Semantic Interoperability in Body Area Sensor Networks and Applications

Vinh Bui, Paul Brandt, Hang Liu, Twan Basten, Johan Lukkien
Eindhoven University of Technology
Eindhoven, The Netherlands
{t.v.bui,p.brandt,h.liu.1,a.a.basten,j.j.lukkien}@tue.nl

ABSTRACT

Crucial to the success of Body Area Sensor Networks is the flexibility with which stakeholders can share, extend and adapt the system with respect to sensors, data and functionality. The first step is to develop an interoperable platform with explicit interfaces, which takes care of common management tasks. Beyond that, interoperability is defined by semantics. This paper presents the analysis, design, implementation and evaluation of a semantic layer within an existing BASN platform for the purpose of improving the semantic interoperability among sensor networks and applications. We adopt an ontology-based approach but rather than having a single overall ontology, we find that using clear semantic domains and mappings between them improves composability and reduces interoperability problems. We discuss the design choices and a reference implementation on an Android phone and actual sensor devices. We show by a qualitative evaluation that this semantic interoperability indeed provides significant improvements in flexibility.

1. INTRODUCTION

Body Area Sensor Networks (BASNs) have been developed for a variety of applications [4, 12], such as health and patient monitoring, medical care, fitness and entertainment. Application-wise BASNs are maturing with an array of applications and innovations. However, mass deployment is hindered by a number of factors. First, most BASNs are special purpose designs with fixed applications, which implies that BASNs and their applications are tightly coupled in the development phase. Consequently, application developers need to be aware of these specifics, which leads to application awareness of details of data representation, sensor types and the likes. Second, the meaning of the data collected by the sensors and any prior knowledge essential for interpretation is not recorded, which renders the data useless for future processing [13]. Prior knowledge may refer, for example, to driver details of the applied sensor, to details about how to convert voltage into a blood pressure value, or to assumptions about the influence of the operating environment on sensor accuracy. Third, many applications are currently closed, vendor-specific and specialized services. The applications cannot access each other's information and in many cases, the data acquired is not even accessible to the users. For example, data from the movement sensor of the Philips DirectLife service that monitors people's physical activity levels are currently not available to other services. Another example is that a BASN may need to interoperate with other BASNs, back-end services and even with electronic health record systems.

In [3], we proposed a generic BASN platform named VITRUVIUS (see Figure 1), which integrates multiple sensors and handles different sensor configurations, allows BASN applications to be installed dynamically and run concurrently. A benefit of a platform like VITRUVIUS is that it provides capabilities for reuse and evolution of existing sensor networks and applications, and for extensions by adding new sensors and applications. These capabilities bring benefits such as lower costs relating to application development and maintenance, a higher degree of comfort since the number of mounted devices remains limited (by reusing existing components and sensors), and a higher dependability because of shared platform evolution and the ability of sharing data and resources. For example, in the healthcare domain, an application that detects atrial fibrillation and an application that detects epileptic seizures can use the same sensor for collecting ECG data.

Interoperability among components of distributed systems refers to the ability to exchange services and data with one another [9]. While interoperability is concerned with syntax and parameters of interfaces, semantic interoperability addresses the meaning of operations and exchanged data. It reduces dependencies, which is important for third parties to make system components that can be integrated into the platform easily. Semantic interoperability is, therefore, a necessary condition for flexibility and evolution of BASNs.

We use the VITRUVIUS platform to analyse how semantic interoperability fits in the architecture and how to work effectively with semantics. Semantic interoperability implies that data is given a context for interpretation, which is either known by the receiver or part of the data. As it turns out such context has typically a local relevance. The architecture should support this locality to avoid burdening the entire system with irrelevant details. In this paper, we

This work was supported by the AgentschapNL, through the VITRUVIUS project, part of the IOP GenCom Programme and by the EC, through the RECAP project, part of the Interreg IVB NWE Programme.

present the analysis, design, reference implementation and qualitative evaluation of a semantic layer added to VITRUVIUS with the purpose to improve the semantic interoperability among sensor networks and applications. We adopt the ontology-based approach of [2], in which the developed ontology on context-aware examinations has the potential of not only supporting correct interpretation of sensor data but also ensuring its appropriate use in accordance with the purpose of a given application. The ontology can be regarded as a vehicle that unifies the data originating from different system components into a universal understanding, thereby achieving interoperability at the semantic level. We focus on the analysis and the resolution of problems regarding the architectural design and implementation of the application ontology, data storage and data sharing. Furthermore, we evaluate the BASN platform through aspects of semantic expressiveness, extensibility and data sharing. The platform is implemented on an Android phone connected to multiple sensor devices (Shimmer sensors and a DTI-2 wristband).

