Knowledge and Information Systems manuscript No.
(will be inserted by the editor)

The MASSIF Platform: a Modular & Semantic Platform for the Development of Flexible IoT Services

Pieter Bonte, Femke Ongenae,
Femke De Backere, Jeroen Schaballie,
Dörthe Arndt, Stijn Verstichel, Erik Mannens,
Rik Van de Walle and Filip De Turck

Received: Nov 27, 2015 / Revised: May 03, 2016 / Accepted: May 14, 2016

Abstract In the Internet of Things (IoT), data-producing entities sense their environment and transmit these observations to a data-processing platform for further analysis. Applications can have a notion of context-awareness by combining this sensed data, or by processing the combined data. The processes of combining data can consist both of merging the dynamic sensed data, as well as fusing the sensed data with background and historical data. Semantics can aid in this task, as they have proven their use in data integration, knowledge exchange and reasoning. Semantic services performing reasoning on the integrated sensed data, combined with background knowledge, such as profile data, allow extracting useful information and support intelligent decision making. However, advanced reasoning on the combination of this sensed data and background knowledge is still hard to achieve. Furthermore, the collaboration between semantic services allows to reach complex decisions. The dynamic composition of such collaborative workflows that can adapt to the current context, has not received much attention yet.

In this paper, we present MASSIF, a data-driven platform for the semantic annotation of and reasoning on IoT data. It allows the integration of multiple modular reasoning services, that can collaborate in a flexible manner to facilitate complex decision making processes. Data-driven workflows are enabled by letting services specify the data they would like to consume. After thorough processing, these services can decide to share their decisions with other consumers. By defining the data these services would like to consume, they can operate on a subset of data, improving reasoning efficiency. Furthermore, each of these services can integrate the consumed data with background knowledge in its own context model, for rapid intelligent decision making. To show the strengths of the platform, two use cases are detailed and thoroughly evaluated.

Keywords IoT · Data-driven platform · Semantic Web · Reasoning · Ontologies · SOA

Ghent University - iMinds, Technologiepark-Zwijnaarde 15, 9052 Ghent, Belgium
E-mail: Pieter.Bonte@intec.ugent.be

1 Introduction

1.1 Background

In the Internet of Things (IoT) paradigm, numerous things are connected to the Internet [6]. Through interactions with these connected things, specific goals can be reached that support our daily tasks. The data these things transmit, originates from numerous heterogeneous sources, each sensing a part of the environment. Combining data from different sources facilitates applications to support context awareness [8]. This enables applications to understand the given situation. For example, to allow elderly people to stay at their own home as long as possible, fall detection systems combine multiple sensors with background knowledge, such as the profile of the elderly. A high fall detection precision can be achieved by combining the profile, habits and whereabouts of the elderly with multiple sensors, such as motion and pressure sensors. The IoT aims at creating intelligent systems that can support people as much as possible during their daily activities. To achieve this awareness, understanding the raw sensor data is necessary. Collection, modeling, reasoning, and distribution of context in relation to sensor data plays a critical role in order to tackle this challenge [45].

Context-aware systems can acquire, interpret and use context information to adapt their behavior to the current context [12]. They have played an important role in tackling this challenge in previous paradigms. Their proven previous success makes them a solution that is ought to be successful in the IoT paradigm as well [45]. According to Perera, et al. [45], one of the important design principles for context-aware systems is scalability and extensibility. Gartner[1] expects 20 billion connected things to be in use worldwide by 2020. Thus, it should be straightforward to add new sensors and devices to a context-aware system. Additional sensors and devices produce new data that might need to be processed differently. Consequently, it should be possible to easily add extra processing services to extract high-level knowledge. Following this thought, the number of services and applications handling the produced IoT-data will increase rapidly. Consequently, context-aware platforms for the IoT should be easily extensible.

According to Strang, et al. [52], semantics are the preferred mechanism of managing and modeling context. Semantics can aid in the integration of the generated heterogeneous IoT data by enabling interoperability between different sources and providing a uniform model [52,10]. For example, the profile information of the elderly, the floor plan of the house and the sensor readings have different sources and data formats. Combining them within the semantic model enables interoperability. A concise introduction to semantics can be found in Section 1.2.3. However, analysis and mapping of data to the semantic model has to be handled for each source individually. Combining the domain information, such as the sensor readings, with profile information, allows to make personalized decisions.

