Using Data Distribution Service for IEEE 11073-10207 Medical Device Communication

Merle Baake[1], Josef Ingenerf[2], and Björn Andersen[2](✉)

[1] Cardioscan GmbH, Hamburg, Germany
merle.baake@cardioscan.de

[2] Institute of Medical Informatics, Universität zu Lübeck, Lübeck, Germany
{ingenerf,andersen}@imi.uni-luebeck.de

Abstract. The concept of an Integrated Clinical Environment can be implemented by a fully connected operation room containing devices from different manufacturers. An exchange architecture and protocol for this kind of environment is defined by the IEEE 11073 Service-oriented Device Connectivity family of standards. Therein, a Domain Information and Service Model is bound to the Medical Devices Communication Profile for Web Services, which is the specification for the information exchange technology. It is employed as the communication layer in the software library SDCLib/J that implements an Integrated Clinical Environment. In order to demonstrate that the functionality of SDCLib/J is independent of the underlying transport technology, its communication layer was replaced with an implementation of the Data Distribution Service. Therefore, its publish-subscribe pattern needed to be redesigned and transformed so that it matches the library's request-response principle.

Keywords: IEEE 11073 SDC · SOMDA · DDS · ICE · Publish-subscribe · Request-response

1 Introduction

Interconnecting point-of-care medical devices from various manufacturers has been shown to enable technological and economic benefits [8]. Apart from data exchange, remote control is of great importance when connecting medical devices [12]. By monitoring parameters like heart rate or blood pressure and using remote control, functions depending on their current values can be controlled, e.g. pausing the injection of a drug when the patient's vital signs drop out of a previously defined safe range.

Unfortunately, in most cases, a device can only be connected efficiently to another device if it is from the same vendor [5]. In case devices from mixed manufacturers are introduced, additional interfaces are required. Therefore, users have a high dependency on the availability of devices from the same manufacturer.

G. M. P. O'Hare et al. (Eds.): MobiHealth 2019, LNICST 320, pp. 127–139, 2020.
https://doi.org/10.1007/978-3-030-49289-2_10

The *Integrated Clinical Environment (ICE)* describes a clinical workplace such as an operation room, where devices can communicate even if they are from different manufacturers [9]. However, the concept needs to be implemented with specific software technologies. In the IEEE 11073 *Service-oriented Device Connectivity (SDC)* series, a communication protocol is described that aims to fulfil the requirements of an ICE and can be implemented as a software library [12]. Such a library acts as a communication layer under the application layer and provides components for the data exchange in distributed systems of medical devices.

SDC uses the *Medical Devices Communication Profile for Web Services (MDPWS)* as its underlying transport technology. It is thus implemented as the communication layer in the SDC implementation SDCLib/J. Other ICE implementations, like OpenICE, use the communication protocol *Data Distribution Service (DDS)* and demonstrated it to be generally appropriate in a medical context [1].

As the design of the SDCLib/J reflects the loose coupling of the IEEE 11073 SDC standards, the transport technology is exchangeable. The interconnection of medical devices based on the IEEE 11073 data model is therefore not bound to MDPWS. In this work, SDCLib's communication layer is extended so that devices can communicate over DDS, broadening the variety of devices being able to exchange information through the library. To integrate the new communication protocol, DDS' publish-subscribe pattern needed to be mapped to SDCLib's request-response principle.

2 IEEE 11073 SDC

The IEEE 11073 series of standards aims for the interoperability of medical devices and information systems [3]. Three recent additions to it are known as the SDC sub-series [6]:

20701 Service-Oriented Medical Device Exchange Architecture and Protocol Binding (SDC)
10207 Domain Information and Service Model for Service-Oriented Point-of-Care Medical Device Communication (BICEPS)
20702 Medical Devices Communication Profile for Web Services (MDPWS)

2.1 20701: Architecture and Protocol

SDC specifies the technical interface for interconnected medical devices at a clinical workplace, e.g. an operation room, as described in the ICE concept [11]. The specification adheres to the *Service Oriented Medical Device Architecture* (SOMDA) pattern, which assumes that medical devices represent their capabilities in the network as *services*. The focus lies on reducing the complexity and costs that come with the integration of devices into distributed enterprise systems. Dealing with medical data, the exchange is highly safety-critical and underlies strict requirements.