The paper is organized as follows. Section 2 summarizes the existing VITRUVIUS platform and presents the ontology-based architecture. Section 3 gives our reference design and implementation of the platform, application ontology and data sharing. The evaluation is given in Section 4. Section 5 concludes the paper.

2. SYSTEM ARCHITECTURE

2.1 Existing system

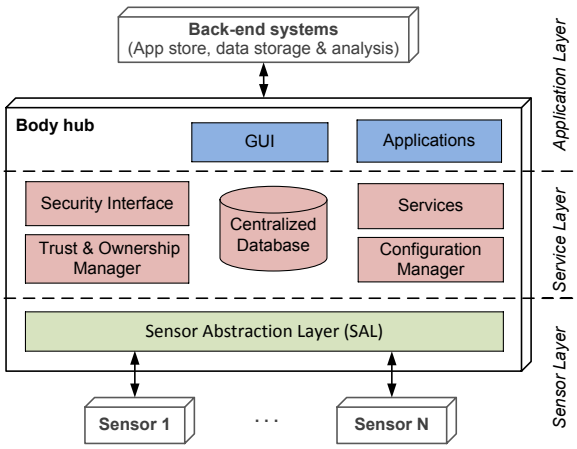

Figure 1: System architecture of the VITRUVIUS Body Area Sensor Network.

Figure 1 shows the architecture of the VITRUVIUS system that was proposed in [3]. The system consists of body sensors together with a more powerful device (body hub) that is equipped with VITRUVIUS software platform. This hub is capable of storing data and running services and applications; it controls the BASN and acts as a single access point. In our reference design, we use a smart phone. We regard VITRUVIUS as a platform on which (concurrent) BASN applications developed by third parties are dynamically installed and configured and to which sensors developed by third parties can be connected. The platform is self-contained and can connect to back-end systems for the purpose of app installation, data storage and analysis, monitoring and control in back-end applications. A modular and layered architecture of the platform is proposed that provides a loose coupling between the applications and the detailed characteristics (e.g. the communication protocol, sensor value representation) of the sensors. Loose coupling can be achieved by separating the architecture into layers, each of them compounding certain concerns. We have applied three functional layers: data and sensor concerns, service concerns and application concerns, as follows.

The *Sensor abstraction layer (SAL)* is an interface between the sensors and the other modules in upper layers of the body hub. This layer allows the system to easily adapt and handle new sensors and their configurations. The layer contains sensor drivers that drive and configure a sensor (e.g. enable/disable sensors or set the sampling rate) to obtain data according to a request from the service or the application.

The *Service layer* contains the downloaded services and the basic services of the platform. A central database is used for the purposes of data storage and sharing among system components. A service is a program that operates only on the data space defined by the user of which the local BASN storage (e.g. a database) is part of. A service typically operates in the background without a user interface. For example, a service derives heart rates from raw ECG signals, or it controls a sensor to obtain data.

The *Application layer* contains the downloaded applications and the GUIs. An application combines the results of services to achieve its functionality. Several applications can share the same service. An application interfaces with the outside world, at least through a user interface but possibly also by communication with other parties (e.g. back-ends) than the body hub. For example, an application that facilitates the user to monitor his daily health conditions and a fitness application share the data generated by the heart-rate detection service. Both may communicate statistics to an external party.

Although this results in a loose coupling in terms of functionality and system components, from a semantic point of view such an architecture still enforces a strict coupling between data schema, syntax and meaning, i.e., one big semantic silo. It is our objective to provide loose coupling also at the semantics level, facilitating true open sensor networks and apps that are truly interoperable. To that end we introduce the following ontology-based architecture.

2.2 Ontology-based architecture

In order to provide loose coupling at the semantic level, first a separation needs to be achieved between semantic and functional concerns. Ontologies are widely recognized as a means to address semantic concerns. Stated in [8], an ontology is a formal, explicit specification of a shared conceptualization. By applying ontologies, i) concepts will become semantically grounded in the referred real-world entities, ii) by reasoning over these concepts, new facts can be inferred or validity of statements can be checked, and iii) by relating the data schemata to an ontology, stored data are enriched with the semantic conventions that apply in the domain of application.

