Self-adaptive Middleware for ubiquitous Medical Device Integration

Andreas Kliem, Anett Boelke, Anne Grohnert, Nicolas Traeder
Technische Universität Berlin
Email: {firstname}.{lastname}@tu-berlin.de

***Abstract*—E-Health systems need to dynamically integrate a huge variety of medical sensors in order to provide a meaningful survey of a patient's condition. Devices like smart phones or gateways usually used as integrators, often underlie resource constraints and have to cope with the mobility of the patient. Therefore it is difficult to realize an overall integration middleware, that allows to handle a sufficient amount of medical sensors and is able to quickly adapt to changing requirements (e.g. new sensors or data aggregation modules) while preserving mobility and resource constraints. We present a middleware solution for the integration of medical devices and the aggregation of resulting data streams, that is able to adapt itself to the requirements of patients and Care Delivery Operators, using a modular approach and external knowledge repositories. Knowledge in the shape of configurations and runtime pluggable software modules is used to properly integrate and handle discovered medical devices on demand.**

***Keywords*-E-health; telemedicine; self-adaptive; dynamic reconfiguration; medical device integration; interoperable medical devices; plug-and-play**

I. Introduction

The evolution of mobile embedded devices in recent years encourages their widespread adoption in the healthcare domain [1]. Especially domains like intensive care or telemedicine are proposed to benefit from ICT based sensors and communication networks [2] [3] [4]. A typical application domain is the monitoring of vital signs with (mobile) sensors connected through Body Area Networks (BAN) or Personal Area Networks (PAN) [5]. Basically, the aim of such systems is to record and analyze the streams of medical data emitted by the sensors in order to support physicians in their decision making process. However, proper decision making is based on a meaningful survey of a patient's condition, which requires to consider a huge variety of typically heterogeneous medical sensors and devices [6]. Moreover, treatment decisions often have to be made under time constraints, which constitutes the need for an aggregated view of the available data streams generated close to the data sources. Each stream utilized can differ regarding its specific characteristics, such as real time requirements, used data formats and nomenclatures or, the communication protocol used by the medical devices that provides the data stream [7].

The resulting device integration and data aggregation problems are highly heterogeneous, require customized and application related software components and therefore, often lead to proprietary solutions. A lot of vendors already provide e-Health solutions especially in the area of vital signs monitoring and telemedicine. However, these systems are often based on closed boxes, integrate only a limited set of medical devices and sensors, usually from the same vendor or based on the same (proprietary) protocol specification, and barely interconnect with each other. Using proprietary solutions, medical device vendors or system integrators gain market exclusivity, which often forces Care Delivery Operators (CDOs) to be dependent on a vendor (i.e. vendor lock-in). Proprietary solutions hinder the development of open and fully integrated e-Health systems, which are required to efficiently deliver cost-effective health services. The problem is intensified, if we have a look at the nature of nowadays treatment processes, that usually involve a lot of different CDOs (e.g. family doctor, specialist, hospital). Since each CDO might rely on different solutions, interoperability cannot be achieved and on-demand access to the medical devices of a patient is difficult to realize. Due to the aforementioned variety, interoperability in the e-Health domain requires, that medical devices can be integrated at any location on-demand, regardless of the protocols or data formats (proprietary or standard-based) they are based on.

It is often depicted, that the device integration problem can be mitigated by standardization efforts. Appropriate standards like ISO/IEEE 11073 (x73) [8] or the Bluetooth Health Device Profile (HDP) [9] exist, but it is unlikely to achieve a widespread standardization in a reasonable time span. Moreover, a lot of standards allow for vendor defined extensions and in particular vendors of complex sensors and medical instruments often rely on pure proprietary protocols to protect their innovations. This raises the question, how a medical device integration middleware can be designed, that:

- is able to handle both standard and proprietary devices
- is adaptable to cope with the rapidly changing requirements (decreased time to market, new sensors, updated data formats and profiles)
- allows to handle all the existing medical devices and preserves interoperability at application level if they are replaced (which extends the integration with a migration challenge)
- can be deployed on resource constrained devices like smart phones or gateways usually used as integration systems [10]

Given these constraints, it is impossible to integrate all the software components required to handle the variety of medical

