Towards Assurance for Plug & Play Medical Systems

Andrew L. King[1], Lu Feng[1], Sam Procter[2], Sanjian Chen[1], Oleg Sokolsky[1], John Hatcliff[2], Insup Lee[1]

[1] University of Pennsylvania, Philadelphia, USA
{kingand | lufeng | sanjian | sokolsky | lee}@cis.upenn.edu
[2] Kansas State University, Manhattan, USA
{samprocter | hatcliff}@k-state.edu

Abstract. Traditional safety-critical systems are designed and integrated by a systems integrator. The system integrator can asses the safety of the completed system before it is deployed. In medicine, there is a desire to transition from the traditional approach to a new model wherein a user can combine various devices post-hoc to create a new composite system that addresses a specific clinical scenario. Ensuring the safety of these systems is challenging: Safety is a property of systems that arises from the interaction of system components and it's not possible to asses overall system safety by assessing a single component in isolation. It is unlikely that end-users will have the engineering expertise or resources to perform safety assessments each time they create a new composite system. In this paper we describe a platform-oriented approach to providing assurance for plug & play medical systems as well as an associated assurance argument pattern.

Keywords: medical device interoperability, safety assurance, compositional safety

1 Introduction & Motivation

Traditionally, safety-critical systems have been designed and integrated as monolithic units before they are delivered to the customer. Typically, a prime contractor manages development of the system from design through final systems integration. Because the prime contractor manages the entire development process, they are in a unique position to assess the completed product for safety: They know what components comprise the system, how those components interact (*e.g.,* as verified via integration testing), the intended use of the system and the system-level safety requirements. Very often in regulated domains, such as aviation and medical systems, the prime contractor must also construct an assurance case which is an argument that the system satisfies its safety requirements.

In medicine, clinicians currently deliver therapy by manually coordinating collections of independently developed devices. Now that many devices marketed today already include some form of network connectivity (serial ports, Ethernet, 802.11 or Bluetooth wireless) clinicians are recognizing the potential to automate device coordination via external control applications [8]. Ideally, future medical devices would support plug & play protocols which would allow clinicians to construct networks of medical devices that automatically interoperate to automate life-critical clinical workflows [7]. The integration model for plug & play systems would differ from traditional systems because

they would not be supplied or integrated by a single vendor. Instead, a Health Delivery Organization (HDO) would purchase interoperable devices, infrastructure (*i.e.,* computational platforms) and software applications implementing clinical algorithms (*i.e.,* "apps") from a variety of different vendors. Specific medical systems would then be assembled from devices on-hand to address a particular clinical need. While practical use of such systems is still in the future, there are emerging interoperability standards [19] and prototype implementations that aim to support this vision [16,18]. In this paper, we study the problem of constructing safety assurance arguments for plug & play medical systems intended to provide life-critical therapy (*i.e.,* where failure of the system could result in death or serious injury - see the example in Section 2). We describe how the plug & play integration paradigm has serious implications for safety assurance and propose an approach for constructing an assurance argument for such systems.

Plug & play medical systems will be assembled by their (non-technical) users which means that there will not be a single entity with technical competency (*e.g.,* prime contractor) positioned to assess the safety of a specific combination of devices. The lack of a traditional prime contractor poses a challenge to ensuring the safety of these systems for two reasons: First, safety is a property of systems that arises from the interactions between system components [17]. Second, what constitutes safe inter-device interactions will vary considerably between different clinical scenarios. It is critical to ensure that interactions between devices are predictable *and* that those interactions satisfy the safety requirements of the given clinical scenario. Since it will not be known specifically which devices will be assembled into the composite system *a priori*, traditional methods of assessing safety cannot be used.

A number of efforts in academia, industry, and standards groups have studied different aspects and implications of plug & play medical systems. Nearly all of the prior work has assumed that plug & play medical systems would exist within a "platform-oriented ecosphere" similiar to what exists in different segments of the consumer electronics industry (*e.g.,* USB peripherals for personal computers or "apps" for smartphones). In the consumer-electronics industry these ecospheres ensure *interoperability* between components in the ecopshere. For example, one can reasonably expect a consumer operating system to seamlessly interoperate with a new USB keyboard or that an "app" downloaded from an official "app store" will be able to access and use smartphone hardware.

We believe that, in addition to supporting interoperability, a platform-oriented ecosphere for plug & play medical systems could be designed to ensure safety. Such an ecosphere would need to establish certification processes and criteria such that system-safety assurance obligations can be appropriately divided between the different system component manufacturers and ecosphere stakeholders.

The premise of our work is that to establish safety of a plug & play system, *application vendors* should be the ones to produce system-level safety arguments. This is because the application vendors will know what clinical scenario their application is targeting (*i.e.,* its *intended use*) and therefore know the system-level safety requirements. These arguments would have to leverage both safety of individual devices and the assurances provided by the platform ecosphere. Our goal is to establish a sound

way of combining component and ecosphere assurances into an assurance case for the application.