2.2 10207: Information and Service Model

Furthermore, IEEE 11073 defines the *Basic Integrated Clinical Environment Protocol Specification* (BICEPS). In the *Medical Device Information Base* (MDIB), the core of the data model, metrics describe the capabilities of a medical device, like vital parameters, settings, states, and contextual information [3].

The following service operations are defined in BICEPS to exchange information in a medical context [3,16]:

GET	reading access (request-response pattern),
SET	writing access to change values → remote control,
EVENT REPORT	reading access (publish-subscribe pattern),
ACTIVATE	execute a predefined job on a remote device → remote control.

To access the metrics, *GET* and *SET* methods allow for reading and possibly writing access as per the request-response pattern: Values can be actively fetched or modified by a client with read/write access. *EVENT REPORTS* are based on reading access as well, but employ the publish-subscribe paradigm – notifications can be sent periodically or when a change of value occurs. The use of an *ACTIVATE* operation triggers an action and thereby allows for external control of a medical device.

2.3 20702: Communication Technology

The communication protocol MDPWS is needed to technically realise the SOMDA [8]. It is implemented as a set of web services, which support interoperability across a network using the *Web Service Definition Language (WSDL)* and SOAP messages. MDPWS fulfils the special requirements of communication between medical point-of-care devices, enabling i.a. remote control and data streaming.

2.4 SDCLib: Communication Library

The Java library SDCLib/J implements IEEE 11073 SDC, thereby putting the ICE concept into effect [18]. Figure 1 shows the setup of devices implementing the library. The application itself serves as the entry and exit point for medical device data into and out of the network. The library establishes the consumer and the provider and processes the exchanged information used by the application. The SDC protocol specification works as a binding between the data model BICEPS and the transport layer MDPWS, which passes on the data and handles the technical details on the information exchange.

Due to the separation of service model and implementation, it does not matter which communication protocol is used to exchange the data as long as the data structure of the BICEPS layer is preserved in the lower layers.

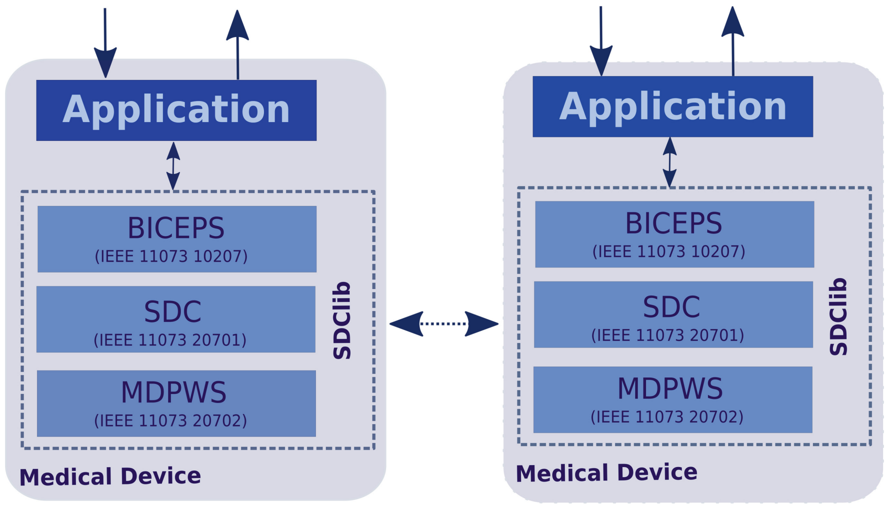

Fig. 1. Based on MDPWS and BICEPS, the library enables the exchange of information between multiple devices. The application using the library serves as the entry and exit point of the data into and out of the network.

3 Data Distribution Service

3.1 A Data-Centric Protocol

The main focus of *Data Distribution Service (DDS)* lies on the data-centrality and timely availability of information [15]. In a global data space, a unit of information – a *Topic* – is transmitted between publishers and subscribers [7]. DDS is designed following the publish-subscribe principle. Instead of the client specifically asking for new information, as in the request-response pattern, clients subscribe to specific types of information and receive the respective data as soon as the server sends it. With a single subscription, nodes can subscribe to many similar data streams to receive data.