When integrating ontologies in the existing architecture of Figure 1, a foundation is laid for separating semantic from functional concerns. A prerequisite, however, is to

fit those ontologies without disrupting the existing architecture. We therefore engage in an approach that addresses semantic concerns in a semantic architecture that is orthogonal to the existing (functional) architecture that already shows loose coupling between (functional) components. We further observe that not all semantic concepts that are available in the ontologies are relevant for all functional layers and, in addition, it pays in terms of efficiency to lump semantic closely related concerns within a semantic layer together. This leads to localized semantics that is translated or mapped at boundaries. We thus adopt an ontology-based approach, shown in Figure 2, in which the semantic architecture, depicted as 4 horizontal layers, is orthogonal to the functional architecture that is depicted as 3 vertical columns. We can now focus on achieving semantic loose coupling without having to fear to destroy functional loose coupling, and do so by introducing two architectural principles that are fundamental to loose coupling, separation of concerns and transparency, and apply these at that semantic architecture.

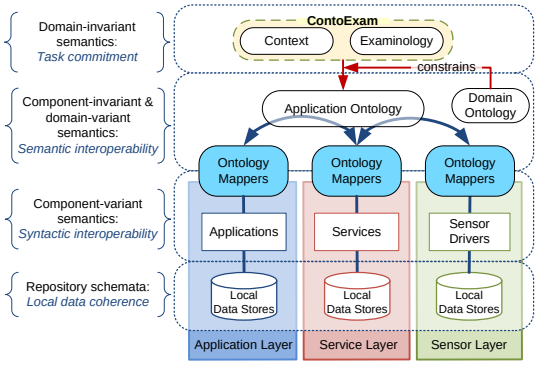

Figure 2: Overview of the ontology-based architecture of the body hub platform.

2.2.1 *Semantic separation of concerns*

In software engineering, *separation of concerns* (SoC) is a design principle that refers to decomposing a software system into parts with as few dependencies as possible [11]. In [2], it is extended and specialized into two separate aspects, the functional SoC and the semantic SoC. As shown in Figure 2, the functional SoC is represented by three vertical layers and the semantic SoC is represented by four horizontal layers. The functional SoC takes care of the functional design of the system, while the semantic SoC simplifies achieving semantic interoperability since it allows separating the semantics into layers with different levels of abstraction. The semantic interoperability then can be achieved at a higher abstraction layer, where less heterogeneous concepts exist than at lower abstraction layers.

The *Repository schemata* layer carries ample semantics, explicitly that is. Computers store data, just data, and there exists no way around that. Hence, semantics at this level are implicitly captured in data and meta-data, and to operate appropriately on that data requires prior knowledge. The inherent semantics is fully settled in their design and development phase. However, we observe that each component in any functional layer will have its own distinct semantics about its own particular data and data format, possibly even about their administration and access, which is relevant

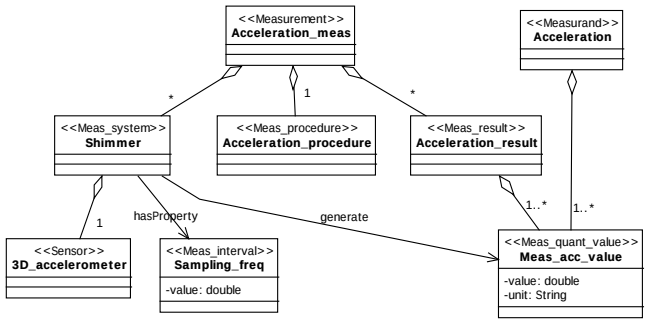

Figure 3: Example of the domain-specific ontology that defines the measurement of acceleration.

for that components operation and purpose, and therefore "owns" its own local data store.

The *Component-variant semantics* layer represents the semantics that enables the management, monitoring and control of operations on the data. This semantic layer is an abstraction from the previous layer, so that a uniform and internal consensus about its local data is provided to the component. In fact, this semantic layer represents the current pragmatics on how services and applications go about semantics: The software engineer has prior knowledge about the meaning of the schemata of the underlying data, and has hard-coded its appropriate use in the software components. For those concepts that are required by other components, their semantics are made explicit by formulating their representation in a local ontology.

The *Component-invariant & domain-variant semantic* layer represents the domain-specific knowledge within the intended universe of discourse (UoD). In our case study, this ontology is set in the Remote Patient Monitoring (RPM) domain. We discern here the domain ontology that is responsible for providing semantics about the domain of application, e.g., the SNOMED ontology, and the application ontology that represents the constrained specification from the upper layer and provides for the shared UoD. Both domain and application ontologies coincide with their definitions from [7].

The *Domain-invariant semantic* layer is responsible for providing the abstract language that is capable to formulate the shared UoD: A stable, semantic foundation about the tasks and activities that apply in the considered world: a task ontology [7]. Since we address BASN applications, we developed an ontology, *ContoExam* [1], a domain-generic task ontology about examinations and their context that is founded in examinology, representing an abstracted UoD.

