Runtime Verification for Interconnected Medical Devices*

Martin Leucker, Malte Schmitz, and Danilo à Tellinghusen

Institute for Software Engineering and Programming Languages,
Universität zu Lübeck,
{leucker,schmitz,tellinghusen}@isp.uni-luebeck.de

Abstract. In this tool paper we present a software development kit (SDK) for the Open Surgical Communication Protocol (OSCP) that supports the development of interconnected medical devices according to the recent IEEE 11073 standards for interoperable medical device communication. Building on service-oriented architecture (SOA), dynamically interconnected medical devices publish their connectivity interface, via which these systems provide data and can be controlled. To achieve the safety requirements necessary for medical devices, our tool, the OSCP Device Modeler, allows the specification of temporal assertions for the respective data streams of the systems and generates automatically corresponding monitors that may be used during testing, but also during the application in field to ensure adherence to the interface specification. A further tool, the OSCP Swiss Army Knife, allows subscribing to the services provided via the interfaces of the system under development and thereby supports its debugging. The whole OSCP SDK makes heavy use of runtime verification techniques and shows their advantages in this application area.

1 Introduction

To enhance the overall functionality of medical devices, their interconnection is a current trend. Within the OR.NET project[1], an open protocol for the interconnection of medical devices has been developed and standardized and is expected to become a typical solution for the communication among medical devices [3].

As in many cases whenever humans' life may depend on the correct functionality of a device, there are strict rules for its development and most medical devices have to be certified before their operation in the field is allowed.[2] In this paper we present a software development kit (SDK) comprising two tools simplifying the development of safe and reliable medical devices. The first tool, called MD Modeler, helps in defining a clear and precise (network) interface of

* This work is supported in part by the European Cooperation in Science and Technology (COST Action ARVI), the BMBF project CONIRAS under number 01IS13029, and the BMBF project OR.NET under number 16KT1231.

[1] www.ornet.org

[2] Strictly speaking, a medical device with high criticality level has to be declared as conformant to the underlying medical device by its manufacturer with consultation of a so-called notified body checking that the conformance declaration follows the rules. For simplicity, we use the term *certification* here anyway.

the system under development as well as giving evidence that it adheres to given correctness properties. The second tool, called OSCP Swiss Army Knife, helps in debugging the medical devices via its network interface.

The communication solution developed within OR.NET is called the *Open Surgical Communication Protocol (OSCP)*, comes in three different layers and basically builds on top of a service-oriented architecture (SOA) with an implementation in terms of web services. The main idea is that devices within the network of the medical units publish their services and clients may connect via well-defined interfaces. Via these interfaces the corresponding devices can also be controlled. As for any interface – let it be human or let it be device-driven – a risk analysis has to be performed when developing medical devices. The risk analysis together with corresponding risk control measures ensures that the corresponding device is most likely only used in the intended fashion [6].

To ensure such a controlled usage of the medical device, we have proposed

1. that the interface for medical devices is formalized in a precise manner together with (temporal) constraints putting additional restrictions on its usage and
2. a monitoring layer ensuring the adherence to the given constraints plus fallback mechanisms whenever a mismatch between the actual usage at runtime and the constraints is detected [9,13].

The first tool of our SDK, the OSCP Device Modeler, helps in realizing this concept. Its web-based interface allows the specification of both the interface definition as well as the additional constraints. Moreover, a Java skeleton for the interface plus corresponding monitoring code can be synthesized automatically simplifying both the development of connective medical devices as well as the monitoring layer considerably. As the monitoring layer is synthesized automatically from high-level specifications it is expected that a certification of the resulting system is simplified a lot, since we realized a clear separation of concerns: the concern of checking the right properties as well as having the right code for checking the properties. To the best of our knowledge no such tool has existed before. Moreover we believe that our tool is the first targeting the simplification of certification by using runtime verification techniques.

The second tool in our SDK, the OSCP Swiss Army Knife, is a network-based debugging tool. It allows to join an existing connected set of medical devices and to show their interfaces as well as the current values of the systems participating in the network. Using the published information about the methods for changing the system's parameters it also allows to steer the corresponding devices. The distinguishing feature of our debugging tool, however, is its ability to check for temporal correctness properties. Interconnected devices, such as medical devices, often follow a sequence of protocol steps for the exchange of data. Likewise a debugging tool should support checking for the correctness of such execution sequences. The OSCP Swiss Army Knife allows the formulation of such correctness properties in temporal logic at runtime and to synthesize monitors that are deployed in the network consisting of the connected medical devices and will then check corresponding properties.