There are two contributions of this paper. First, we propose a platform-oriented ecosphere with specific certification processes designed to support the assurance of plug & play medical systems. The second contribution is an assurance argument pattern that exploits the design of our proposed ecosphere. Vendors would use the pattern to construct application assurance cases. We illustrate the use of this pattern by instantiating it for a particular medical system use-case.

The organization of this paper is as follows: In Section 2 we describe an example clinician proposed safety-critical medical control system as a use-case for plug & play medical systems. Section 3 gives an overview of our proposed ecopshere and how that ecosphere should be managed. Section 4 contains the assurance argument pattern and its instantiation for an application that implements the system of Section 2. We give an overview of related work in Section 5 and conclude with some ideas for future work in Section 6.

2 Motivating Example: Clinical Scenario: Patient Controlled Analgesia

One common method used to control patient pain in clinical settings is Patient Controlled Analgesia (PCA). PCA therapy provides consistent control of pain by allowing patients to self-administer doses of an opioid. Typically, a patient is attached to a special infusion pump equipped with a "bolus trigger." When the patient presses the trigger, the pump will deliver a predetermined amount of opioid to the patient. Evidence from systematic reviews of randomized controlled clinical trials indicate that the use of IV PCA leads to better pain relief, improved patient outcomes (e.g., reduction in pulmonary complications) and increased patient satisfaction compared with conventional nurse administered parenteral opioids [13].

One major opioid side effect is respiratory depression. Respiratory depression increases progressively with dose. If respiratory depression increases to the point that the patient's ability to take in oxygen is compromised it is called respiratory distress and serious injury or death can result. In theory, properly configured PCA provides some protection from overdose because it is inherently self-limiting: Patients will usually lose consciousness before respiratory depression reaches dangerous levels which prevents them from requesting further doses and causing an overdose. Additionally, modern PCA-pumps allow clinicians to set limits on the total amount infused per hour as well as lockout intervals between boluses.

Despite these protections, PCA therapy is associated with a large number of adverse events: There were over 9500 cases of PCA related errors voluntarily reported to the Institude of Safe Medicine between 2000 and 2004 alone [11] and there continues to be cases of severe respiratory complications due to PCA [6]. Patients undergoing PCA therapy can still receive overdoses if the pump is misconfigured, if the pump configurer overestimates the maximum dose a patient can receive, if the wrong concentration of drug is loaded into the pump, or if someone other than the patient presses the bolus trigger (known as PCA-by-proxy).

2.1 A Fail-Safe Device Coordination Protocol

Previously Arney *et al.* ([3]) developed a fail-safe device coordination protocol designed to prevent overdose resulting from PCA therapy. The control-loop of the system is illustrated in Fig. 1 and consists of a network-connected PCA pump, pulse-oximeter, and a supervisory controller. The pulse-oximeter periodically transmits measurements of the patient's blood oxygen saturation (SpO_2) to the controller. The controller maintains a "worst-case" model of patient opioid pharmokinetics. This model relates opioid concentration levels to SpO_2 and respiratory depression. This model allows the controller to use SpO_2 to infer the patient's level of respiratory depression. Using the inferred level of respiratory depression, the controller will calculate a control decision and transmit that decision to the PCA pump.

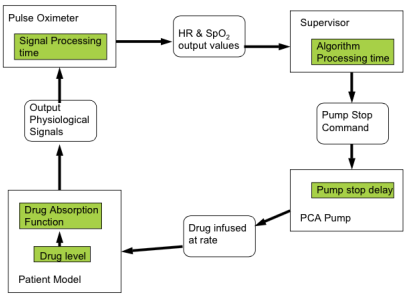

Fig. 1. Control loop with sources of timing delays [3].

Arney *et al.*'s control-loop is fail-safe in the sense that failures of omission (*e.g.,* dropped messages in the network) will not result in an overdose. Fail-safety is accomplished using a ticket-based control strategy. Instead of simply sending the pump activation or deactivation commands, the controller sends a timed-ticket. This ticket encodes the maximum amount of time (called Δt_{safe}) the pump can continuously run without pushing the patient into respiratory distress.

The calculation of Δt_{safe} must take into account the maximum timing delays present in each stage of the control loop (see Fig. 1). Pulse-oximeters use a sliding-window averaging algorithm to calculate SpO_2 from raw sensor data. The size of the window and processing time ensures that the SpO_2 measurement is delayed from the "real" SpO_2 value by several seconds. The network adds delay between both the pulse-oximeter and the controller and between the controller and the PCA pump. The controller introduces some delay because it takes time to calculate the ticket. Finally, there will be a delay from the moment the pump's ticket timer times out and the moment the pump stops because it will take time for the physical pump mechanism to cease infusion.