Semantic reasoning allows to compute logical consequences defined in the semantic model. For example, the model could define an alarming fall as a sensor reading from a fall detection sensor with an accuracy above a certain threshold

[1] http://www.gartner.com/

(e.g., 78%) resulting from a sensor in the home of a resident with a profile that states that the patient is in a wheelchair. When such a sensor reading is detected by the reasoner, it will know it has detected a fall and someone should be called to assist the patient, even when it is not explicitly stated in the sensor data. Utilizing semantic reasoning enables transforming the integrated low-level data into high-level knowledge, allowing accurate and intelligent decisions. However, expressive logics such as Description Logic (DL)-Reasoning have EXPTIME complexity [30], resulting in slow reasoning times with growing datasets [26,3]. This is inconsistent with the vision that events in the IoT should be processed in a timely manner [8].

1.2 Related Work

The following section describes existing context-aware and IoT frameworks. Ontologies are discussed and considered to be the most used semantic model in current practice. The discussion of the context-aware and IoT frameworks will focus on four important aspects:

1. The capability to semantically annotate raw data. To be able to extract useful knowledge from the IoT-data, the data needs to be semantically annotated first [55].
2. Inference techniques. The extraction of knowledge is an important instrument in the IoT [8]. More advanced techniques allow to extract more complex knowledge.
3. Context model. Platforms can utilize a central context model that contains all context information in one central knowledge base for easy access, a duplicated context model for resilience or a distributed context model for efficiency.
4. Service Collaboration. Service composition provides functionality to build a specific (IoT) application, which is composed of various independent services [45]. By allowing services to collaborate, more complex tasks can be tackled.

1.2.1 Context-Awareness

A system is context-aware if it uses context to provide relevant information and/or services to the user, where relevancy depends on the user's task [1,7]. Context-awareness frameworks typically support acquisition, representation, delivery and reaction [19].
Various methods have been proposed to model context information. The six most popular are: key-value modeling, markup scheme modeling, graphical modeling, object-based modeling, logic-based modeling and ontology-based modeling. According to many surveys in context-aware computing, ontologies are the preferred mechanism of managing and modeling context [45,52].
Over 30 distinctive context-aware systems have been developed, each providing a different kind of system. An exhaustive analysis of these systems can be found in Perera, et al. [45] and Li, et al. [34]. The most recent and related semantic context-aware platforms are elaborated upon in the following paragraphs.

SeCoMan [29] is a context-aware platform, designed to provide privacy-preserving solutions in the design of context-aware services. These services or applications are

in charge of managing their own context for security issues. The privacy handling itself is implemented using a rule-based approach. However, the platform is aware of locations only, other sensor data cannot be semantically annotated and incorporated.

CoCaMAAL [22] is a cloud-oriented context-aware middleware solution for Ambient Assisted Living (AAL). It is able to annotate and abstract raw data from the AAL systems, based on pre-designed ontologies. Service providers enable specific applications based on the context-aware middleware. They can subscribe to the context-aware middleware by providing service rules. However, these service providers do not collaborate. The context model utilized in the context-aware middleware is duplicated in the cloud, however, it is not possible to isolate the context model for the various services providers.

CASF [32] is a framework for context-aware service discovery and integration. It consist of 3 layers: (i) a physical sensor layer which captures the raw sensor data, (ii) a public context layer that processes the sensor data and administers various context providers, and (iii) a context service layer, which consumes context information from one or more context providers. The context services can consume the provided context, but are not capable of sharing conclusions. The platform uses Web Services for automatic discovery and integration of context information.

Although these frameworks look promising, they do not provide advanced reasoning capabilities, such as description logics, and lack the capability to coordinate high-level workflows.

1.2.2 IoT Frameworks

IoT frameworks serve as a middleware solution to provide connectivity for sensors and actuators to the Internet. Numerous IoT frameworks exist, most of them focus on the integration of the devices and sensors, less attention is given to intelligent data processing of IoT data [8]. The following paragraphs discuss recent attempts to process IoT data, more specifically through the use of semantics.

Patkos et al. propose an ambient intelligence framework that combines rule-based reasoning with causality-based reasoning, to reason about actions and causalities [44]. The proposed framework does not provide capabilities to annotate raw IoT data.

The LinkSmart platform [33] was designed to support interoperability and integration of various devices, sensors and services. It provides an abstraction of the devices and sensors as regular programming objects towards the application layer. It allows to compose workflows through the use of business rules to optimize the service composition.

Gray et al., propose a system to annotate and integrate heterogeneous streaming data with stored data, through the use of ontologies [23]. Their approach focuses on the discovery and integration of data sources, both static as streaming data. Reasoning and service collaboration techniques are not presented.

Sense2Web [17] is a multilayer platform, allowing to annotate and integrate sensor data in the form of Linked Data and makes it available to other Web applications via SPARQL endpoints. Its data source layer is modeled using expressive ontologies, allowing high-level data retrieval. However, service collaboration and

advance reasoning capabilities are not provided for the service and application layer.