Figure 2 shows the interaction between the basic components needed for the communication process. The scenario assumes that all entities are part of the same data space (domain). Each of the two Topics is associated with a DataReader/ DataWriter pair, which is used by the application(s) to exchange information on the specific Topic. The DataWriter for Topic A writes samples, which can only be read by the respective DataReader. The DataReader associated with Topic B has no access to the samples of Topic A and vice versa.

User-specific types for the Topics can be defined comprehensively. Rather than sending general messages, the communication layer understands the types syntactically, creating a type-safe environment.

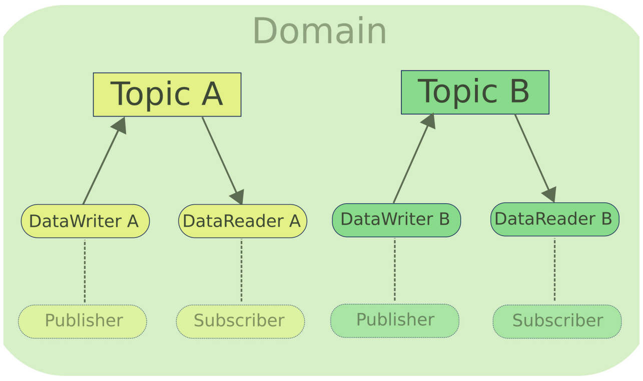

Fig. 2. DataReaders and DataWriters can only communicate through their associated Topic. Although they exist in the same domain and know of each other's existence, DataReader B will not receive any samples written by DataWriter A.

3.2 Exchanging Data Within a Domain

Communication takes place in the *Domain*, which represents a global data space and enables multiple applications to communicate within this data space [7]. It is important to note that the data is not physically located on a central domain storage; it is rather kept decentrally in the caches of its DataWriters and DataReaders.

Creating a DomainParticipant with a specific 'domainId' is the ticket for the application to enter the domain. Based on the DomainParticipant, all other entities which enable communication are created by the software implementation of DDS. Multiple applications can exchange data when initiating their DomainParticipants with the same 'domainId'.

3.3 Quality of Service Parameters

The setting of Quality of Service (QoS) parameters determines how the components of the application behave before, during, and after sending and receiving data [4]. With these parameters, memory preallocation for the samples, the size of the caches, and details of the transport process can be defined comprehensively. The main goal is to maximise the likelihood of readers and writers to match while controlling the entities' behaviour. By default, the QoS parameters are set to fit general use cases.

As visualised in Fig. 3, a DataReader and a DataWriter are in the same Domain and bound to the same Topic. Because their QoS parameters are compliant, they are able to communicate (A). In case of a discrepancy between their defined QoS parameters, the two entities are declared as incompatible by DDS and can not exchange any information (B).

If the QoS parameters of a DataReader request e.g. a higher rate of receiving data than the DataWriter offers through its own QoS parameters, the communication layer does not enable communication between the two.