Arney *et al.* modeled their coordination protocol as a network of communicating Timed Automata [1]: Each component (including network links) illustrated in Fig. 1 were modeled as Timed Automata. Each source of delay was modeled and the network was allowed to arbitrarily drop messages. SpO_2 was used as a proxy for respiratory distress: The patient model included a state variable representing the patient's current SpO_2 value. The value of the variable was periodically calculated via a pharmokinetic model that relates drug concentration in the patient to respiratory rate and SpO_2. The UPPAAL model checker [5] was used to verify that the patient will not experience overdose.

3 A Platform-Oriented Ecosphere

We believe that a platform-oriented ecosphere of medical components, if appropriately designed, could be used to ensure the safety of plug & play medical systems. In this paper, we define an *ecosphere* as as set of devices, software applications and computational platforms intended to interact with one another using standardized plug & play interoperability protocols; the stakeholders that organize, manufacture, and use these products; as well as the explicitly defined processes that are followed to develop, certify, and use these products.

Our proposed ecosphere contains three categories of interoperable system components. The component categories are *device*, *application*, and *platform*. Devices expose a logical interface that acts like an API which applications can use to control or receive data from the device. Applications implement the clinical algorithms used to address a specific clinical scenario. Applications are not just executable code; they have a requirements specification which declares what interfaces compatible devices must implement and a QoS specification that declares timing requirements (*e.g.,* periods and deadlines on program execution). Each platform consists of a network, computational resources (CPU, RAM, *etc.*), real-time operating system and *platform services*. The platform's job is to act as a trusted base to enforce the correct assembly of on-demand systems: When the Health Delivery Organization (HDO) plugs a device into the platform the device will upload its interface specification. Then, when the HDO staff tries to launch an application with a set of selected devices the platform will (1) check if those devices' interfaces are compatible with the application requirements and (2) verify that the application's requested QoS can be guaranteed. If either 1 or 2 is false the platform will prevent application launch.

There are a number of *actors* that participate in our vision of the ecosphere. Each actor has different responsibilities and assurance obligations:

- *The Ecosphere Standards Consortium* consists of representives of the other actors and follows a consensus process to define ecosphere standards: The connectivity protocols used by each component to exchange data, the logical interfaces devices can implement, what it means for a device to be compatible with an application, and the compliance requirements that each type of component (applications, devices, and platforms) must satisfy before that component can be certified as a member of the ecosphere. We emphasize that the consortium does not explicitly define specific systems - rather it establishes constraints on the architecture and interfaces of such systems and their sub-components.
- *The Device Vendor* designs, manufactures, and markets their devices. Before their device can be admitted to the ecosystem they must provide assurance (*e.g.,* via an assurance case) that their device satisfies the ecosphere compliance requirements for all interfaces the device claims it implements.
- *The Application Vendor* is responsible for providing assurance that their application is safe when instantiated with compatible devices. Application vendors play a role analogous to"system integrators" in conventional systems: They define the overall system function, and reason about overall system safety. However, the distinction is that they define the system using a software application and requirements/assumptions on the devices and platforms. They do not specify a single system but a

family of possible system instances that satisfy the functional and safety goals of the clinical scenario. Thus, the integration is "virtual": they do not integrate specific physical devices and platforms but specifications of devices and platforms where each such specification represents a set of compliant components.

- *The Platform Vendor* must provide assurance that their platform will correctly perform its responsibilities. Because the platform is the trusted base for each system these responsibilities include correctly executing application code, correctly implementing the ecosphere device-application compatibility check and providing adequate system security.
- *The Certification Authority* polices component membership in the ecosphere: The certification authority only grants certification to components that satisfy the ecosphere compliance requirements. When a component becomes certified the authority will sign the component with a digital certificate. If postmarket surveillance reveals that a component has a previously undetected problem resulting in non-compliance, the certification authority can revoke the certificate associated with that component's make and model. The digital certificate enables the platforms to use cryptographic methods to verify whether or not applications or devices have been certified. [10] contains an overview of how cryptographic methods can be used to establish trust and how the platform acts as a trusted base for this process.
- *The HDO* does not have assurance responsibilities *per se* (*i.e.,* they are not required to provide assurance to any other ecosphere actor) however, they must still use the application as intended. If the HDO uses an application in an unintended way (*i.e.,* off-label use) then the (safety) assurances provided by the Application Vendor for that application are not guaranteed to apply.