To illustrate the above architecture, we present an example of how *ContoExam* can be instantiated into a UoD about acceleration. The survey in [5] shows that ontologies for sensor networks can be classified into three perspectives: the sensor perspective, e.g., sensor properties such as the sampling rate; the system perspective, e.g., components of a sensor system and their organization; and the measurement perspective, e.g., a measurement and its result.

The example about an acceleration measurement is shown in Figure 3, represented as an UML class diagram. As an instantiation of the *Measurement*, the *Acceleration_meas* is a structured activity that is prescribed in the *Acceleration_procedure* and the activity of *Acceleration_meas* essentially is to obtain, through a *Shimmer* sensor system [14], an

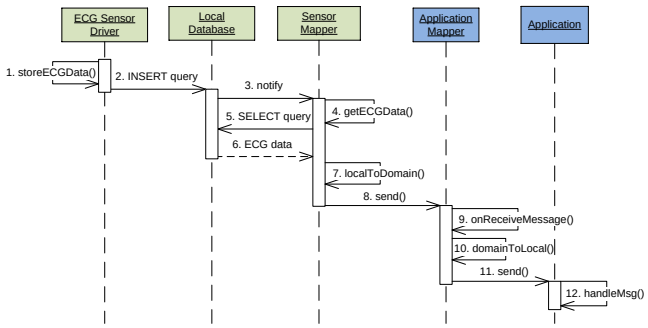

Figure 4: Example of data sharing between the sensor driver and the application via ontology mappers.

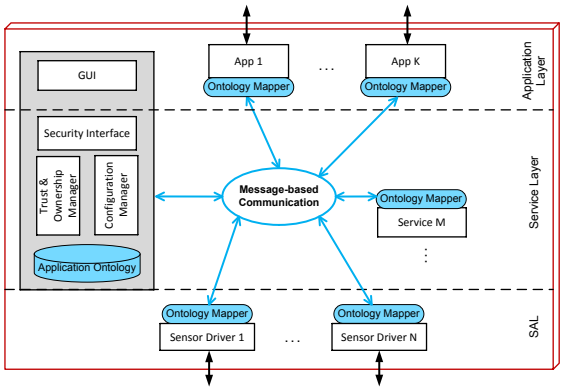

Figure 5: The architecture with decentralized data stores (local data stores are not shown).

Acceleration_result, including its accuracy, *Meas_acc_value*. The *Shimmer* sensor system has a *Sampling_freq* property and consists of a *3D_accelerometer* sensor.

2.2.2 Semantic transparency

With semantic SoC, different semantic layers are identified, which clarifies that the semantic interoperability is designed to take place at the component-invariant semantic layer. With *semantic transparency*, interoperability with minimized mutual dependence is achieved. Semantic transparency is defined in [2] as the characteristic that the external world is unaware of how each component establishes appropriate local semantics from a globally shared conceptualization (the application ontology), i.e., minimizing *syntactical dependencies*. The word 'appropriate' is used to reflect the fact that each component will use their own interpretation of a domain conceptualization on behalf of their own purpose. In addition, the *ontology mappers* are used to translate or interpret from the local ontology (local syntactic and structural representations) into the application ontology and vice versa. From the development perspective, by using the ontology mapper, a new component (e.g. a sensor driver) can be integrated into the platform easily. This does not require re-implementing the component; only the ontology mapper requires an updated specification of how local terminologies and structures that refer to the new sensor can be represented in terms of *Meas_system* and *Measurand* that are used by the application ontology. Since each functional layer, e.g. the SAL, provides for rather homogeneous services and syntax, the mappers can also be implemented based on patterns that are generic for each functional layer.

We consider an example of two system components, the ECG sensor driver that collects data from the ECG sensor and the application that visualizes the data received from the driver. Figure 4 depicts the sequence diagram of data sharing between the sensor driver and the application via their ontology mappers. The sensor driver implements the local database to store persistent data (if this is not required, the driver sends data directly to the mapper). The sensor mapper gets ECG data by using *getECGData* method, which implements the SELECT query to select data from the local database. The received data are translated into the UoD messages by the *localToDomain* method. These messages then are sent to data subscribers (the application) that subscribed to the ECG data. When the application mapper receives the messages, it translates these into the local ontology messages (or data format) of the application by using the *domainToLocal* method. The application mapper can also implement a message filter that accepts only subscribed data messages sent by verified data publishers. The local ontology messages then are sent to the application for further processing, such as data visualization.