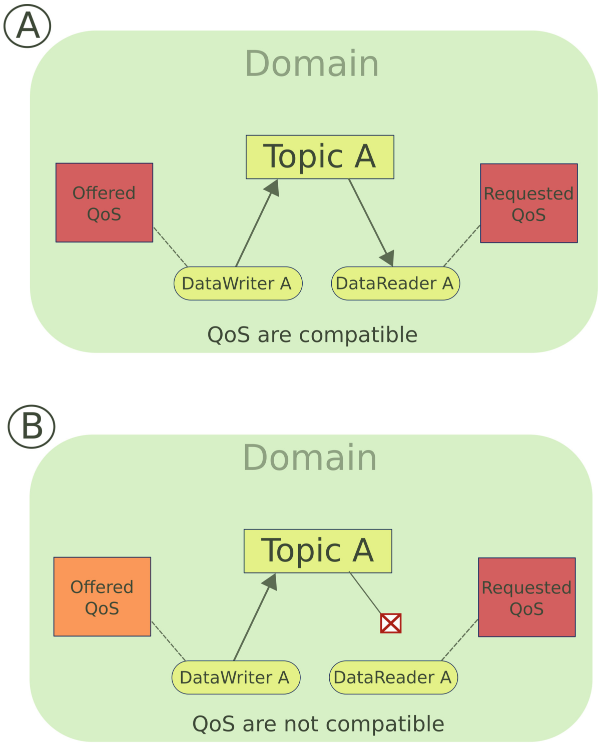

Fig. 3. DataWriter and DataReader must be in the same Domain, bound to the same Topic and have agreeing QoS parameters. If the DataWriter can not fulfil the DataReader's requested QoS parameters, or vice versa, the pair is declared as incompatible.

4 Concept

To recreate the functionality of SDCLib/J with DDS, the communication layer is replaced by the new protocol. For that, the request-response pattern is recreated with the use of DDS' Topics.

The initial implementation of SDCLib/J based on MDPWS is designed so that all information is mapped to messages containing XML before being sent trough the network. After receiving the data, it is transformed to its original form so it can be processed by the upper layers.

With DDS, information types are handled individually. For every *GET* and *SET* operation, as well as the *EVENT REPORTS*, individual Topics need to be created containing the necessary data. Along with the created Topics, the consumer side needs to implement DataWriters for the requests and DataReaders for the responses and the reports. Respectively, the provider's side implements DataReaders for the requests as well as DataWriters for the responses and the reports. Figure 4 exemplifies this based on the entities that are needed to exchange information on the MDIB.

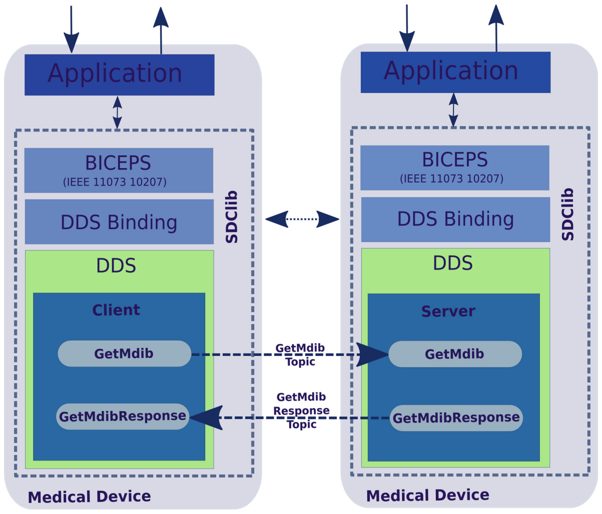

Fig. 4. In the new setup, the library exchanges data through DDS. The consumer requests the MDIB of the provider by sending a sample of a GetMdib Topic, which is received by the provider and answered with a sample of the GetMdibResponse Topic.

ICE emphasises the need for reliable one-to-one communication. Three steps guarantee that the client finds the intended server and exchanges the information needed:

1. When a device enters the domain in order to communicate with other devices, a new DomainParticipant is initiated for the role of the client and another one for the server. All participants are based on the same domainId so that communication takes place in the same data space. Automatic discovery is enabled so that the client is notified when new DomainParticipants enter the domain and can extract information, e.g. the server's endpoint reference. Thereby, the client can directly address the server to request information or to initiate remote control.
2. The endpoint reference of the server is added to the request sample. On the server side, a filter is used to only read requests that hold the server's endpoint reference.
3. The DataReader for the response waits for samples of the response Topic. Only the sample containing the identifier of the previously sent request is processed.

5 Implementation

In SDCLib/J, the concept of the information exchange is basically the same for every use case: the consumer sends a *GET* or *SET* request, which is received by the provider. The provider creates the response and sends it back. Upon receiving a subscribe request, *EVENT REPORTS* are triggered that are sent directly to the subscriber.

SDCLib/J is designed so that the transport layer is connected to the upper layers via a binding interface. Through the extension described in this work, the library user can now switch between MDPWS and DDS to use the respective implementation of the interface.

To enable the communication process, for each consumer and each provider one DomainParticipant is created and initiated with the same 'domainId'. The Topics, DataReaders, and DataWriters are initiated based on the DomainParticipants on both sides.