The ecosphere assurance and compliance obligations (combined with the runtime checks performed by the platform) create a series of "gating functions" that prevent the HDO from assembling potentially unsafe combinations of devices and applications. Fig. 2 illustrates the relationship between the ecosphere actors, ecosphere components, the gating functions and the final physical instantiation of a system. The unringed circles indicate steps in development or assembly of the physical system. Lines indicate interactions between the actors. The ringed circles indicate completion of one of the primary assurance steps and represent the gates. First the Ecosphere Standards Consortium must establish the ecosphere standards and component compliance requirements. Once the standards have been defined the component manufacturers can design their respective components. The certification authority enforces the first set of gates: components are only allowed into the ecosphere if they satisfy their respective compliance requirements. The final set of gates are enforced by the platform: The platform will only let the application run if it is being paired with compatible and compliant (*i.e.,* certified) devices and if the platform can guarantee that the application's QoS requirements will be met.

3.1 A Note on Interfaces, Compatibility, and Device Compliance:

For the purposes of this paper we imagine that device and application interfaces are analogous to software interfaces from programming languages like Java: When an application specifies that it requires a device interface it is much the same as declaring

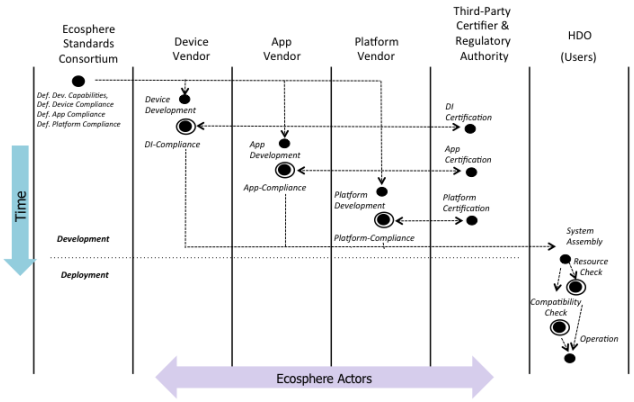

Fig. 2. Ecosphere actors, their interactions and certification activities.

a field variable in a Java class to have an interface-type: Any object that implements that interface can be substituted for that variable. Compatibility checking between devices and applications thus amounts to checking if the device implements the required interface(s). A device is compliant with an interface if it satisfies the Consortium defined compliance requirements for that interface type. Consider the PCA pump from the motivating example. The Ecosphere Consortium could define a standardized interface for PCA pumps called "void InfusionTimedTicket(x)" which applications could use to send a timed ticket to the pump. The Consortium would then define the behavior a PCA pump must have in order to comply with that interface. In this case, the pump should correctly implement the ticket timer and cease infusion after some mandated amount of time.

3.2 A Note on Platform Assurance & Compliance:

A compliant platform must correctly implement compatibility checking and resource management. Ideally, applications would be portable across platforms in the ecosphere. This means that the Consortium would also standardize an execution model for the applications (*i.e.,* application byte code format, semantics, and available APIs). A compliant platform must then also correctly implement the standard model of execution. Different applications will have different levels of criticality: Applications with low criticality do not pose serious consequences in the event of failure while failure of a high-criticality application may result in catastrophic consequences. Because the application is totally dependent on the platform to function correctly, the assurance requirements for each platform should be at least as stringent as the assurance requirements for the most critical application that will be admitted to the ecosphere. While the specifics of these requirements are well beyond the scope of this paper, we can imagine that the

Consortium could mandate that all Platform Vendors follow guidance that would result in levels of assurance similar to that of DO-178C Level A. [12].

4 The Platform Argument Pattern

Each plug & play application defines a *set* of possible systems: One for each allowed combination of devices and platform with the application. The multitude of potential systems implied by a single application presents a challenge for both the application developer and Certification Authority. The application vendor will need to devise an assurance argument that explains why all these possible combinations are safe. Practically, it won't be possible for the vendor to analyze or test each combination individually because the number of possible combinations would prohibitively large. Additionally, new components (*i.e.,* platforms & devices) may be admitted to the ecosphere *after* the application is certified. Because the application vendor will not be able to directly analyze all possible device combinations they will have to use some form of model-based reasoning: They would analyze their application for safety using models as proxies for the concrete devices. In theory, as long as the models capture the range of behavior allowed by the different ecosphere gating functions (*i.e.,* the compliance/certification checks and the platform compatibily checks) then safety conclusions derived from the model-based reasoning should hold for any allowed instantiation of the application.

Of course, in practice, it is generally impossible to capture *all* the allowed behavior of a physical system in a model. If an application vendor is using model-based reasoning to support safety-claims they should justify *why* the models they used are *adequate*. In our context, adequacy depends on the intended use of the application (*i.e.,* the meaning of adequate will vary from application to application) *as well as* the assurances on each component provided by the ecosphere itself. To this end we propose an assurance argument pattern that requires application vendors to make model adequacy arguments explicit. Our hope is that it can help both application vendors and the Certification Authority to quickly identify assurance deficits or other fallacious reasoning in application assurance arguments, especially those related to model-based reasoning. The remainder of this section is organized as follows: First we define the terms the pattern uses. Then we introduce and explain the platform argument pattern. Lastly, we instantiate the pattern to make a mock assurance case for an application implementing the PCA coordination protocol from Section 2.