2.3 Data storage and sharing

A common way to work with ontology data is to load the ontology components (e.g. classes, instances) at run-time from a set of source documents (e.g. OWL files). This is a very flexible approach, but it has limitations for efficiently processing large amounts of instance data since the application must parse the source documents each time it is run. Another approach is to store the ontology data persistently in a database. This saves the loading overhead and means that we can store large volumes of ontology data, but it has the expense of a higher overhead to retrieve and update data. There are several approaches to store the ontology data and execute queries on that data in a database [6].

We consider two architectural approaches for the purpose of data storage and sharing. In the first approach, a centralized ontology database is used to store both ontology-based data (e.g. sensor data) and their respective meanings (ontologies). This approach is similar to the architecture in [3], albeit a more formal application ontology is employed. The use of the centralized database may facilitate the data access control and management, but it has some issues. Using a single schema impedes loose-coupling between applications and services, whilst the approach that accepts local schemata indeed requires mappers but provides a loosely-coupled architecture. The database could become a performance bottleneck of the platform when there are too many modifications to the database. Extra effort is needed in developing the ontology database. Moreover, the procedure of updating ontologies is complex.

To address the above issues, an alternative approach of using decentralized data stores is proposed in Figure 5. This architecture conforms to the layered architecture of the existing platform. The grey box represents the components of the Basic Platform, which provide necessary services for other components (e.g. applications) running on the platform. For example, the Configuration Manager is responsible for handling system configurations and installing applications, the Security Interface and the Trust & Ownership

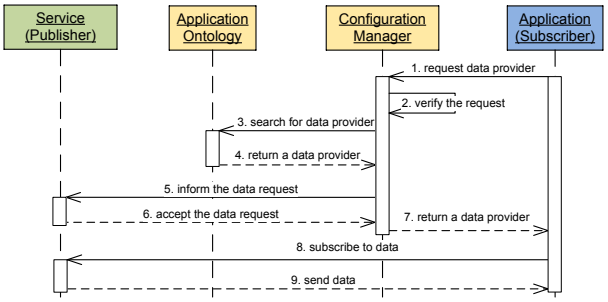

Figure 6: Example of data sharing between an application and a service.

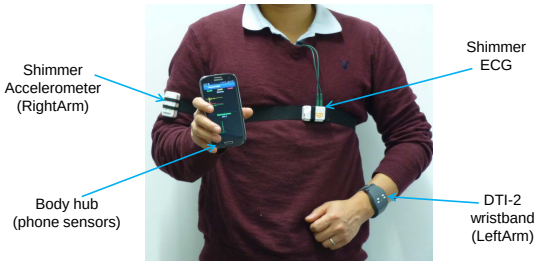

Figure 7: Configuration of the BASN consisting of the Android phone and sensor devices.

Manager are responsible for authenticating the downloaded software and evaluating the system trustworthiness.

In comparison to the former architectural approach, the ontologies are stored separately in the application ontology repository. This has advantages: i) a good performance of loading, retrieving, and updating data since the data storage is more lightweight; ii) no extra implementation efforts are needed; and iii) flexibility of updating and modifying ontologies. Besides, each component (sensor driver, service or application) uses its local data store (not shown) for storing data. This avoids the issue of performance bottleneck. The modification of the data schema can be easily performed and authorized by the data store owner. Also, it simplifies the reuse of existing applications and services since they can maintain their original data storage mechanism (e.g. relational database or file system). In the former architecture, when a new application is installed, its data schema and tables need to be created in the centralized database.

This approach, however, has practical issues that we need to address: 1) how can data in local data stores be shared? and 2) how does a component know which component it contacts for a particular type of data? For example, the heart monitoring application needs to find a proper service that can provide heart rate data.

To address the first question, we use the publish-subscribe pattern and message-based communication. The publish-subscribe pattern provides a loosely-coupled form of interaction among system components. The publisher has useful data or events, on which interested subscribers can subscribe in order to get data or stay synchronized with the publisher. A publisher can share data to multiple subscribers by maintaining a list of subscribers for a certain subject. A subscriber can also subscribe to data from multiple publishers. For example, the epilepsy monitoring application can subscribe to data from both the accelerometer sensor driver and the heart-rate detection service. The subscriber that wants to receive data from a publisher needs to have a permission granted by the platform. Moreover, the subscriber can implement a filter method that accepts only interested data sent by authenticated publishers.