The DataReaders that read requests on the server side continuously wait for data without interrupting the application. Implemented Listeners allow them to wait for requests in the background, allowing all other processes to run while being ready to take and process new requests at all times.

The information that is requested by the consumer is implicitly asked for and used by the application. Therefore, the DataReaders for the responses start to wait for data right after the request is sent, causing the current thread to stop temporarily until the requested information is available to them.

Furthermore, Listeners are also implemented on the DataReader's side to listen for *EVENT REPORTS*.

6 Evaluation

The implementation process was closely tied to tests, which were initially designed to test the basic functionalities of the library based on MDPWS. Now, the binding interface implemented with DDS is passed as the transportation layer. It was thereby guaranteed that the new transportation layer enables the same functionalities as MDPWS.

6.1 Qualitative Distributed Analysis

As SDCLib's purpose is to enable communication in distributed systems, the implementation was tested on virtual machines representing medical devices. A consumer and a provider with basic information are initialised on one virtual machine each. Communication is possible due to the machines entering the same Domain. Therefore, the consumer's DataWriter for the *request* Topics can exchange data with the respective DataReader on the provider side.

Several tests assure that information is exchanged correctly. This includes basic information being requested and sent, parameters being set remotely, and alerts being triggered correctly.

Indeed, the machines were able to exchange the information accordingly. The *GET* and *SET* methods could be executed as intended and *EVENT REPORTS* were triggered correctly. To enable two or more devices to exchange information, the setting of the QoS parameters was crucial. With the default parameters, information could only be exchanged between instances running on a single machine. Therefore, specific adjustments were necessary as described in Sect. 6.2 to enable communication between multiple devices.

6.2 QoS Parameters for Distributed Systems

Running the application on multiple machines for the purpose of evaluation required adjusting the QoS parameters as the defaults were not feasible. To assure that all requests and the related replies are received, reliable communication was enabled. If a DataReader does not receive a sample, it is repaired and resent, making sure it will receive the sample eventually. To further increase safety in the process, acknowledgements are sent automatically after a DataReader has read a sample and returned it to the cache. The number of instances of each sample is limited so that very old instances that are most likely obsolete are deleted and do not occupy memory.

It may occur that a client requests an MDIB of a device that is not on the network (yet). Still, it should be possible for the device to receive the request. The durability parameter is set so that late joiners receive all sent requests.

With limited memory, it proved to be better to reduce the number of samples that are stored after they were sent or received. Only a specific number of samples can be resent if they were not received by the intended DataReader, either because it did not join the network yet or because the sample got lost.

To make sure all participants/applications can be found on the network, discovery through *UDPv4* is enabled. That assures the discovery of applications on different machines.

7 Results

By replacing SDCLib/J's built-in communication protocol MDPWS with the alternative DDS, it has been proven that communication between medical devices based on this library is not limited to one transport technology. This meets the requirement for data exchange between medical devices having to be independent of the underlying communication protocol as described in the ICE concept.

8 Discussion

8.1 Challenges

Whereas MDPWS is able to implement both the request-response pattern (*GET* and *SET* operations) and the publish-subscribe principle (*EVENT REPORTS*), DDS is specifically designed to support only the latter. Therefore, it was challenging to map the other services to DDS and to support the request-response pattern as DDS does not intend the client to start the communication process by addressing the server directly. Rather, the provider starts the communication process, sending out data to all participants. This had to be considered when integrating the new protocol.

Therefore, our implementation does not think of the consumer/client as one DataReader, which receives information, and the provider/server as a DataWriter, which publishes data. Both sides uphold multiple DataWriters and DataReaders. Different from DDS' initial approach, where a DataWriter starts the information exchange by broadcasting data whenever it is available (except for *EVENT REPORTS*), the DataWriters are triggered by certain events, usually by receiving a request.

8.2 Advantages of Using DDS

The main focus of this work lies on the substitution of the communication layer, enabling information exchange in a distributed system with a different protocol than MDPWS while keeping the functionality and the data model of SDCLib/J.