XGSN [13] is an end-to-end, semantic-enabled IoT platform that allows to semantically annotate sensors and processes the produced sensor stream using the Linked Sensor Middleware (LSM). The stream can be archived or processed using stream processors. External application can query the context through Web Services. However, these applications cannot share conclusions back to the context layer.

Ali et al. propose an IoT-enabled communication system that allows the annotation of sensory data through the use of XGSN and the continuous analysis of data streams through the use of a stream query processing module for the detections of events [4]. These events can then be further processed in the Stream Reasoning component that can infer implicit semantic statements. Applications can subscribe to events generated in the centralized stream processing and reasoning layer. However, they cannot collaborate to achieve complex workflows.

OpenIoT [51] is an open source IoT platform enabling the semantic interoperability of IoT services in the cloud. It allows the integration and annotation of virtually any sensor. LSM is utilizes and acts as a cloud database which enables the storages of the annotated data streams. Services can access the the annotated data through the use of SPARQL queries. However, service collaboration and advanced reasoning capabilities are lacking.

SOFIA2 [31] is an ontology-based Big Data IoT middleware, allowing interoperability and semantic annotation of multiple heterogeneous devices. It facilitates Complex Event Processing (CEP) to orchestrate the context-data between context consumers. However, besides context subscription based on CEP, there is no real semantic reasoning possible.

These frameworks can annotate data semantically and draw conclusions in an efficient reactive manner. Efficient processing of data is often provided, however advanced reasoning capabilities are still missing. Furthermore, these platforms fail to mutually collaborate in a high-level manner. An overview of the discussed ontology-based context-aware and IoT platforms is summarized in Table 1.

1.2.3 Ontology

Gruber [24] defined an ontology as "an explicit specification of a conceptualization". An ontology formally describes concepts, properties and their relations, within a certain domain, that can easily be reused [46, 49] in different settings. This allows to model knowledge and make data machine-readable. The Web Ontology Language (OWL) [9] is the most popular language to describe ontologies. Ontologies form an excellent model to integrate data, exchange knowledge and to reason upon that enhanced information. The ontology's Terminology Box (TBox) describes the concepts and their relations, while the Assertion Box (ABox) contains the data instances with respect to the TBox. The concepts are also called classes and the relations between classes are called object properties. Data properties describe the relation of a concept to a specific data value. The ABox consists of entities of the defined classes (the individuals), their relations and their data properties (the literals). An OWL ontology is described as a collection of axioms, e.g., the class axioms describe the concepts in the ontology in a formal manner. OWL

Table 1: Comparison of existing ontology-based context-aware and IoT platforms.

	Year	Semantic Annotation	Inference	Context Model	Service Collaboration
Patkos et al. [44]	2010	/	Rules & Causality	Central	/
LinkSmart	2011	✓	Rules	Central	✓
Gray, et al. [23]	2011	✓	/	Central	/
Sense2Web	2012	✓	/	Central	/
SeCoMan	2013	locations	Rules	Distributed	/
CoCaMAAL	2014	✓	Rules	Duplicated	/
CASF	2013	✓	/	Distributed	/
SOFIA2	2014	✓	/	Central	✓
XGSN	2014	✓	Basic Stream Processing	Central	/
Ali et al. [4]	2015	✓	Stream Processing & Stream Reasoning	Central	/
OpenIoT	2015	✓	/	Central	/
MASSIF	2016	✓	OWL DL	Distributed	✓

contains multiple sublanguages, each of which varies between expressivity and complexity. The sublanguages are listed below with increasing expressivity [35]:

1. OWL-Lite: supports classification and simple restriction functions.
2. OWL-DL: the largest subset without the loss of computational completeness.
3. OWL-Full: maximal expressivity and syntactical freedom. However, the reasoning process might not be computable.

The popularity of OWL has led to the creation of three OWL 2 profiles [37], each offering a specific subset of the overall expressivity to obtain advantages in specific applications. Each profile is a subset of the OWL DL sublanguage.

1. OWL 2 Existential quantification Language (EL) is useful in applications utilizing an ontology that contains a large number of properties and/or classes.
2. OWL 2 Query Language (QL) is created for applications where query answering is the main task.
3. OWL 2 Rule Language (RL) provides scalable reasoning without sacrificing too much expressiveness.

1.3 Objectives

There are still many challenges left to tackle in the IoT-paradigm. Over the past years, efforts in IoT have mainly focused on developing infrastructures to collect and communicate IoT data. Less attention was given to intelligent data processing of this data [8]. The aim of this research is to propose a platform for reactive and real-time data processing, that complies with the following objectives:

– **Semantic Annotation**: To be able to extract useful knowledge from the IoT-data, the data needs to be semantically annotated first [55]. Since it cannot be expected that all sensors and devices generate semantically annotated data natively, it should be possible to enrich raw (sensor) data according to the semantic model.