To address the second question, we apply the broker pattern, in which the broker mediates communication (e.g. data requests and responses) among the system components. The Configuration Manager plays a role of the broker that verifies and serves the requests from the subscribers (e.g. a data subscription) and the publishers (e.g. a publishing advertisement). Figure 6 shows the interaction among the publisher (service), the subscriber (application) and the Configuration Manager. The verification process is performed with the help of the Security and Trust Manager (not shown). When a new publisher is added to the platform, the publisher advertises its metadata with the Configuration Manager. If the publisher is verified, its information (e.g. publishing data) is stored into the application ontology. When a subscriber wants to subscribe to a particular data, it sends a request to the Configuration Manager, which then queries the application ontology for the matching publisher of such request. Upon the response from the Configuration Manager, the subscriber communicates directly to the publisher for the data subscription. The data messages exchanged between the publisher and the subscriber are handled by their ontology mappers (not shown), which translate between the local data format and the domain data format (from the syntactic perspective) and between the local semantics and the domain semantics (from the semantic perspective).

3. IMPLEMENTATION

3.1 Platform configuration

Figure 7 shows an example of the platform configuration consisting of the Android phone (body hub) and three sensor devices: the Shimmer ECG worn on the chest, the Shimmer accelerometer worn on the right arm and the DTI-2 wristband worn on the left arm. We chose the Android platform for implementing the body hub since Android is an open source mobile operating system used by many devices on the market. The Android smart phone provides a wide range of communication protocols, e.g. WLAN and Bluetooth (or Bluetooth LE). The smart phone also provides sensors, e.g. accelerometer, gyroscope and GPS. The Shimmer and DTI-2 devices are chosen since they are wearable and configurable, support Bluetooth communication and provide several types of sensor data. The DTI-2 wristband integrates the accelerometer, skin conductance, skin temperature, environment temperature and light sensors.

The Basic Platform, which provides the services for other components and the GUI, can be downloaded and installed as a regular Android app. The GUI (see Figure 9) allows the users to manage their BASN, such as adding and configuring a new sensor or installing a new application. The sensor driver, service and application can also be installed as regular apps. For the demonstration and evaluation, we implemented several apps that can be downloaded (via scanning a QR-code) from our App-server at the link:
http://www.win.tue.nl/san/projects/vitruvius/download.php

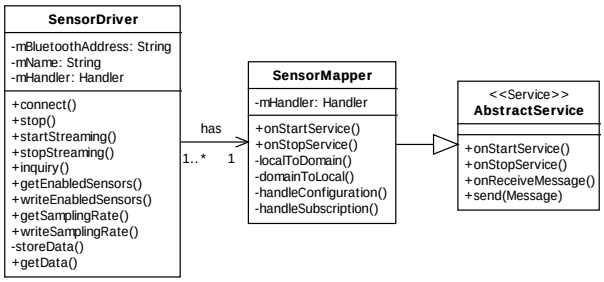

Figure 8: A simplified class diagram of the sensor mapper and the sensor driver.

3.2 Application ontology

Many ontology representation languages (e.g. RDF, OWL, UML, KIF and F-logic) have been proposed [10]. We chose to use the OWL DL (Description Logics) as the ontology representation language for the application ontology implementation since it meets our requirements on both the semantic aspects and the practical aspects. OWL is adopted as the W3C standard for the development of the semantic web; it is popular and well supported. OWL DL supports the maximum expressiveness without losing computational completeness (all entailments are guaranteed to be computable) and decidability (all computations will finish in finite time), which is important for the resource constrained mobile platform. In addition, the Java APIs for OWL and the SPARQL-DL query language that provides a query engine on top of the OWL API, are available for the Android platform. These APIs are fully aligned with OWL-2, the new version of the original OWL standard. The implementation of the domain-specific ontology model (e.g. the measuring system) is performed by using Protégé, which is an open source ontology editor and a knowledge acquisition system with full support for the OWL-2 ontology language. The local ontology (e.g. local data) is implemented by SQLite, which is a lightweight SQL database supported by the Android platform.

3.3 Ontology mapper and message

The mapping between the application ontology and the local ontology is performed by the mapper component, which also provides interfaces (APIs) for the data communication and control (e.g. sensor configuration). The mapper is implemented as an Android service, which can quickly be implemented using a generic development pattern. Figure 8 shows the simplified class diagram of the sensor mapper and the sensor driver. The *SensorMapper* class is extended from the *AbstractService* class. The *send(Message)* and *onReceiveMessage* methods implement the message-based communication. The *localToDomain* and *domainToLocal* methods are used to translate between the local and domain ontologies. The *SensorMapper* calls the methods provided by the *SensorDriver* for the purpose of streaming data or configuring the sensor. The mappers of the service and the application are implemented similarly.