978-1-4799-6644-8/14/$31.00 ©2014 IEEE

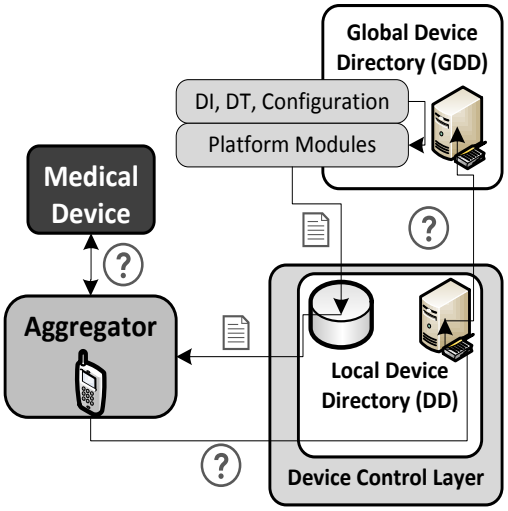

Fig. 1: Overview of a device directory knowledge request

devices and the resulting data streams into one middleware. The middleware needs to be able to adapt itself to the requirements of the environment. It has to recognize the environment by detecting currently available devices on-demand and adapting itself in order to handle the devices properly. Knowledge required to cope with so far unknown devices has to be gathered dynamically from external repositories and injected during runtime, which keeps the amount of actual required resources low, since only software components currently in use have to be maintained. Therefore, the approach we like to present relies on a modular, OSGi based [11], device integration and data aggregation middleware, that allows to integrate and process data from unknown devices by loading respective integration and aggregation modules from external repositories (i.e. adapting itself).

The rest of the paper is structured as follows. Section II gives an overview of fundamentals to the solution, such as the layout of the external knowledge repositories or the relation of semantic interoperability to the framework. Section III will explain the architecture of the middleware and its relevant components. Finally Section IV will highlight some related work and Section V will conclude the paper.

II. Background

This section will introduce the *Device Directory (DD)*, which implements the knowledge repository used by the middleware to gather required information about medical devices to be integrated. Additionally, a brief introduction to the fundamentals of moving medical devices between CDOs and its relation to the DD approach is given. Throughout the following sections, the device that hosts the middleware and acts as the medical device integrator will be called *Aggregator*.

A. Device Directory

The *Device Directory (DD)* can be envisioned as a directory service for devices. It hosts knowledge, that allows the *Aggregator* to reconfigure itself in case an unknown medical device is discovered and needs to be integrated. Therefore, basically three entities are known and maintained by the DD:

- **Device Type (DT):** Refers to the type of a medical device manufactured by a certain vendor. It stores information and configurations that are applicable to all medical devices of that type and has dependencies to a set of *Platform Modules (PM)* required to properly handle the device (i.e. integrate and process data)
- **Device Instance (DI):** Refers to a concrete physical device used by a patient. An instance always belongs to a DT. It holds information and configurations that are only applicable to the instance (e.g. security credentials required to connect to the device). The relation between device instance and type can be envisioned as the relation between serial number and product number.
- **Platform Module (PM):** Refers to a software component (i.e. module or OSGi bundle) that can be requested from the DD and deployed to the middleware at runtime. Several categories like device driver modules, discovery modules or, data transformation modules are supported. Each module has to align to the specifications of the middleware.

Each of these entities can have several attachments (e.g. specific files or software components) and configuration entries, that describe how to handle a medical device and allow to customize the behavior of the middleware's core components or the PM's. Basically, upon discovery of a medical device, the *Aggregator* would request all required knowledge using a DI identifier. This allows the DD to identify the corresponding DT and all required PM's and return them as a result to the *Aggregator*. More details about PM instantiation and linkage to the data streams is given in Section III.

Requesting knowledge of a certain DI from the directory leads to the question how the knowledge is registered. As shown in Figure 1, we assume that a hierarchy of *Device Directories* exists. Each CDO using the middleware framework operates its own local DD instance. The root instance to be operated by a third party is called *Global Device Directory (GDD)* and maintains knowledge about each medical device available. New medical device introduced, have to be registered with the GDD. If a vendor releases a new DT, he has to register the DT along with all mandatory PM's (this task can also be carried out by third parties like system integrators or the CDOs themselves). Additional PM's can be added later by each participant. If knowledge about an unknown DI is requested from a local DD, it checks if the DI is known by the GDD and starts a synchronization process.

Besides knowledge about medical devices, the DD provides a locking mechanism to dynamically bind a device to an aggregator. If we assume, that medical devices shall be integrated on-demand at every required location (i.e. the location the patient carrying the medical devices currently stays), a mechanism that allows to bind a medical device to an *Aggregator* dynamically, while avoiding access conflicts, is required. Each CDO is likely to host its own *Aggregators*,

which implies that *Aggregators* are changed during a treatment process and requires dynamic bindings. The mapping to the actual patient has to be maintained by the CDO (e.g. on the basis of the locks). As each entity in the DD allows to store a custom set of configuration keys and attachments, the patient mapping can also be stored within the local device directory and marked as private, which prevents the DD from synchronizing this information with other DD's (e.g. due to privacy constraints).