– **Knowledge Extraction**: The extraction of knowledge is an important instrument in the IoT [8]. It allows to analyze the data, infer new data and to abstract the data for easy data consumption [2]. Utilizing advanced reasoning capabilities, such as description logics, allows the extraction of intelligent high-level conclusions and the execution of intelligent decisions.
– **Extensibility**: To be able to cope with the ever growing amount and types of connected sensors and devices, IoT platforms should be extensible [45]. This allows adding new functionality without altering the existing components. Thus, a plug-in architecture is mandatory.
– **Performance**: Due to the fact that the produced data in the IoT is only temporary valid, a timely processing is required [8].
– **Scalability**: Cisco[2] expects there will be more than 50 billion devices connected to the Internet by 2050. The processing of a growing number of connected devices requires a scalable platform.
– **High-level Workflows**: IoT data consumers are often interested in the high-level concepts, such as concepts higher in the class hierarchy of ontologies or implicit concepts requiring reasoning to infer [8]. Service composition provides functionality to build a specific (IoT) application, which is composed of various independent services. The composition of these services, through the use of workflows, should be possible base on these high-level concepts [45].
– **Real-time Processing:** Many solutions provide IoT analysis tools [36]. However, to detect and immediately react to events, real-time processing of the IoT data is necessary [8].

1.4 Paper Organization

The remainder of this paper is structured as follows. Section 2 highlights our contribution and provides an overview of the proposed platform. Section 3 elaborates on the implementation-specific details of the platform. Two use cases are explained in Section 4 to illustrate the capabilities of the platform to derive useful information in a timely manner. These use cases handle data originating from a home care and media setting. The limitations of the platform are discussed in Section 5. Section 6 highlights the conclusions and introduces tracks for the future work.

2 The MASSIF Platform

This section highlights the novelty of the presented work and provides an architectural overview of the platform.

2.1 Our Contribution

In this paper, we present the MASSIF platform (ModulAr, Service, Semantic & Flexible platform). It is designed for rapid enrichment and reasoning on IoT data. It uses ontologies to represent context information and different kinds of reasoning can be enabled to derive high-level knowledge. To be able to cope with any

[2] http://www.cisco.com/

kind of input data, the platform allows to *semantically annotate* raw data. Once the data is annotated, it can be combined with static context data. The use of semantic reasoning allows data consumers to *extract useful knowledge*, make intelligent decisions and take appropriate actions. Furthermore, the data consumers that extract this knowledge can become data producers and share their findings with other data consumers. This allows the creation of *workflows*. Since abstractions and high-level concepts are mandatory to create complex workflows, reasoning is performed to coordinate the data flow. To allow dynamic and context dependent workflows, we propose a data-driven workflow composition, where data consumers define the data they would like to receive, based on high-level concepts. When data is processed by the platform, it will check which data consumers are interested in the particular type of data. These types of workflows allow loosely-coupled modular services, which enable *extensibility and scalability*. Furthermore, a distributed context model is utilized. Each of the data consumers manages its own context. The distributed context model, combined with the data-driven workflows allows each data consumer to operate on a subset of data. Minimizing the dataset enables more effective reasoning, even when utilizing logics such as description logics.

MASSIF is a reactive data-driven platform, in the sense that it reacts to the received data and handles accordingly. This eliminates the need for active polling the various MASSIF components for updates or actions.

2.2 The Platform Architecture

The platform consists of five types of components, whereof two of them can be extended to provide specific functionality in each use case. These are called API-components and can be distinguished with the dotted lines in Figure 1.

The API-components consist of the *Context Adapters*, which can semantically annotate the data and *Services*, which process the semantic data to retrieve high-level knowledge.

The various components that make up the MASSIF Platform are discussed below. The explanation will start with the Semantic Communication Bus (SCB)[21], since it regulates the data flow within the platform.