The message-based communication is implemented based on two message classes provided by the Android platform, including the *Intent (android.content.Intent)* and the *Message (android.os.Message)*. The *Intent* object is a passive data structure, which can be used to activate Android activities, services and broadcast receivers. The *Intent* messaging is a facility for late run-time binding between components in the same or different applications. For example, the Basic Platform can use the *Intent* messaging to launch or to communicate with a sensor driver or a data processing service. The *Message* object is used for other purposes, such as the advertisement of a data publisher, the configuration of a sensor and the data sharing. The *Message* object contains the *'what'* field used to identify different types of messages.

4. QUALITATIVE EVALUATION

We evaluate the semantic interoperability of the platform with three aspects: 1) semantic expressiveness; 2) extensibility (e.g. adding a new type of sensor); and 3) data sharing.

4.1 Semantic expressiveness

We adopted the ontology-based approach to support the semantic interoperability of the platform. Instead of using the database schema as in [3], the OWL DL ontology language is used to implement the application ontology. The high expressiveness of the ontology language provides a more accurate and precise representation of the domain knowledge, which limits the ambiguity during knowledge sharing. According to a continuum of kinds of ontologies [15], the very lightweight ones (at the extreme) may consist of terms only, with little or no specification of the meaning of terms. As we move along the continuum, the amount of meaning specified and the degree of formality increase (OWL DL is a kind of Description Logic); the support for automated reasoning also increases. Although the SQL engine of the relational database can be used to perform reasoning, it is highly specialized and tuned for answering queries and ensuring data integrity. The fundamental role of a reasoning engine for ontologies is, however, to derive new information via automated inference. For example, we define the *Meas_model* to be the composition of *Input_quant*, *Output_quant* and *Meas_func*. An ontology can derive that a new instance, which is identified to have these three parts, can be automatically classified to be a *Meas_model*.

4.2 Extensibility

As mentioned, the semantic interoperability indicates that the platform is easily extended to add a new type of sensor or application. A successful addition of a sensor means that it should be properly configured and used by the platform. The platform allows adding a new sensor by installing its driver as a regular app. The ontology mapper of the sensor driver simplifies the development and integration processes. With a new sensor driver, only the mapper needs to be developed to map the local ontology of the driver to the application ontology and vice versa. The mapper pattern can be used to make the development even simpler.

Figure 9 (left) shows the three sensor devices (namely ECG, LeftArm and RightArm) that are connected to the platform. The Bluetooth addresses of these devices are shown on the right column. When a new device is connected, its configuration properties (e.g. available sensors, sampling rate) are advertised to the Configuration Manager and stored into the application ontology. Depending on the context, an application may request a particular configuration. Figure 9 (right) shows five available sensors of the DTI-2 device, but only the *Acceleration* and *SkinConductance* sensors are enabled for sending data.

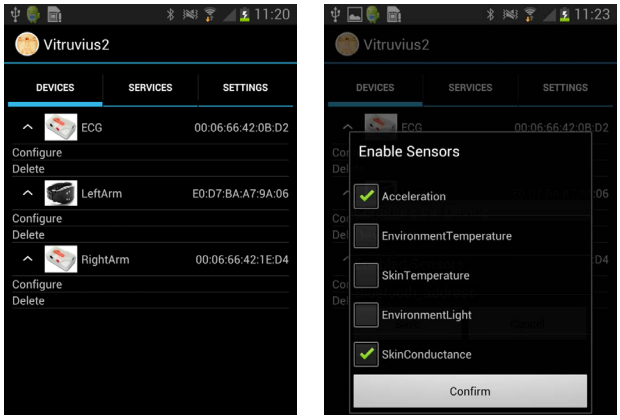

Figure 9: A GUI of the Basic Platform shows the sensor devices (left) and the configuration (right).

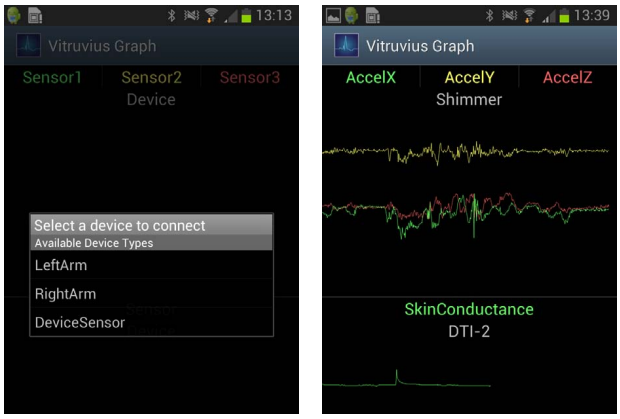

Figure 10: The app's GUI shows the available acceleration sensors (left) and data visualization (right).