B. Semantic Interoperability

Semantic interoperability is an important challenge for the integration middleware and the e-Health domain in general [12], because it allows to integrate all kind of medical devices using different protocols and data formats. Semantic interoperability refers to a common understanding of the incoming data streams among all actors in a system [13] and is based on a common data model that serves as a mediated super set of all other models. A common nomenclature and data format give the opportunity for interoperable applications that are not affected by the replacement of a medical device. As mentioned, it is unlikely to achieve this at device integration layer (e.g. by constraining each vendor towards one standard), which also would stand in contrast to the concept of our middleware (i.e. supporting both standard-based and proprietary devices). However, the benefits at application layer can still be achieved using data transformation approaches. Therefore, the *Device Directory* allows to store PM's of the type data transformation module, that are able to intercept a certain data stream and transform the payload to a canonical format. As the set of PM's attached to a DT is customizable by each CDO (i.e. each local DD), the resulting format can be arranged according to the CDOs requirements and even differ from device to device. However, our prototypes are based on the format introduced by the x73 family of standards, which already provides a rich nomenclature and is currently promoted by the industry (e.g. Continua Health Alliance) [14] [15] and research activities [16] as well.

The approach behind the transformation modules can be referred to as template mapping [17]. Compared to ontology mapping approaches, for instance, template mapping requires providing multiple pairs of templates and their mappings. However, the approach is more lightweight [18] [19], gives a better modularity and therefore fits to our general system model, because it allows to split the overall problem into a lot of lightweight modules directly related to a certain device type and output format.

C. Medical Device Sharing

Referring to the introduction, one major motivation is that treatment processes nowadays include several CDOs. This implies that the medical devices of a patient need to be shared among CDOs in order to access them on-demand at any location. Currently, medical devices are often bound statically to a patient and the CDO owning the device. This leads to an inefficient utilization and higher costs, because each CDO participating in the treatment process has to re-equip the patient with own medical devices to get an overview of its vital signs, regardless whether the patient is already equipped with fitting ones. The introduced hierarchy of *Device Directories* allows us to establish the notion of collaborating CDOs that constitute a federated medical device cloud [20].

Each participating CDO can offer a set of its devices to other CDOs by extending the locking mechanism to a medical device access negotiation protocol used between the DD's. If an *Aggregator* discovers a medical device not owned by its operating CDO, the corresponding knowledge request to the local DD results in a search request to the global DD. Besides all required knowledge to integrate the device, the global DD delivers information about the owning CDO. This allows the local DD serving the current discovery request to contact the device owning DD. The local DD's then negotiate access to the device on a peer-to-peer basis. Security and privacy issues of this approach are introduced in [20].

III. Architecture

The architecture of the device integration and aggregation middleware is based on a modular design using the OSGi specification [11], which allows us to deploy *Platform Modules (PM)* during runtime to the system. First prototypes of the middleware and the *Device Directory* discussed in the previous section were already implemented in scope of the RehaInterAct project [21], which aims to develop a sensor-based interaction- and communication-platform for the physical therapeutic attendance of rehabilitation exercises in clinical and domestic environments.

The middleware architecture shown in Figure 2 can be seperated into two layers, the device integration and the aggregation layer, whereas each layer is responsible to handle different types of PM's. Following types of PM's are available:

- **Discovery Module:** is used by the device integration layer to handle the discovery process. Generates a *Discovery Record* that is required to identify the *Device Instance (DI)* and allows to request required modules and configurations from the *Device Directory (DD)*.
- **Device Driver Module:** is used by the device integration layer to integrate a discovered DI. Implements the application layer communication protocol to exchange messages with the medical device and record medical data from it.
- **Data Transformation Module:** refers to the semantic interoperability feature of the aggregation layer. These modules can intercept a medical data stream and transform it to a canonical data format.
- **Data Utilization Module:** is used by the aggregation layer. Performs any kind of processing (e.g. visualization, data analysis) on a data stream.
- **Data Output Module:** is used by the aggregation layer to transmit data streams to other nodes (e.g. other aggregators, CDO backend or clinical information system)

Common to both layers is a system component (i.e. system service) called DD service, which is the interface to the *Device*