4.1 Pattern Terms

Fig. 3 maps out the terms used in the pattern. Our terms make an explicit distinction between models[1] and physical embodiments. We ultimately care about the physical embodiments but we are left with the models to analyze. The rows correspond to the

[1] Through out this section we adopt a formal notation that might lead some readers to believe that when we use the term "model" we are explicitly refering to formal models (*i.e.,* ones that could be analyzed by a model-checker). This is not the case. We are using "model" in a very general sense and a model could range from an informal "mental model" to an executable model that could be simulated to a formal model that could be analyzed by a model-checker.

different types of ecosphere components (with the addition of a row for the environment and instantiated system). The columns separate out different abstractions for each of the component categories: The specifications refer to the actual specification artifacts created by either the application developer or device manufacturer, the models are semantic (*i.e.,* analyzable) objects created by the application developer based on the specifications. The last column (physical embodiments) represent the physical object that correspond to the models and specifications.

		Model	Specification	Physical Embodiment
Devices		-	$DI_1, \ldots, DI_l$	$D_1, \ldots, D_l$
App	Algorithm	A^m	A	$P(A)$
	Interface	AI_j^m	AI_j	$\mathbb{D}_j = \{D_i \mid DI_i \simeq AI_j\}$
Platform		-	-	P
Environment		E^m	-	E
System		$A^m \mid\mid_{j=1}^{n} AI_j^m \mid\mid E^m$	$A \mid\mid_{j=1}^{n} AI_j$	$\{P(A) \mid\mid_{j=1}^{n} D_j \mid\mid E \mid D_j \in \mathbb{D}_j\}$

Fig. 3. Pattern Terms: The relationship between models, specifications, and physical embodiments.

Each entity in Fig. 3 is defined as follows: The l devices admitted to the ecosphere are $D_1, \ldots, D_l$. Each D_i is compliant with its interface DI_i, Each application consists of an A and set of AI_j. The A is the algorithm of the application and represents executable code. Since these applications are typically real-time we assume any QoS specifications in the application are contained within A. The AI_js represent the application's required device interfaces (If the application uses n devices then $1 \leq j \leq n$). The physical emobodiment of each AI_j is the set of devices that implement the interface AI_j (we use $\simeq$ to represent the compatibility relation). The AI_j^m are models created by the application developer and are intended to capture all the behaviors of the devices that implement the AI_js. Since A is a program, it has no physical embodiment until it is executed on a platform, therefore $P(A)$ represents platform P executing A. The device interfaces of an application are syntactic objects. They don't have explicit semantics but they do imply a set of behaviors (*i.e.,* the union of the behaviors of all the compliant devices that are compatible with that interface). Each platform is represented by a P. E represents the environment where the application will be deployed and E^m is the model of that environment. The last row are the system entities. $A^m \mid\mid_{j=1}^{n} AI_j^m \mid\mid E^m$ is the model of the system. It is the composition of the application model, the device models, and the environment model (We borrow the parallel composition operator, $\mid\mid$, from process algebras to denote the combination of two or more components running together). $A \mid\mid_{j=1}^{n} AI_j$ (*i.e.,* the application) represents a specification of the system. $\{P(A) \mid\mid_{j=1}^{n} D_j \mid\mid E \mid D_j \in \mathbb{D}_j\}$ is the set of possible physical systems specified by the application (one system for each compatible combination of application and device(s)).

4.2 The Pattern

Fig. 4 is a specification of the argument pattern using Goal Structured Notation (GSN) [14]. The top level goal (**G:AllSat**) states that all instantiantions of the applica-

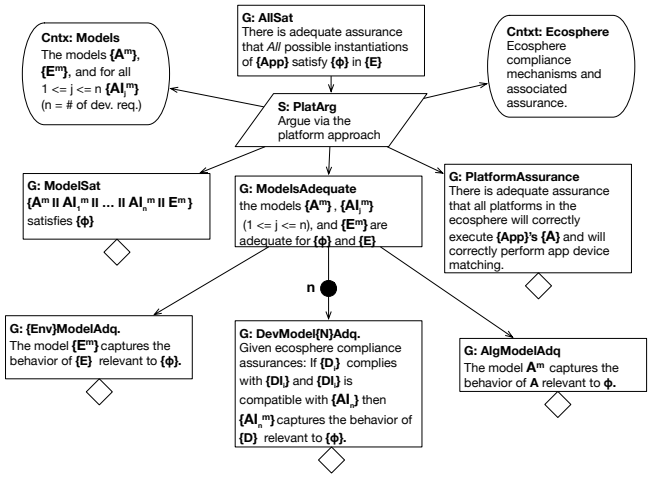

Fig. 4. The argument pattern for application assurance.