4.3 Data sharing

We evaluate the capability of data sharing between sensor drivers and an application. The VitruviusGraph application, which is able to visualize data received from multiple sensor drivers, is then installed on the platform. Assuming that a user wants to visualize acceleration data, the application then sends a request to the Configuration Manager (the broker) to determine which sensor device can provide the acceleration data. The Configuration Manager then queries the application ontology and returns a list of available devices. Figure 10 (left) shows the returned devices on the GUI of the application. The devices include the LeftArm (DTI-2 wristband), the RightArm (Shimmer accelerometer) and the DeviceSensor (phone accelerometer). Assuming that the user selects the RightArm device, the application then sends a data subscription message to the sensor driver. The application may also send a configuration request (e.g. set the sampling rate) to the driver. The received acceleration data are visualized on the GUI as shown in Figure 10 (right). Similarly, the user then selects the device to visualize the skin conductance data. In this case, only the LeftArm device that provides such data is returned. The skin conductance data are visualized at the bottom of the GUI.

5. CONCLUSION

We have presented an ontology-based architecture of the BASN platform to improve the semantic interoperability among sensor networks and applications. We have analyzed the architecture regarding the principles of separation of concerns and semantic transparency. Moreover, we have discussed the design choices of the data storage and our reference implementation of the application ontology and the message-based communication for data sharing.

We have evaluated the platform through aspects of semantic expressiveness, extensibility and data sharing. It was shown that such platform design and implementation indeed provide significant improvements regarding semantic interoperability. By using the ontology mapper, the semantic interoperability can be achieved with minimized mutual dependence. Thus, the development and integration processes of adding a new type of sensor or application are simplified. We have further verified the functions of configuring a sensor and sharing data among sensors and applications. In future work, we want to implement the complete application ontology and evaluate the semantic interoperability with a quantitative evaluation.

6. REFERENCES

[1] P. Brandt, T. Basten, and S. Stuijk. Contoexam: an ontology on context-aware examinations. In *FOIS*, 2014.

[2] P. Brandt, T. Basten, S. Stuijk, V. Bui, P. de Clercq, M. Iacob, L. F. Pires, and M. van Sinderen. Semantic interoperability in sensor applications - making sense of sensor data. In *IEEE Symp. on Comp. Intelligence Healthcare and e-health*, pages 34–41, 2013.

[3] V. Bui, R. Verhoeven, and J. Lukkien. A body sensor platform for concurrent applications. In *Proc. of IEEE Int. Conf. on Consumer Electronics*, pages 38–42, 2012.

[4] M. Chen, S. Gonzalez, A. Vasilakos, H. Cao, and V. C. Leung. Body area networks: A survey. *Mobile Networks and Applications*, 16(2):171–193, Apr. 2011.

[5] M. Compton, C. Henson, H. Neuhaus, L. Lefort, and A. Sheth. A Survey of the Semantic Specification of Sensors. In *Int. Workshop on Semantic Sensor Networks*, pages 17–32, 2009.

[6] A. Gali, C. X. Chen, K. T. Claypool, and R. Uceda-sosa. From ontology to relational databases. In *ER Workshops 2004, LNCS 3289*, pages 278–289. Springer-Verlag, 2004.

[7] N. Guarino. *Semantic matching: Formal ontological distinctions for information organization, extraction, and integration*, volume 1299 of *LNCS*, pages 139–170. SpringerSpringer Berlin Heidelberg, 1997.

[8] N. Guarino, D. Oberle, and S. Staab. What is an ontology? In S. Staab and R. Studer, editors, *Handbook on Ontologies*. Springer, second edition, 2009.

[9] S. Heiler. Semantic interoperability. *ACM Comput. Surv.*, 27(2):271–273, June 1995.

[10] M. Lenzerini, D. Milano, and A. Poggi. Ontology representation & reasoning. Technical Report IST-508011, Universit di Roma La Sapienza, 2004.

[11] T. Mens and M. Wermelinger. Separation of concerns for software evolution. *Journal of Software Maintenance and Evolution: Research and Practice*, 14(5):311–315, 2002.

[12] S. Movassaghi, M. Abolhasan, J. Lipman, D. Smith, and A. Jamalipour. Wireless body area networks: A survey. *Communications Surv. & Tutor., IEEE*, PP(99):1–29, 2014.

[13] M. Patel and J. Wang. Applications, challenges, and prospective in emerging body area networking technologies. *Wireless Comm., IEEE*, 17(1):80–88, 2010.

[14] Shimmer. Shimmer sensors, 2013.

[15] M. Uschold and M. Gruninger. Ontologies and semantics for seamless connectivity. *SIGMOD Rec.*, 33(4):58–64, 2004.