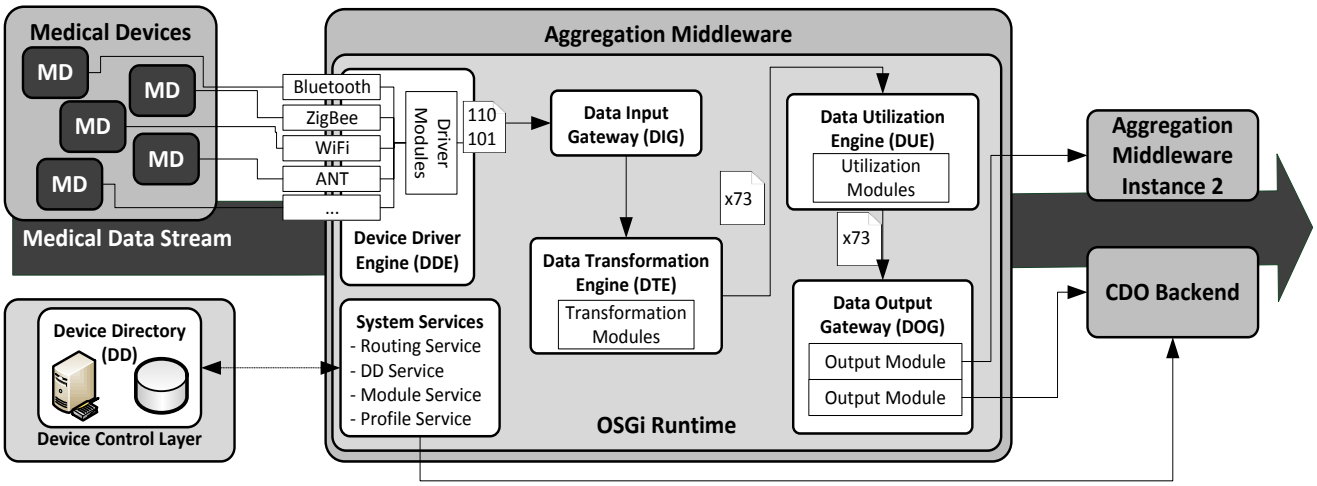

Fig. 2: Overview of the general system architecture. The middleware consists of several mandatory and optional components (i.e. Bundles), that run inside an OSGi engine to allow for dynamic reconfiguration.

Directory and allows to request PM's and configurations required to integrate a DI. Therefore the *Discovery Record* generated by the device integration layer is used. This leads to the requirement, that DI's need to be uniquely identifiable among all connected DD's. The DD's are able to maintain several identifiers for a medical device, which allows to cope with several different approaches of identifying a medical device. The most common approach is to use MAC addresses. However advanced devices, such as x73 compliant ones, already provide means for such globally unique identifiers.

A. Device Integration Layer

The device integration layer (i.e. *Device Driver Engine (DDE)*) is a framework for easy development and runtime deployment of device drivers. A device driver module designed according to the DDE specification can be deployed on each *Aggregator*, regardless of the actual operating system or platform the *Aggregator* runs on. Therefore, the DDE implements a hardware abstraction layer, which allows to provide uniform interfaces to the transport protocols available through the operating system. Moreover, the hardware abstraction layer tries to unify the access to different transport protocols by exposing only one abstract service API to read and write bytes from a transport. After a medical device was discovered, the *Discovery Record* holds a reference to an abstract communication channel, which later can be used by the device driver to communicate with the medical device. The abstract communication channels are based on the channel concept introduced by Java NIO. This allows a device driver for a certain DT to be used with DI's based on different transports. A use case for this feature is given by the x73 standard, which allows to implement several transports based on its application level protocol. The same device driver module could be used for all x73 blood pressure monitors for instance, regardless whether they are based on Bluetooth (i.e. HDP), ZigBee or Ethernet.

Besides the hardware abstraction capabilities, the DDE follows a two step approach to integrate medical devices. The discovery process and the integration process are separated from each other, which corresponds to the two module types defined above. Discovery of a medical device is subject to a lot of different heterogeneous approaches. However, without being able to discover a device, it cannot be integrated and there would be no need for self-adaptivity of all following components. Therefore, discovery modules are the only pluggable components of the middleware that have to be deployed manually (i.e. runtime deployment is still possible, but it has to be triggered manually). Our assumption is, that the discovery process itself is much more lightweight than the integration process (i.e. the implementation of the application level protocol used by the medical device) and in case of some transports (e.g. Bluetooth), the discovery process can be unified (i.e. handled by one single module). Therefore, the separation of discovery and integration allows us to comply with the resource constraints while still being able to support a huge variety of medical devices.

The discovery of a medical device results in the creation on a *Discovery Record*, which is used to trigger the gathering of required device driver modules from the DD. After the device driver module was deployed to the DDE (i.e. installed into the OSGi framework), a session is wrapped around the driver and the *Discovery Record* is passed. The session concept allows us to handle several DI's of the same DT in parallel. Note that the integration can be intercepted by a call to CDO, in order to check whether the medical device is actually required, but this is up to the actual use case. Additionally, the DD locking process described in Section II-A has to succeed, prior to integrating the device. The reference to the abstract communication channel contained in the *Discovery Record*, then allows the driver to establish a connection to the medical device and record data from it. The data is forwarded to an application layer component of the DDE, which simply

forwards the data again to the Data Input Gateway (DIG) of the aggregation layer.