Such monitors may be used to find bugs in the system but may also be used to identify points of interests in the execution sequence of the systems. We are not aware of any similar tool that allows the specification of temporal patterns to identify these points of interest at runtime. While on the one hand it is limited to examining the system via its network interface, on the other hand our method does – in contrast to many existing runtime verification approaches – not require the re-compilation of the system under test and does not interfere with its main functionality or its timing behavior.

The paper is organized as follows:

- section 2 describes the communication protocol and the basics of our runtime verification approach and
- section 3 and section 4 introduce the two main tools of our SDK.

2 Preliminaries

OSCP consists of the data transmission technology Medical Device Profile for Web Services (MDPWS), standardized as IEEE 11073-20702 (see [7]), and the domain information and service model, standardized as IEEE 11073-10207 (see [8]). MDPWS is based on a SOA and allows devices to find each other in a local network using WS-Discovery.

The domain information provides a generic framework which is used to describe configuration parameters and measured values of a medical device in terms of physical quantities and units. A medical device publishes this description along with the current values in the network. Clients can subscribe to changes of these values and are notified either periodically after a specific amount of time or episodically for every change. Furthermore the service model is used to describe how clients can control a medical device through its public interface.

As already stated in the introduction, we want to assert the correct temporal behavior of the medical devices regarding their network interfaces. The devices' state changes over time can be seen as a sequence of states. We call such a sequence the run of a medical device. We want to express correctness properties regarding the run of a medical device and monitor at runtime whether the interface of a medical device fulfills this correctness property. We call a finite prefix of the run an execution of the medical device, which grows with every new state. Doing runtime monitoring means we want to create a monitor which tells us at every step whether the current execution satisfies the property [10,12].

Our tools support several temporal logics for the specification of the correctness properties:

- ω-regular expressions,
- Linear Temporal Logic (LTL, [14]),
- Smart Assertion Logic for Temporal Logic (SALT, [1]), which adds syntactic sugar to LTL and
- regular LTL (RLTL, [11,15]), which offers the expressiveness of regular expressions while being as applicable to the domain of temporal specifications as LTL.

In order to evaluate the properties on executions, we adopt the LTL_3 semantics [2] to all the logics mentioned before. Hence our monitors report fulfillment or violation of the correctness properties as soon as possible while the execution continuously grows.

Each of the available logics uses atomic propositions as basic building blocks. Our tools allow the specification of these propositions as comparison of variables published by the medical device either with constants or other variables of the same or another medical device. New states of the execution are created based on variables changing their values. Hence along with a specification formula we need to define the list of its change-inducing variables.

3 OSCP Device Modeler

The OSCP device description provides a generic formalism to describe the (network) interface of a medical device. We enrich this description with (temporal) constraints putting additional restrictions on its usage. At runtime the device needs to check the adherence to these constraints.

The Device Modeler supports the process of developing network interfaces for medical devices: It provides a graphical user interface to design a MD description and it generates source code which publishes the interface description and the current values to the network. The generated code is also capable of handling incoming requests changing the devices parameters and controlling the device and contains synthesized monitors continuously enforcing the (temporal) constraints by rejecting requests which would lead to invalid states. The generated code can either be used as network interface for a real medical device or without further need of writing any code as a stand-alone simulator.

The Device Modeler is a web application written in Ruby on Rails. It is equipped with a modern HTML5 front end guiding the user through the definition of the device description (see Figure 1, left). Users can create new devices from scratch or import existing XML serializations of device descriptions. The user input is stored in an internal database, which allows users to store and maintain their device modelings. The back end uses Rails' default template engine ERB to generate Java code using the OSCP library OpenSDC[3], which is the reference implementation for the OSCP standard [5,4].

Along with the device description, the user can specify (temporal) correctness properties for the device in the front end of the Device Modeler. These correctness properties are synthesized in monitors: We use our library LamaConv[4] to translate the specified formula into a deterministic Moore machine using an adopted LTL_3 monitor synthesis [2]. The current valuation for all propositions used in the formula serves as input for the monitor.