Similiar to Mastouri and Hasnaoui in [13], it shall be further investigated to which extent the number of created entities (publisher, subscribers, etc.) influences the performance of the implementation. The suitability of the library shall also be evaluated and compared to other implementations, as done by Kasparick et al. in [10]. To decide whether MDPWS or DDS are more appropriate to exchange data with SDCLib/J, the performance and reliability of both communication protocols need to be examined and compared. Implementations of further communication protocols are currently under evaluation and are expected to be available for comparison in the near future as well.

As evaluated by Serrano-Torres et al. in [17], variances in the performance of different implementations of DDS occur as well and need to be considered.

8.3 Security

Although it is not within the scope of this work, it should be noted that there are data security issues when using this implementation of DDS. The basic concept of DDS, the publish-subscribe principle, is designed in a way that every DataReader of a certain Topic receives all samples from every respective DataWriter.

In terms of medical data, patient information may only be sent to certain participants. The way DDS is integrated into SDCLib/J at this point, all samples are sent to every participant. The discovery process only guarantees that the client uses the server's device reference when writing requests. The listeners on the server side filter all samples that are intended for the respective device. Still, the request can technically be read by any DataWriter (of the specific Topic) of all DomainParticipants in the Domain. Furthermore, new DataWriters may be created bound to any Topic, causing false alarms or unwanted operations.

Adjusting the QoS parameters to send authentication details as the participant's metadata or creating a stricter matching process is not sufficient in a real-world medical context. Depending on which DDS implementation is used, plugins can be added to increase the security level. These include *RTI Connext DDS Secure*, based on *RTI Connext DDS*, which offers comprehensive mechanisms to improve security in critical environments, such as autonomous vehicles, medical and defence industries [14].

8.4 QoS Parameters

A big advantage of DDS lies in the QoS parameters [2]. They allow the developer to determine how much memory is preallocated, making the most of the available resources, and how exactly data is transferred. In a medical context, the parameters that control reliability are especially useful, making sure no important data is lost. It is guaranteed that all data is received as intended. Lost or broken samples are being repaired and resent. Running the application on multiple devices with limited memory, adjusting the resource limits is inevitable. Developers can not only specify the structure of the types by creating Topics, but the QoS parameters also allow for them to determine how the data moves around in the distributed system [4].

When using SDCLib/J with DDS in a practical environment, optimal QoS parameters should be determined beforehand. They must be sufficiently restrictive, especially in a distributed system of medical devices. With the knowledge of available memory capacity, the QoS parameters for memory allocation can be adjusted to process sufficiently large amounts of data.

9 Conclusion

The variety of medical devices from different manufacturers in an operation room demands libraries that allow communication between different medical devices independent of the underlying communication protocol [1]. This principle is specified as part of the ICE concept, which describes a fully connected clinical environment. Within the scope of this work, the communication protocol of the library SDCLib/J is exchanged, proving independence in the choice of communication protocol. It has been shown that the library SDCLib/J is not limited to exchanging data over MDPWS. Replacing MDPWS with DDS as its communication layer shows that the upper layers are agnostic to which communication protocol is used below.

To integrate DDS into the library, the publish-subscribe pattern of the protocol needed to be mapped to the request-response pattern of SDCLib/J. Though being profoundly different, it was possible to recreate this communication pattern with DDS to integrate it into SDCLib/J as its transportation layer. Thereby, the functionalities of the library and the data model remained unaffected.

The data-centric principle of DDS provides certain advantages for the library. With DDS, user-specific data types can be modelled comprehensively, allowing for a type-safe communication process. Furthermore, the QoS parameters are of great advantage, especially for enabling and optimising communication in distributed systems. This is especially important when using DDS in a real-world medical scenario, which imposes stringent safety requirements.

In the future, other communication protocols could be integrated as well, broadening the variety of devices being able to communicate using SDCLib's comprehensive data model.