B. Aggregation Layer

1) Input Gateway: After integrating the medical device, the aggregation layer is responsible for handling the data stream in a way defined by the CDO. When the first data container of a newly integrated device has arrived at the DIG, a new session for this DI is created. All following containers can be assigned to the session using a session ID, which is already attached to the container by the DDE. The session ID refers to the ID of the lock, generated by the DD prior to integrating the medical device. This lock ID uniquely identifies the binding between an *Aggregator* and a medical device and therefore can be used as a session ID, since it will not change while the DI is connected to the *Aggregator*.

Since the DI and the DT are already known and cached by the DD service (due to the device integration layer request), the DIG can examine the corresponding configuration and knows about all PM's required to handle the data stream. However, the actual orchestration of data transformation and data utilization modules depends on the requirements of the CDO and can also differ from patient to patient. Thus, the DIG has to request a DI and possibly patient related profile from the CDO using the profile system service (the corresponding patient can be obtained on the basis of the lock ID). The profile defines which data utilization and data output modules have to be executed. The order currently has to be defined within the profile and is not generated automatically to avoid loops within the orchestration. However, each PM defines which data input formats are expected and which output formats are generated. This allows the DIG to automatically identify the set of data transformation modules required to execute the requested module orchestration. If a required transformation module is missing, the regarding execution path of the module orchestration is not executed and a failure is reported.

Afterward, the DIG requests the required PM's from the DD using the module system service, which allows to load the modules asynchronously and not to delay the processing until all required PM's are available. The data container then is forwarded to the system routing service, which creates a routing configuration for the session (i.e. the order of PM's). The general ordering of modules is dynamic, which means that it is not required to constrain the execution path to a fixed sequence, as displayed in Figure 2 due to simplicity reasons. It is allowed that a transformation is followed by a utilization and another transformation again. As shown in Figure 3, it is also allowed to establish several execution paths within a data stream session. Each execution path defined in the profile is terminated with a data output module.

2) Receiver Engines: After having created a routing configuration for the session, the routing system service controls the processing of the data stream using a set of so called *Receiver Engines*, whereas each receiver engine refers to a component that handles PM's of a certain type. The aggregation layer knows three types of receiver engines: *Data Transformation Engine*, *Data Utilization Engine* and, *Data Output Gateway*. Each engine is responsible to manage PM's of the respective type. The main tasks of the *Receiver Engines* are to ensure a proper ordering of data containers corresponding to one stream, to cache data containers in case the next module to be executed is not already installed by the module system service and, to prepare the current configuration of a PM. Because several medical devices have to be handled in parallel, a PM might have to deal with data streams from different DI's that can correspond to different DT's (even streams belonging to different patients are possible). According to Section II-A, DI's, DT's and PM's can have separate configurations. Additionally, the patient profile loaded by the profile system service also can have a configuration attached. Because each of these configurations can have a different priority and keys contained in one configuration can override keys in another one, it is impossible to prepare a configuration used by the PM's once for the session (e.g. by the DIG). Therefore the session related configurations (i.e. DI, DT, Profile) and the PM related configuration have to be merged by the *Receiver Engine* prior to trigger the next PM contained in the routing setup. In order to avoid a duplicate generation of routing information (i.e. at the routing system service and the engines), the routing system service pushes an engine related module execution stack along with the data container, which contains the sequence of all PM's the engine is responsible for and that have to be executed for the current execution path.

The data output gateway takes a special role among the engines. It is responsible for transmitting the data stream to other nodes (e.g. other aggregators or the CDO backend). It is designed as an engine, to allow for separate output destinations even among one data stream. Since the routing sequence is not fixed, the data output modules can intercept the data stream at any point of execution, which for instance allows to forward the stream using different data formats as shown in Figure 3 (e.g. the raw stream for monitoring and logging purposes and a transformed stream for further processing). A concatenation of several *Aggregators* is also possible, because the session ID was initially generated by the device integration layer and the DD and therefore can be maintained among several *Aggregator* instances. In this case the stream is simply transmitted to the DIG of the subsequent *Aggregator*.