In case of any change of the devices values, the monitor compares the updated values with the change-inducing variables of its formula. If the change induces a new state as described in the previous section, the monitor computes the

[3] `sf.net/projects/opensdc/` [4] `www.isp.uni-luebeck.de/lamaconv`

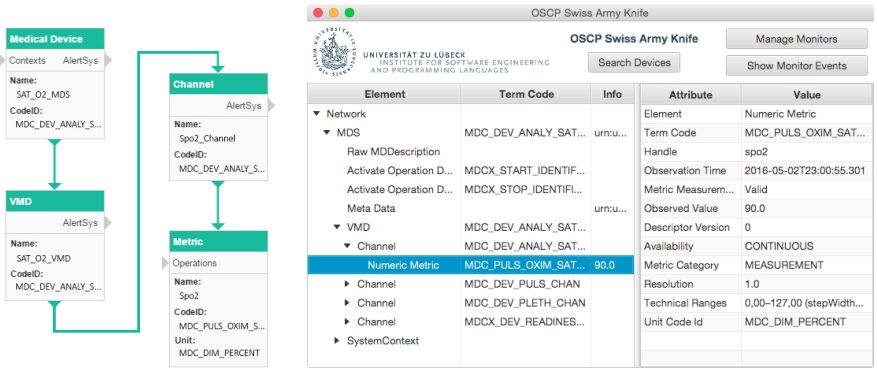

Fig. 1. Screenshot of modeling a pulse oximeter in the Device Modeler's GUI (left) and inspecting the running pulse oximeter with the Swiss Army Knife (right).

valuations for all its propositions using the current properties of the model. These valuations are then used as input for the Moore machine. The output of the monitor can naturally be used for debugging and logging purposes, but furthermore the user of the Device Modeler can also demand that the value changes of the defined metrics must fulfill the specified correctness properties. Any change either through the local UI or the network layer that would break the correctness property will be rejected in this case.

4 OSCP Swiss Army Knife

While the Device Modeler allows us to automatically synthesize monitors when generating the code for the network interface of the device, it is also desirable for debugging purposes to synthesize monitors checking the correct behavior of the interconnected devices at runtime without restarting the system. The Swiss Army Knife is a generic client which can connect to all available medical devices in an OSCP network. It allows the user to inspect the devices' descriptions and current states as well as watch the state changes over time, either manually or with synthesized monitors checking the adherence of the system to user defined correctness properties.

In order to do this, the Swiss Army Knife subscribes to all devices available in the local network using the OSCP library OSCLib[5]. It is written in Scala and uses the C++ library OSCLib through Java SWIG[6] wrappers. The GUI is written in ScalaFX[7] and consists of a tree table view displaying the hierarchical device descriptions of all available devices (see Figure 1, right). The values of the devices are automatically updated in the GUI by callbacks registered with the OSCLib. Attributes of interest for the monitoring are stored in JavaFX properties which are updated along with the GUI in the OSCLib callbacks. This

[5] `www.surgitaix.com/cms/osclib` [6] `www.swig.org` [7] `www.scalafx.org`

way multiple monitors can easily observe the same property and receive its values in case of any change.

A monitor observes the JavaFX properties of its change-inducing variables in order to generate a new execution state every time any of them change. Then the monitors behave the same as the one generated by the Device Modeler described in the last section. They evaluate the propositions based on the current values of the involved properties. Each event handled by a monitor is displayed on the GUI in the event log together with the current valuation of the watched variables and the current output of the monitor is displayed in the list of monitors.

As an example consider a foot switch controlling a pump. If the foot switch's value changes from `OPEN` to `PRESSED`, the pump's state must either change from `ON` to `OFF` or from `OFF` to `ON`. Note the usage of the SALT operator `nextn[2]` which translates to two nested next operators in LTL.

```
assert always (("@switch.value = OPEN" and next "@switch.value = PRESSED") implies
   ((next "@pump.value = ON" and nextn[2] "@pump.value = OFF") or
    (next "@pump.value = OFF" and nextn[2] "@pump.value = ON")))
-- on change of switch.value and pump.value
```

The above example shows how the Swiss Army Knife can be used as a non-invasive inspection tool in order to monitor the correct behavior of the connected medical devices. Furthermore this generic client is able to manipulate the configuration parameters and take control over the available devices. Such invocations trigger changes of the published values of the devices as well, which are again recognized by the monitors. This way one does not have to wait for edge cases to occur in order to monitor them, but can induce them manually.