tion must satisfy some property ϕ in a specified environment. Assurance for this claim is argued via the platform argument strategy (**S:PlatArg**). The strategy must always be applied in at least two contexts: One referencing the models used in the model-based reasoning and the other referencing the ecosphere assurance and compliance requirements. **S:PlatArg** requires adequate assurance for three sub goals. The first goal (**G:ModelSat**) is the model-based reasoning step. The argument application vendor must argue that the chosen models satisfy ϕ. The remaining two goals explicitly relate the models used in **G:ModelSat** to the possible physical systems via the ecosphere assurance and compliance requirements. **G:ModelsAdequate** asks the developer to argue why the models chosen in **G:ModelSat** capture all the possible (relevant) behaviors allowed by the application's specification. Typically, the arguments for the adequacy of the environment, application, and devices models will all take on a different character so the pattern separates the arguments for each as a different sub-goal (Note the multiplicity on the **G: DevModel{N}Adq.** that forces a sub-goal for each device model). **G:PlatformAssurance** asks the developer to argue why the minimum level of assurance provided by any ecosphere compliant platform is sufficient to support the application: The application developer relies on the platform to correctly execute their application and ensure that the application is only instantiated with compatible devices. If a platform fails to do either of these correctly, then ϕ could be violated even if sound models were used in **G:ModelSat**.

4.3 An Example Assurance Argument

Fig. 5 shows an example assurance argument for an application implementing the PCA device coordination protocol of Section 2. We assume that the application specifies two required interfaces ("void $\mathsf{InfusionTimedTicket}(x)$" and "float $\mathsf{getSpO2}()$"). We also assume that the application code was auto-generated from Arney *et al.*'s con-

troller timed automata model via the TIMES tool [2]. The application would have a QoS specification that requires that all network delays are bounded by 500*ms* and that the controller will take no more than 200*ms* to generate a ticket after it receives a fresh SpO_2 reading. We assume for the purposes of this argument that the Consortium requires all PCA pumps that comply with void $\mathsf{InfusionTimedTicket}(x)$ halt infusion within $100ms$ of the ticket expiring. We also assume that any pulse-oximeter compliant with "float $\mathsf{getSpO2}()$" have an averaging time $\leq 2000ms$. Lastly, we assume that the ecosphere compliance requirements for platforms is very stringent (*i.e.,* generally accepted by domain experts to be adequate for life-critical systems.)

The assurance argument now proceeds as follows: The top-level goal **G: NoOverinfusion** claims that all allowed instantiations of the **PCAapp** prevent over-infusion. **G: NoOverinfusion** is argued by applying the platform argument. The models used in the model-based reasoning step are Arney *et al.*'s timed automata models (in this setting the patient model is the environment model). The system is shown to avoid patient overdose states by verifying its models in the UPPAAL model-checker (using the process described in [3]). The patient model is claimed to be sound because it is an accepted textbook model of opioid pharmokinetics. The device models are argued to be adequate because they capture all allowed behavior of compliant devices (we elide the argument for the PCA model due to space constraints). For example, the model of the pulse-oximeter assumes that all pulse-oximeters compliant with "float $\mathsf{getSpO2}()$" exhibit an averaging time $\leq 2000ms$. This is the exact range of timing behavior captured in Arney *et al.*'s pulse-oximeter model. The application model is argued to be sound because the TIMES tool generated the executable code from the model. Finally, all platforms are argued to have adequate levels of assurance for **PCAapp** because ecosphere platform compliance requires a stringent level of assurance generally accepted to be adequate for life-critical systems.

Of course, just because a developer is able to instantiate the pattern does not mean their system is safe or their argument is good. The Certification Authority or any other reviewer may or may not accept the argument. For example, they could employ a domain expert in opiod pharmokinetics who judges that the textbook pharmokinetic model used in the analysis is not adequate or too simplistic. Or perhaps the application vendor had erroneously interpreted the compliance requirements for pulse-oximeters implementing"float $\mathsf{getSpO2}()$" and some compliant pulse-oximeters may in fact have an averaging time $> 2000ms$ (in which case the app's ticket calculation would be wrong). The point is that, by making the soundness argument explicit it helps both the application vendor and Certification Authority more quickly identify assurance deficits.