C. Data Stream Representation

A medical data stream is modeled by a data container representation. Each PM and each *Receiver Engine* expects an instance of an abstract data container as its input. The data container holds a reference to the session it belongs to, which allows each module participating in an execution path to identify the session and the corresponding DI and DT as well. Additionally, the container provides a sequence number and the data format its payload complies to. If a patient profile includes two utilization modules in an execution path that differ regarding their corresponding output and input formats, the middleware can automatically intercept the execution path with the required transformation module.

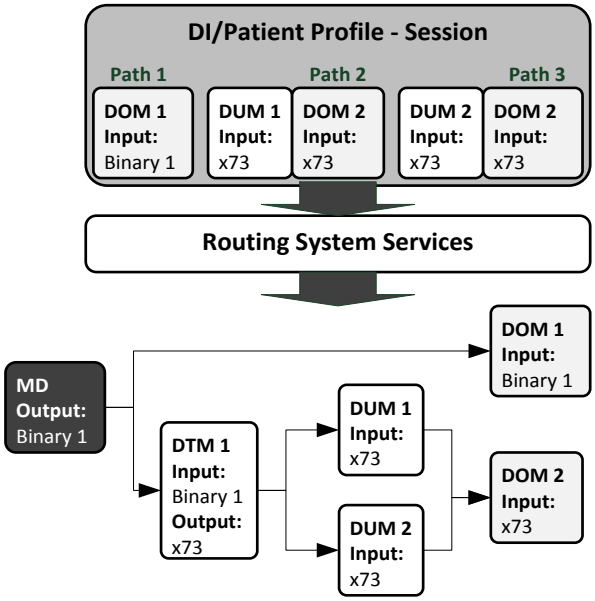

Fig. 3: Routing setup generated from a profile configuration with automatically integrated transformation modules.

The possibility to aggregate data from different data streams (i.e. different medical device sessions) is given implicitly by the middleware, because there is always only one instance of a certain PM deployed. The isolation between different data streams is introduced by the session concept described in the last sections. Based on the session ID and the profile configuration, a PM is able to identify medical data streams that belong to the same patient. However, there is currently no notion available, that allows to explicitly model such PM's that intercept different sessions. This means, currently the PM itself has to take care of synchronizing two data streams it wants to aggregate. Enhancing the middleware with such feature will be target of future work.

D. Prototype

In the scope of the ReahInterAct project [21], we implemented a first Java based prototype of our middleware solution. Based on sensors integrated in training equipment (e.g. shoes) and a virtual reality feedback system, interactive exercise scenarios can be realized. The systems can be seperated into local components (e.g. sensors, video system) used in the exercise room and backend components (e.g. clinical information base). The *Device Directory* is deployed as a part of the backend and uses a websocket communication protocol (based on JSR356 [22] and Jetty [23]) to expose its interfaces. The middleware is deployed on a local PC (Intel Core i5 3320m, 4GB RAM, JDK 1.7) and uses Apache Felix [24] as its OSGi runtime. So far, two movement and pressure sensors, developed by project partners and based on Bluetooth (BT2.1 and BT4.0 respectively) with proprietary data formats (25Hz, 10 bytes payload), were integrated. We developed two corresponding transformation modules to align the proprietary format to a common one based on x73 [8]. Because x73 currently does not support movement and pressure sensors, we extended the private sections of the nomenclature.

We examined the feasibility of our approach by measuring the delay introduced by routing the data stream through our middleware's core components and the transformation module as well as the time required for the first data container to pass through, which includes the inital setup of the session and the routing tables. The time required for the actual download of the platform modules was not considered, because it higly depends on the available bandwidth between local and backend site. It showed that the average delay was less than 40ms, which is appropriate for the given data rate of the sensors and does not affect the virtual reality system's feedback loop. Due to the inital setup of the routing, the delay of the first data container was at most 110ms higher.

IV. Related Work

The device integration problem is common to a broad range of application domains like Ambient Assisted Living (AAL), E-Health or, Smart Homes. Due to the static and monolithic nature of available solutions (in terms of integratable sensors, adaptability, mobility, patient-sensor binding, device sharing), a lot of research is conducted to simplify the integration process. Brito et al. [25] presented a middleware for the integration of heterogeneous medical sensors. Similiar to our approach, new medical devices and services (i.e. data utilization modules) can be integrated. However, the process is limited to the start-up process and the movement of sensors between different CDOs is not covered. King et al. [26] focused on the integration of stationary medical devices in clinical environments. Additionally they presented a test bed to evaluate the coordination of networked medical devices. Asare et al. [27] proposed a medical device dongle to integrate heterogeneous and proprietary devices. Their approach is based on an open-source hard- and software platform allowing to connect to proprietary medical devices over a network using a common protocol (x73). Each dongle implementation has to be specifically designed according to the requirements of the medical device to be integrated. Dagtas et al. [28] presented a health monitoring system using cell phones. Similiar to the *Aggregator* approach presented by us, the cell phone is capable of collecting measurements and analyzing the recorded data. Multiple transport technlogies are supported and preprocessing of recorded data can be carried out using a rule-based approach with XML.