1. The *SCB* provides a publish-subscribe mechanism based on high-level ontology concepts. The services can subscribe by defining what kind of data they would like to consume. These definitions are called *semantic filter rules* and are in fact OWL class expressions. The services can be both consumers and producers. They can decide to share their conclusions by publishing their findings on the SCB. The SCB has its own context model and utilizes reasoning on the subscribed filter rules and the published data to determine which services subscribed to the published data. Note that through the use of reasoning, services can define their input data in an abstract and high-level manner.
2. The *Gateway* serves as the primary communication interface of the platform. It allows both input and output with external devices.
3. The *Matching Service* inspects the raw data that have been sent from an external source to the *Gateway*. It selects a *Context Adapter* that is able to annotate the low-level data according to the semantic model.
4. A *Context Adapter* receives low-level data from the *Matching Service* and semantically annotates it. Multiple *Context Adapters* can be active to annotate

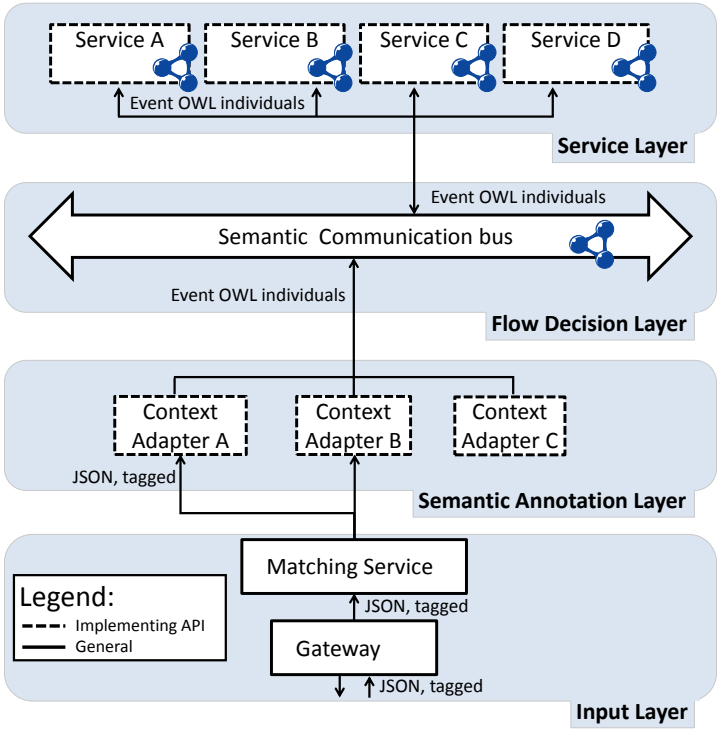

Fig. 1: Conceptual Architecture of the MASSIF platform.

numerous kinds of raw data. Once the data is converted to OWL individuals, it is pushed on the *SCB*. The platform also allows Virtual *Context Adapters*. These context adapters do not receive data from the *Matching Service*, but annotate data they capture from existing sources, such as Twitter streams.

5. A *Service* subscribes to the *SCB* with one or more filter rules. These filter rules describe the data that the *Service* would like to consume. Each *Service* performs a distinct reasoning task or algorithm and can share its inferred knowledge with other *Services*, over the *SCB*.

Note that the *SCB* and the *Services* each contain their own ontology model and can preload it with background knowledge, such as profile data.

3 Implementation Details

The following section explains the introduced components from Section 2 in greater detail. First, a running example is presented that will be used to clarify the further details of the components in the platform. Second, additional implementation details about the used technology are given in Section 3.2. Third, more clarification is provided in Section 3.3 on how the ontologies are internally represented. Finally, each component is described in great detail in Section 3.4, based on the provided information from the first three subsections.

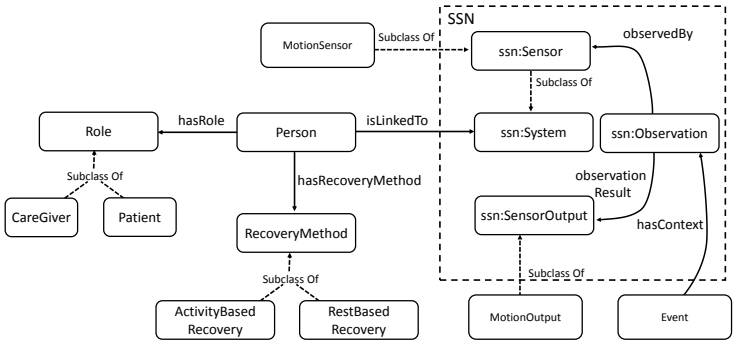

Fig. 2: Overview of used extension of the SSN ontology.

3.1 Running Example

To further explain the various components, a running example is introduced to provide practical insights. In the example, a motion sensor is integrated in the home of a patient. The patient should be active for a certain amount of time, to fully recover. The exact upper and lower bound of the allowed active time is patient- and situation-dependent and should not be exceeded. The sensor will capture the activity of the patient and send it to the platform, that will compare it against the background knowledge, which describes the profile of the patient. If the platform detects that not enough/too much activity has been reached, an alarm is triggered.

