. As these assumptions would have to be monitored, they provide us with a blueprint that describes the interoperability failures that are of interest to us and therefore need to be detected.

2. Given the failure model we derive high-level monitor logic for detecting potential interoperability failures in the IMDS. However, the monitor logic is a high-level representation of interoperability failures that is independent of the individual medical devices and the app used in the IMDS. For the monitor logic to detect interoperability failures, we need have knowledge of the current internal state of the IMDS components. For this we utilize low-level platform-specific detectors that output the state of the medical devices in the IMDS and the controller app. In this part of the work, our aim is to provide guidance for the device manufacturers and the app developers so that they can expose appropriate the internal states of the their respective device and app, by adding hooks on their software.

3. Given high-level monitor logic and the platform-specific detectors, we can take the output of the detectors and combine them with the monitor logic to detect interoperability failures. However, since the monitor logic and the platform-specific detectors may not be located on the same entity and may communicate over the network, i.e., they are distributed, we need to ensure that the detection of interoperability failures does not suffer during the operation of the IMDS, despite issues such as lack of universal clock or network loss/delay.

IV. Conclusions

The medical device industry is undergoing a rapid transformation, embracing the potential of embedded software and network connectivity. Instead of stand-alone devices that can be designed, certified, and used independently of each other to treat patients, we are facing a near future with distributed systems that simultaneously monitor and control multiple aspects of the patient's physiology. In this work we are focused on the detection of interoperability failures in IMDS. We provided a definition of interoperability failures, their importance in the IMDS context and our overall approach to detecting them at run-time.

References

[1] D. Arney, J. M. Goldman, S. F. Whitehead, and I. Lee. Synchronizing an x-ray and anesthesia machine ventilator: A medical device interoperability case study. 2009.

[2] D. Arney, R. Jetley, P. Jones, I. Lee, and O. Sokolsky. Formal methods based development of a PCA infusion pump reference model: Generic infusion pump (GIP) project. In *High Confidence Medical Devices, Software, and Systems and Medical Device Plug-and-Play Interoperability, 2007. HCMDSS-MDPnP. Joint Workshop on*, pages 23–33. IEEE, 2007.

[3] Medical devices and medical systems – Essential safety requirements for equipment comprising the patient-centric integrated clinical environment (ICE). http://enterprise.astm.org/filtrexx40.cgi?+REDLINE_PAGES/F2761.htm.

[4] R. Carroll, R. Cnossen, M. Schnell, and D. Simons. Continua: An interoperable personal healthcare ecosystem. *Pervasive Computing, IEEE*, 6(4):90–94, 2007.

[5] J. Hatcliff, E. Vasserman, S. Weininger, and J. Goldman. An overview of regulatory and trust issues for the integrated clinical environment. *Proceedings of HCMDSS 2011*, 2011.

[6] T. Ishimatsu, N. G. Leveson, J. Thomas, M. Katahira, Y. Miyamoto, and H. Nakao. Modeling and hazard analysis using stpa. 2010.

[7] ISO/IEEE 11073 Committee. http://standards.ieee.org/findstds/standard/11073-10103-2012.html.

[8] C. Kim, M. Sun, S. Mohan, H. Yun, L. Sha, and T. F. Abdelzaher. A framework for the safe interoperability of medical devices in the presence of network failures. In *Proceedings of the 1st ACM/IEEE International Conference on Cyber-Physical Systems*, pages 149–158. ACM, 2010.

[9] K. Lesh, S. Weininger, J. Goldman, B. Wilson, and G. Himes. Medical device interoperability-assessing the environment. In *HCMDSS-MDPnP*, 2007.

[10] Medical device "plug-and-play" interoperability program. http://mdpnp.org/, 2008.

[11] P. J. Prisaznuk. ARINC 653 role in integrated modular avionics (IMA). In *IEEE/AIAA Digital Avionics Systems Conference (DASC)*, 2008.