Although most of the related approaches already allow to handle several different medical devices and are able to dynamically deploy application logic (e.g. [29]) close to the data sources, a runtime adaption of the middleware itself, required when continuously broadening the set of available medical devices and sharing them among CDOs, is not supported.

V. Conclusion and Future Work

We presented an approach for a protocol and platform agnostic device integration and data aggregation middleware, that is able to adapt itself to the requirements of the current

environment during runtime. We assume, that a hierarchy of device directories exists, that similar to common directory services, maintain knowledge about the medical devices available. The knowledge exists in shape of configurations and software modules, that can be loaded by the middleware at runtime. Therefore the middleware is able to adapt its behavior regarding device integration and data stream processing capabilities on-demand. The high modularity enables to manage the amount of software modules deployed in parallel efficiently and allows to deploy the middleware on mobile and resource constrained devices, while not limiting the set of medical devices usable. The presented device directory approach additionally introduces the notion of a federated medical device cloud, which allows to share medical devices among CDOs participating in a patient's treatment process.

Future work targets the dynamic orchestration of all utilization modules. Currently only transformation modules can be integrated into an execution path dynamically, which is due to the possibility of creating loops when orchestrating all modules based on their input and output data formats. Security and privacy issues will be further investigated, which includes policy enforcement (e.g. only secure output module that allows to encrypt the data stream are integrated) or code signing features (e.g. only trusted modules signed by the DD or the CDO authority are allowed to be integrated in an execution path).

ACKNOWLEDGMENT

This work was carried out within the scope of the RehaInteract Project sponsored by the Federal Ministry of Economic Affairs and Energy (BMWi, Germany).

REFERENCES

[1] U. Varshney, "Pervasive healthcare and wireless health monitoring," *Mobile Networks and Applications*, vol. 12, no. 2-3, pp. 113–127, 2007.

[2] L. Gatzoulis and I. Iakovidis, "Wearable and portable ehealth systems," *Engineering in Medicine and Biology Magazine, IEEE*, vol. 26, no. 5, pp. 51–56, Sept 2007.

[3] K. Lorincz, D. Malan, T. Fulford-Jones, A. Nawoj, A. Clavel, V. Shnayder, G. Mainland, M. Welsh, and S. Moulton, "Sensor networks for emergency response: challenges and opportunities," *Pervasive Computing, IEEE*, vol. 3, no. 4, pp. 16–23, Oct 2004.

[4] D. Bates and A. Bitton, "The Future Of Health Information Technology In The Patient-Centered Medical Home," *Health Affairs*, vol. 29, no. 4, pp. 614–621, apr 2010. [Online]. Available: http://content.healthaffairs.org/content/29/4/614.full.html

[5] J. Ko, C. Lu, M. Srivastava, J. Stankovic, A. Terzis, and M. Welsh, "Wireless sensor networks for healthcare," *Proceedings of the IEEE*, vol. 98, no. 11, pp. 1947 –1960, nov. 2010.

[6] I. Korhonen, J. Parkka, and M. van Gils, "Health monitoring in the home of the future ," in *Engineering in Medicine and Biology Magazine*, vol. 22, no. 3, May-June 2003, pp. 66–73.

[7] D. Arney, S. Fischmeister, J. M. Goldman, I. Lee, and R. Trausmuth, "Plug-and-play for medical devices: Experiences from a case study," *Biomedical Instrumentation & Technology*, vol. 43, no. 4, pp. 313–317, 2009.

[8] "ISO/IEC/IEEE Health Informatics–Personal Health Device Communication–Part 20601: Application Profile–Optimized Exchange Protocol," *ISO/IEEE 11073-20601:2010(E)*, pp. 1 –208, 1 2010.

[9] Bluetooth SIG, "Health Device Profile (HDP)." [Online]. Available: https://www.bluetooth.org/Technical/Specifications/adopted.htm

[10] M. N. Boulos, S. Wheeler, C. Tavares, and R. Jones, "How smartphones are changing the face of mobile and participatory healthcare: an overview, with example from ecaalyx," *Biomedical engineering online*, vol. 10, no. 1, p. 24, 2011.

[11] O. Alliance, "Osgi service platform release 4 specification," *Online: http://www. osgi. org/Specifications*, 2007.