To model the different concepts and relations for the running example, the Semantic Sensor Network (SSN) [15] ontology is utilized. Figure 2 shows the TBox of the extended SSN ontology, describing the designed concepts and their relations. The MotionSensor, MotionOutput and Observation concept are used to model the motion sensor readings.

3.2 OSGi

The platform has been developed utilizing OSGi [41], which is an extra layer on top of the Java Virtual Machine, enabling modularity. It allows components to be dynamically added, even at run time. This enables our platform to be extended with additional *Context Adapters* or *Services*, even when the platform is fully operative. Additionally to the enabled extensibility, OSGi allows straightforward scalability. All components are *OSGi Services*, which can be distributed utilizing *Distributed OSGi* [41].

3.3 Ontology Representation

The ontologies are internally represented using the OWL API [27]. Data is shared, over the SCB, as a set of OWLAxioms, describing the data semantically. The OWL API provides an *OWLReasoner*-interface, which is implemented by numerous

popular reasoners [43] such as Pellet [50], Hermit [48], Fact++ [53], JFact [42], Chainsaw [54] and RacerPro [25]. This allows services to choose which reasoner to utilize. Various reasoners provide different functionality, complexity and efficiency.

3.4 MASSIF Implementation

Since it cannot be expected that all data-producing entities transmit their sensory observations semantically annotated, the platform can annotate raw sensor data itself. It is assumed that each sensor is capable of transmitting its raw data in JavaScript Object Notation (JSON) [16], annotated with a certain *tag*, which provides some extra information about the origin of the data. The tag itself is added by the sensor gateway. The modular structure of the platform allows to extend the platform in order to accept different input formats. If sensors are able to transmit semantic data, the semantic annotation step can be skipped.

3.4.1 Gateway

Low-level data, in JSON format, enters the platforms through the *Gateway*, as depicted at the bottom of Figure 1. Since the platform is fully data-driven, devices can push their data to the platform. The Gateway serves as an entry point and forwards the data to the *Matching Service*.

3.4.2 Matching Service

The *Matching Service* analyses the data and decides which *Context Adapter* can semantically annotate the received data. The decision is made based on the *tag* in the arriving JSON message. The *tag* indicates which type of device sent the low-level data. An example of this low-level data can be seen in Listing 1. The data describes a motion sensor reading with a precision of 85%.

Listing 1: Raw data fragment in JSON format.

```
{
 "prefixes": {
        "ssn": "http://purl.oclc.org/NET/ssnx/ssn"
 },
 "userID": "00001",
 "data": {
                 "n": "motion_sensor",
                 "v": "0.85f",
                 "tag": "MotionSensor"
        }
}
```

3.4.3 Context Adapters

When a *Context Adapter* is added to the platform, it provides the types of sensors, i.e., *tags* of low-level data, it is able to annotate semantically. Each *Context Adapter* can annotate a specific kind of received data to the semantic model. The result of the annotation phase is semantic annotated data, i.e., ontological individuals. Since each *Context Adapter* indicates the type of data it is able to annotate itself,

additional adapters can easily be added to cope with new types of raw data, such as additional sensors. The *Context Adapters* enrich the data to a set of *OWLAxioms*, which are pushed on the SCB. Each *Context Adapter* provides a mapping, describing how the raw data translates to the semantic data. Listing 2 shows an extract of the created OWLAxioms in the annotation phase for the fragment in Listing 1.

Listing 2: Enriched data as OWLAxioms

```
ClassAssertion(pre:Event pre:event_1),
ClassAssertion(ssn:Observation ssn:observation_1),
ClassAssertion(pre:MotionSensor pre:motionSensor_1),
ClassAssertion(pre:MotionOutput pre:motionOutput_1),
ObjectPropertyAssertion(pre:hasContext pre:event_1 pre:observation_1),
ObjectPropertyAssertion(ssn:observedBy pre:observation_1 pre:motionSensor_1),
ObjectPropertyAssertion(ssn:observation_result pre:observation_1 pre:
    motionOutput_1),
DataPropertyAssertion(ssn:hasValue pre:motionOutput_1 "0.85f"^^xsd:float)
```

The fragment illustrates various assertions: ClassAssertions, ObjectPropertyAssertions and a DataPropertyAssertion. For example, the first ClassAssertion states that the event_1 individual is a member of the class Event. It indicates that an *Event* has been created that is linked to an *Observation*. The *Observation* states the kind of sensor that made the observation and the output of the observation, combined with the actual measured value.