5 Related Work

The study of assurance and ecospheres for plug & play systems is relatively new. ASTM F2761 [19] is a standard that defines the Integrated Clinical Environment (ICE) out of work started at the Medical Device Plug & Play Interoperability Program at CIMIT [7]. ASTM F2761 abstractly defines a medical application platform and alludes to how the platform could support an ecosphere of plug & play medical devices. More recently, [9] described how a medical application platform would facilitate the safe integration of

applications and medical devices drawn from an ecopshere of interoperable componets. OpenICE [18] and the MDCF [16] are both prototype medical application platforms and have been used to inform both academic and industry research on plug & play medical systems.

While, as far as we know there has not been any work on assurance arguments for plug & play systems, there has been some work on assurance arguments for model-based development [4]. The authors of [4] describe an assurance argument pattern for systems developed using a model-based development process. Like our proposed pattern, their pattern requires that the argument preparer to first prove a propery using a model, and then justify the use of that model. Their pattern does not address the peculiarities of model based reasoning for plug & play systems. There has been some interesting work on modular certification [20] and compositional safety arguments [15]. These works are primarily concerned with argument reuse but introduce some concepts that may be applicable to providing assurance for plug & play systems.

6 Conclusions and Future Work

This paper described two contributions. First, we proposed a platform-oriented ecopshere designed to support the assurance of plug & play medical systems. Second, we described an assurance argument pattern that exploits the design of this ecosphere. The key challenge we sought to address is that, unlike traditional systems, plug & play systems do not fully exist until they are assembled by their users. Our approach leverages a specially managed ecosphere of components that enables application vendors to constrain which combinations of devices can be used with their application. This puts the application developer to be in the unique position of being able to use model-based reasoning to predict the possible behaviors of the allowed instatiations of their application. Our proposed assurance argument pattern explicitly links the model-based reasoning performed by the vendor to assurances provided by the ecosphere. We illustrated the use of this pattern via a small case-study of a closed-loop medical system. For future work we would like to apply this approach to a more involved case-study. One objective of this case-study could be to submit a mock-submission to a regulatory agency (*e.g.,* the FDA) and then report on the feedback received. It would also be interesting to explore how approaches for argument reuse (*e.g.,* [20] or [15]) could be incorporated into our proposal.

Acknowledgements This research was supported in part by NSF CNS-1035715, NSF CPS 1239324, NIH 1U01EB012470-01, and DGIST Research and Development Program of the Ministry of Science, ICT and Future Planning of Korea (CPS Global Center).

References

1. Alur, R., Dill, D.L.: A theory of timed automata. Theoretical computer science 126(2), 183–235 (1994)

2. Amnell, T., Fersman, E., Mokrushin, L., Pettersson, P., Yi, W.: Times: A tool for modelling and implementation of embedded systems. In: Tools and Algorithms for the Construction and Analysis of Systems, pp. 460–464. Springer (2002)
3. Arney, D., Pajic, M., Goldman, J.M., Lee, I., Mangharam, R., Sokolsky, O.: Toward patient safety in closed-loop medical device systems. In: Proceedings of the 1st ACM/IEEE International Conference on Cyber-Physical Systems. pp. 139–148. ACM (2010)
4. Ayoub, A., Kim, B., Lee, I., Sokolsky, O.: A safety case pattern for model-based development approach. In: NASA Formal Methods, pp. 141–146. Springer (2012)
5. Behrmann, G., David, A., Larsen, K.G., Hakansson, J., Petterson, P., Yi, W., Hendriks, M.: Uppaal 4.0. In: Quantitative Evaluation of Systems, 2006. QEST 2006. Third International Conference on. pp. 125–126. IEEE (2006)
6. Bonner, J., McClymont, W.: Respiratory arrest in an obstetric patient using remifentanil patient-controlled analgesia*. Anaesthesia 67(5), 538–540 (2012)
7. Goldman, J.: Advancing the adoption of medical device plug-and-play interoperability to improve patient safety and healthcare efficiency. Medical Device "Plug-and-Play" Interoperability Program, Tech. Rep (2000)
8. Goldman, J.M.: Getting connected to save lives. Biomedical Instrumentation & Technology 39(3), 174–174 (2005)
9. Hatcliff, J., King, A., Lee, I., MacDonald, A., Fernando, A., Robkin, M., Vasserman, E., Weininger, S., Goldman, J.M.: Rationale and architecture principles for medical application platforms. In: Cyber-Physical Systems (ICCPS), 2012 IEEE/ACM Third International Conference on. pp. 3–12. IEEE (2012)
10. Hatcliff, J., Vasserman, E., Weininger, S., Goldman, J.: An overview of regulatory and trust issues for the integrated clinical environment. In: Proceedings of HCMDSS 2011 (2011)
11. Hicks, R.W., Sikirica, V., Nelson, W., Schein, J.R., Cousins, D.D.: Medication errors involving patient-controlled analgesia. American Journal of Health-System Pharmacy 65(5), 429–440 (2008)
12. Hilderman, V., Baghi, T.: Avionics certification: a complete guide to DO-178 (software), DO-254 (hardware). Avionics Communications (2007)
13. Hudcova, J., McNicol, E.D., Quah, C.S., Lau, J., Carr, D.B.: Patient controlled opioid analgesia versus conventional opioid analgesia for postoperative pain. The Cochrane Library
14. Kelly, T., Weaver, R.: The goal structuring notation–a safety argument notation. Dependable Systems and Networks Workshop on Assurance Cases (2004)
15. Kelly, T.P.: Concepts and principles of compositional safety case construction (2001)
16. King, A., Procter, S., Andresen, D., Hatcliff, J., Warren, S., Spees, W., Jetley, R., Jones, P., Weininger, S.: An open test bed for medical device integration and coordination. In: Software Engineering-Companion Volume, 2009. ICSE-Companion 2009. 31st International Conference on. pp. 141–151. IEEE (2009)
17. Leveson, N.: A new accident model for engineering safer systems. Safety science 42(4), 237–270 (2004)
18. Plourde, J., Arney, D., Goldman, J.M.: Openice: An open, interoperable platform for medical cyber-physical systems. In: Cyber-Physical Systems (ICCPS), 2014 ACM/IEEE International Conference on. pp. 221–221. IEEE (2014)
19. Quigley, P.: F2761 and the integrated clinical environment. Standardization News 37(5), 20 (2009)
20. Rushby, J.: Modular certification. Tech. rep., SRI CSL (Sep 2001)