[12] V. Bicer, G. B. Laleci, A. Dogac, and Y. Kabak, "Artemis message exchange framework: semantic interoperability of exchanged messages in the healthcare domain," *SIGMOD Rec.*, vol. 34, no. 3, pp. 71–76, Sep. 2005. [Online]. Available: http://doi.acm.org/10.1145/1084805.1084819

[13] S. Heiler, "Semantic interoperability," *ACM Comput. Surv.*, vol. 27, no. 2, pp. 271–273, Jun. 1995. [Online]. Available: http://doi.acm.org/10.1145/210376.210392

[14] C. H. Alliance, "http://www.continuaalliance.org/index.html."

[15] J. Yao and S. Warren, "Applying the ISO/IEEE 11073 Standards to Wearable Home Health Monitoring Systems," *Journal of Clinical Monitoring and Computing*, vol. 19, pp. 427–436, 2005, 10.1007/s10877-005-2033-7. [Online]. Available: http://dx.doi.org/10.1007/s10877-005-2033-7

[16] A. Fioravanti, G. Fico, M. Arredondo, D. Salvi, and J. Villalar, "Integration of heterogeneous biomedical sensors into an ISO/IEEE 11073 compliant application," in *Engineering in Medicine and Biology Society (EMBC), 2010 Annual International Conference of the IEEE*. Buenos Aires, Argentina: IEEE Engineering in Medicine and Biology Society, 2010, pp. 1049–1052.

[17] A. A. Sani, F. Polack, and R. Paige, "Generating Formal Model Transformation Specification Using a Template-based Approach," in *Scope of the Symposium*, 2010, p. 3.

[18] D. Ivanov, A. Kliem, and O. Kao, "Transformation middleware for heterogeneous healthcare data in mobile e-health environments," in *Proceedings of the 2013 IEEE Second International Conference on Mobile Services*, ser. MS '13. Washington, DC, USA: IEEE Computer Society, 2013, pp. 39–46. [Online]. Available: http://dx.doi.org/10.1109/MS.2013.15

[19] K. Czarnecki and S. Helsen, "Feature-based Survey of Model Transformation Approaches," *IBM Systems Journal*, vol. 45, no. 3, pp. 621–645, 2006.

[20] A. Kliem and O. Kao, "Cosemed - cooperative and secure medical device cloud," in *e-Health Networking, Applications Services (Healthcom), 2013 IEEE 15th International Conference on*, Oct 2013, pp. 260–264.

[21] Fraunhofer Institute for Open Communication Systems, "RehaInterAct Project," accessed 19-August-2014. [Online]. Available: http://rehainteract.fokus.fraunhofer.de/

[22] Oracle Corporation, "JSR 356: Java API for WebSocket," accessed 19-August-2014. [Online]. Available: https://jcp.org/en/jsr/detail?id=356

[23] Eclipse Foundation, "Jetty Web Server," accessed 19-August-2014. [Online]. Available: http://www.eclipse.org/jetty/

[24] Apache Foundation, "Apache Felix," accessed 19-August-2014. [Online]. Available: http://felix.apache.org/

[25] M. Brito, L. Vale, P. Carvalho, and J. Henriques, "A sensor middleware for integration of heterogeneous medical devices," in *Engineering in Medicine and Biology Society (EMBC), 2010 Annual International Conference of the IEEE*, 2010, pp. 5189–5192.

[26] A. King, S. Procter, D. Andresen, J. Hatcliff, S. Warren, W. Spees, R. Jetley, P. Jones, and S. Weininger, "An open test bed for medical device integration and coordination," in *Software Engineering - Companion Volume, 2009. ICSE-Companion 2009. 31st International Conference on*, 2009, pp. 141–151.

[27] P. Asare, D. Cong, S. G. Vattam, B. Kim, A. King, O. Sokolsky, I. Lee, S. Lin, and M. Mullen-Fortino, "The Medical Device Dongle: an Open-source Standards-based Platform for Interoperable Medical Device Connectivity," in *Proceedings of the 2nd ACM SIGHIT symposium on International health informatics*, 2012, pp. 667–672.

[28] S. Dagtas, Y. Natchetoi, and H. Wu, "An integrated wireless sensing and mobile processing architecture for assisted living and healthcare applications," in *Proceedings of the 1st ACM SIGMOBILE international workshop on Systems and networking support for healthcare and assisted living environments*. ACM, 2007, pp. 70–72.

[29] M. Eisenhauer, P. Rosengren, and P. Antolin, "Hydra: A development platform for integrating wireless devices and sensors into ambient intelligence systems," in *The Internet of Things*, D. Giusto, A. Iera, G. Morabito, and L. Atzori, Eds. Springer New York, 2010, pp. 367–373. [Online]. Available: http://dx.doi.org/10.1007/978-1-4419-1674-7_36