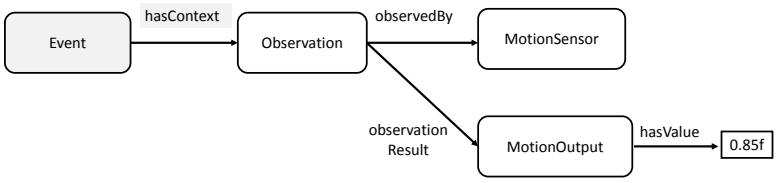

Fig. 3: Conceptual representation of the events in the platform.

All semantic data passing through the platform are *Event*s. Figure 3 shows an example of how the *Event* is linked to the data. From now on, all data flowing through the platform will be called events. Each event has a starting point of the type *Event* in the graph-like data structure. With the relation *hasContext* it is linked to the actual data. Note that the *Event* concept and the *hasContext* relation are pictured in grey, since they are required by the platform. All other types and relations, such as the *Observation* and the *MotionSensor* are use case specific and are not obliged by the platform.

Virtual *Context Adapters* are a special type of *Context Adapters* that do not receive data from the *Matching Service*, and thus do not register a *tag*. These adapters annotate data they capture from external sources, such as Facebook and Twitter streams.

3.4.4 Semantic Communication Bus

The SCB supports communication and collaboration between the different components. The different components publish their data on the SCB in the form of

OWLAxioms. Each component can subscribe to the SCB by passing a filter rule in the form of an OWL class expression. The class describes the kind of data the component wants to consume, on a semantic level. Upon subscription, the registered classes are added to the ontology model of the SCB. Note that the SCB loads its ontology, describing its domain, at startup. When data gets published on the SCB, the published data is temporarily added to the ontology model. The SCB's ontology model now contains the loaded concepts from startup, the registered filter rules as OWL class expressions and the temporary data as OWLAxioms. Through the use of semantic reasoning on the ontology, the type of the published event is retrieved, which matches the subscribed filter rules of those services that would like to consume the data. This way, high-level data coordination is achieved. When the data is forwarded to the selected services, it is removed from the ontology.

In axiom (1), an example filter rule is depicted. It shows that the *MotionFilter* is an *Event* with a relation *hasContext* to an *Observation* and that the *Observation* should have a relation *observedBy* with a *MotionSensor*.

$$MotionFilter \equiv Event \\ \wedge \exists hasContext.(Observation \wedge (\exists observedBy.MotionSensor)) \quad (1)$$

A direct match can be seen between the Event in Figure 3 and the filter. When the reasoner in the SCB is asked for the type of the event, it will also return the *MotionFilter*.

Assume that the following classes are also present in the ontology:

$$MotionSensor \sqsubseteq Sensor \quad (2)$$

$$MotionObservation \sqsubseteq Observation \wedge \exists observation\_result.MotionOutput \quad (3)$$

The class definition in (2) defines the *MotionSensor* as a subclass of *Sensor*. To subscribe to all *Sensors*, including the *MotionSensor*, one can easily subscribe the filter rule in (4).

$$MotionFilter_2 \equiv Event \\ \wedge \exists hasContext.(Observation \wedge (\exists observedBy.Sensor)) \quad (4)$$

To show the added value of the reasoning in the SCB, an additional filter is added in (5).

$$MotionFilter_3 \equiv Event \wedge \exists hasContext.MotionObservation \quad (5)$$

This rule makes use of the axiom in (3). Even though the *MotionObservation* is not explicitly defined in the received data, the reasoner knows what a *MotionObservation* is and will return the filter rule as the type of the event. This enables collaboration between services based on high-level concepts.

The *SCB* enables intelligent collaboration and distribution of retrieved knowledge between *Services*.

3.4.5 Services

Each *Service* subscribes to the *SCB* through the use of one or more filter rules. After processing the consumed data, inferred knowledge can be published to the *SCB*, to notify other *Services* about its findings. Each *Service* contains its own ontology and reasoner. The use of filter rules limits the data each *Service* receives, resulting in more efficient reasoning, since each *Service* only needs to incorporate a subset of data. Note that reasoning might become slow when the size of the dataset increases.

Lets assume a new *Service*, the *MotionService*, which loads profile information in its ontology at startup and subscribes to all motion data with axiom (5). Compared to the event data, big datasets are typically loaded directly into the ontology model of the *Services*, through the use of tools such as Ontop[3] and D2R[4]. The loading of such static background data allows to combine the low-level event data with background knowledge. This combination facilitates the extraction of high-level knowledge. At run time, the sensor data from the motion sensor is combined with the profile data to check the activity of recovering patients. Note that the time period a patient needs to be active is person-dependent. When aberrant activity has been detected, a *Task* is generated indicating that someone should check on that person.