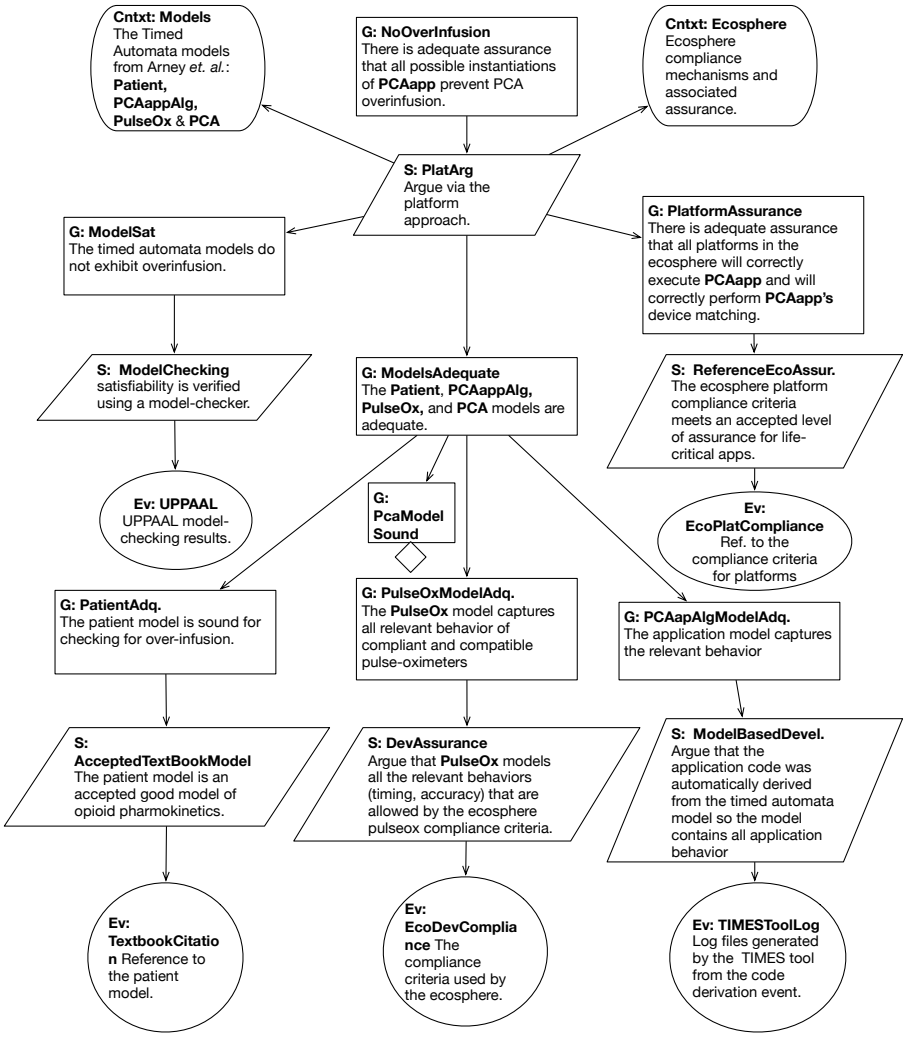

Fig. 5. Assurance case fragment for the PCA-Control Application