References

1. Kavya, K.A., Annapurna, V.K.: Integration of medical devices into MIOT using OPENICE. Int. J. Eng. Sci. Comput. **6**, 7323–7324 (2016)
2. Basem, A.M., Ali, H.: Data Distribution Service (DDS) based implementation of Smart grid devices using ANSI C12. 19 standard. Procedia Comput. Sci. **110**, 394–401 (2017). https://doi.org/10.1016/j.procs.2017.06.082
3. Andersen, B., et al.: Interoperabilität von Geräten und Systemen in OP und Klinik. Technical report, Frankfurt, Germany (2015)
4. Arney, D., Plourde, J., Goldman, J.M.: OpenICE medical device interoperability platform overview and requirement analysis. Biomed. Eng. Biomed. Tech. **63**(1), 39–47 (2018). https://doi.org/10.1515/bmt-2017-0040
5. Beger, F., Janß, A., Kasparick, M., Besting, A.: Der Operationssaal OP 40 - Für mehr Sicherheit und Effizienz durch offene Vernetzung in Operationssälen und Kliniken der Zukunft. meditronic-journal - Fachzeitschrift für Medizin-Technik, **4**, 20–24 (2017)
6. Besting, A., Stegemann, D., Bürger, S., Kasparick, M., Strathen, B., Portheine, F.: Concepts for developing interoperable software frameworks implementing the new IEEE 11073 SDC standard family, vol. 1, pp. 258–263. CAOS, 13 June 2017
7. Corsaro, A., Schmidt, D.C.: The data distribution service-The communication middleware fabric for scalable and extensible systems-of-systems. Syst. Syst. **2**, 19 (2012). https://doi.org/10.5772/30322
8. Gregorczyk, D., Fischer, S., Busshaus, T., Schlichting, S., Pöhlsen, S.: An approach to integrate distributed systems of medical devices in high acuity environments. In: 5th Workshop on Medical Cyber-Physical Systems 2014. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik (2014)
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: 2012 IEEE/ACM Third International Conference on Cyber-Physical Systems, 17 April 2012, pp. 3–12. IEEE (2012). https://doi.org/10.1109/ICCPS.2012.9
10. Kasparick, M., Beichler, B., Konieczek, B., Besting, A., Rethfeldt, M., Golatowski, F., Timmermann, D.: Measuring latencies of IEEE 11073 compliant service-oriented medical device stacks. In: IECON 2017-43rd Annual Conference of the IEEE Industrial Electronics Society, October 2017, pp. 8640–8647. IEEE (2017). https://doi.org/10.1109/IECON.2017.8217518
11. Kasparick, M., Schlichting, S., Golatowski, F., Timmermann, D.: Medical DPWS: new IEEE 11073 standard for safe and interoperable medical device communication. In: 2015 IEEE Conference on Standards for Communications and Networking (CSCN), 28 October 2015, pp. 212–217. IEEE (2015). https://doi.org/10.1109/CSCN.2015.7390446

12. Kasparick, M., Schmitz, M., Golatowski, F., Timmermann, D.: Dynamic remote control through service orchestration of point-of-care and surgical devices based on IEEE 11073 SDC. In: 2016 IEEE Healthcare Innovation Point-Of-Care Technologies Conference (HI-POCT), 9 November 2016, pp. 121–125. IEEE (2016). https://doi.org/10.1109/IC.2016.7797712
13. Mastouri, M.A., Hasnaoui, S.: Performance of a publish/subscribe middleware for the real-time distributed control systems. Comput. Sci. Netw. Secur. **7**(1), 313–319 (2007)
14. Real-Time Innovations, I.: RTI Connext DDS Secure. https://www.rti.com/products/secure (2018). Accessed 09 July 2018
15. Rowstron, A., Druschel, P.: Pastry: scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In: Guerraoui, R. (ed.) Middleware 2001. LNCS, vol. 2218, pp. 329–350. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-45518-3_18
16. Schlichting, S., Pöhlsen, S.: An Architecture for distributed systems of medical devices in high acuity environments - a technical whitepaper (2014)
17. Serrano-Torres, R., García-Valls, M., Basanta-Val, P.: Performance evaluation of virtualized DDS Middleware. In: Simposio de tiempo real, Madrid, pp. 18–19 (2014)
18. SurgiTAIX: SDCLib Bitbucket Repository. https://bitbucket.org/surgitaix/sdclib. Accessed 09 July 2018