The defined monitors can be edited at any time and can be activated and paused independently. As described in the section on runtime verification above the monitors can be defined in regular expressions, LTL, RLTL and SALT. The editor supports syntax highlighting, auto-completion and in-place error annotations. All defined monitors are automatically stored in a simple XML serialization and restored with every program start.

The Swiss Army Knife will be made available for evaluation, education and teaching purpose[8].

5 Conclusion

In this paper we have presented the OSCP SDK that simplifies implementation and testing of medical devices using communication libraries OpenSDC and OSCLib. Its two tools, the OSCP Device Modeler and the OSCP Swiss Army Knife, allow the specification of correctness properties using benefits of the well-known established techniques of LTL_3 and SALT and make use of runtime verification to ensure safe interconnection of medical devices. The OSCP Swiss Army Knife allows the user to synthesize and use monitors for debugging at runtime while the OSCP Device Modeler adds monitors to the devices which adds an additional safety layer to their network interface. Thus, both tools use runtime verification in practical applications adding value for industrial users.

[8] `www.isp.uni-luebeck.de/oscp`

References

1. Bauer, A., Leucker, M.: The Theory and Practice of SALT. In: NASA Formal Methods. LNCS, vol. 6617, pp. 13–40. Springer (2011)
2. Bauer, A., Leucker, M., Schallhart, C.: Runtime Verification for LTL and TLTL. ACM Transactions on Software Engineering and Methodology 20(4), 14:1–14:64 (2011)
3. Birkle, M., Bergh, B.: OR.NET: Ein Projekt auf dem Weg zur sicheren dynamischen Vernetzung in OP und Klinik. In: Jahrestagung der Gesellschaft für Informatik e.V. (GI). vol. 208, pp. 1235–1236. GI (2012)
4. Gregorczyk, D., Bußhaus, T., Fischer, S.: Systems, Signals and Devices (SSD). In: SDD. pp. 1–6. IEEE (2012)
5. Gregorczyk, D., Fischer, S., Busshaus, T., Schlichting, S., Pöhlsen, S.: Workshop on Medical Cyber-Physical Systems. In: MedCPS. OASIcs, vol. 36, pp. 15–27. Dagstuhl (2014)
6. Johner, C., Wittorf, S., Hölzer-Klüpfel, M.: Basiswissen Medizinische Software. dpunkt.verlag (2011)
7. Kasparick, M., Schlichting, S., Golatowski, F., Timmermann, D.: Medical DPWS: New IEEE 11073 standard for safe and interoperable medical device communication. In: Standards for Communications and Networking (CSCN). pp. 212–217 (Oct 2015)
8. Kasparick, M., Schlichting, S., Golatowski, F., Timmermann, D.: New IEEE 11073 standards for interoperable, networked point-of-care Medical Devices. In: IEEE Engineering in Medicine and Biology Society (EMBC). pp. 1721–1724 (Aug 2015)
9. Kühn, F., Leucker, M.: OR.NET: Safe Interconnection of Medical Devices (Position Paper). In: FHIES. vol. 8315, pp. 188–198. Springer (2013)
10. Leucker, M.: Teaching Runtime Verification. In: Runtime Verification (RV). Springer (2012)
11. Leucker, M., Sánchez, C.: Regular Linear Temporal Logic. In: Theoretical Aspects of Computing (ICTAC). LNCS, vol. 4711, pp. 291–305. Springer (2007)
12. Leucker, M., Schallhart, C.: A Brief Account of Runtime Verification. The Journal of Logic and Algebraic Programming 78(5), 293–303 (2009)
13. Leucker, M., Schmitz, M.: Secured SOA for the Safe Interconnection of Medical Devices (Position Paper). In: Software Engineering (SE). CEUR Workshop Proceedings, vol. 1337, pp. 11–14. CEUR-WS.org (2015)
14. Pnueli, A.: The Temporal Logic of Programs. In: Foundations of Computer Science (FOCS). pp. 46–57. IEEE Computer Society (1977)
15. Sánchez, C., Samborski-Forlese, J.: Efficient Regular Linear Temporal Logic Using Dualization and Stratification. In: Temporal Representation and Reasoning (TIME). pp. 13–20. IEEE Computer Society (2012)