A second *Service* captures all *Tasks* and tries to assign the most suited person to perform these *Tasks*. The *Service* could subscribe to all tasks with the following filter rule:

$$TaskFilter \equiv Event \wedge \exists hasContext.Task \quad (6)$$

Additional *Services* can easily be added to provide extra functionality. Let us assume that the lifetime of certain sensors is limited and when a sensor fails, it starts to produce random values. A *Service* can be added that captures all sensor values and analyses them to see if they start to produce aberrant values. It can then choose to share this knowledge with the other *Services* over the SCB.

When ontologies have been constructed with modularity in mind, each *Service* can load only the necessary parts of the whole ontology, again improving performance. A modular ontology is an ontology that consists of multiple stand-alone ontology modules which improves reasoning efficiency [20]. When a part of the ontology holds a specific profile [38] (e.g., OWL 2 EL, OWL 2 QL, OWL 2 RL) instead of OWL 2 DL, different reasoners can be utilized in each service to optimize performance. Even when no specific profile can be used or no modularity can be detected, different reasoners or techniques can still be used on the whole ontology in each *Service* to optimize the performance of the task at hand.

Each *Service* can share its inferred knowledge through the SCB, which might be of interest to other *Services* which can also process the data and share its knowledge. Combining and passing the results of each *Service* allows the creation of complex reasoning chains. Since each *Service* only defines its data of interest and shares its conclusions, dynamic workflows can be created without the need

[3] http://ontop.inf.unibz.it/
[4] http://d2rq.org/

to predefine a static workflow. A special *Service*, the *Notification Service*, gathers all final knowledge and sends it to all interested parties outside the platform, through the *Gateway*. Each service can decide if the data is ready and label it as final knowledge. The flexibility of the platform allows multiple implementations for the *Service* components. Multiple types of reasoners can be used and other techniques such as data mining and machine learning can easily be applied within these *Services* to process and retrieve knowledge.

3.5 Policy Management

This section elaborates on how inconsistencies are handled in the platform. Since the platform allows users to define their own mapping, describing how raw data should be semantically annotated in the *Context Adapter*s and which data their *Service*s should consume through registering OWL class expressions, inconsistencies can occur. The following scenarios can occur:

1. A *Service* subscribes with an OWL class expression which makes the SCB's ontology inconsistent. To resolve this, the SCB will check at the time of subscription whether the registered OWL class expression is consistent with the remainder of the ontology. If this is not the case, the subscription of the expression is deemed unsuccessful and it is not added to the ontology. A warning is sent to the service provider that the subscription has failed.
2. A *Context Adapter* or *Service* publishes data on the SCB and when reasoned upon in the SCB, a realization inconsistency occurs. This inconsistency can have two causes:

   (a) A *Context Adapter* or *Service* failed to format the semantic data correctly. When the malformed data is reasoned upon in the SCB, a realization inconsistency occurs. For example, the types of an individual have been modeled as two classes that are in fact disjoint with each other.
   (b) A *Service* has recently added an OWL class expression that does not make the ontology inconsistent upon consistency checking, but does causes problems upon realization. For example, a Service can subscribe a new class expression that is disjoint with a previously subscribed filter rule.

   When these types of inconsistencies occur, the cause of the problem is first determined. To achieve this, all class expressions that have been registered by the Services are removed and a realization consistency check is performed on the SCB's core ontology and the published data. When this causes inconsistencies, this means that we are dealing with inconsistencies of type 2.a. The platform can then track which component published the data, deactivate it and send a warning.
   If this does not cause any problems, this means that we are dealing with inconsistencies of type 2.b. The platform needs to trace the Service that subscribed the OWL class expression that is causing problems. Therefore, the subscribed class expressions are added one by one to the SCB's core ontology and for each addition a consistency check is performed on the published data. The expression that causes the inconsistency is removed from the ontology and a warning is sent to the subscriber.

3.6 Supporting Components

The platform provides multiple additional services such as logging, back-up of the knowledge in the *Services* and visualization of the workload.

3.6.1 Journaling

Tailored logging is provided in the core of the platform to track the data flow between the different components. Each component logs its output data, if any, and its destination in the platform. The *Services* typically log their inferred knowledge they would like to share and indicate the SCB as the destination. After determining which Services are interested in the data, the SCB will log to which Services the data is sent. Logging is important to provide accountability. It allows to justify the choices made at a given time.

3.6.2 Backup

Since robustness and resilience is an important aspect in the IoT, a specialized backup system is provided to minimize the data loss upon failure. All messages between components are logged using the journaling and each service makes a backup of its current knowledge base at discrete intervals to further minimize data loss.

3.6.3 Cached SCB

The SCB is the central communication link between different components and can thus easily become a bottleneck. The performance of the SCB is dependent on the used ontology, because reasoning is used to determine the services that are interested in the arriving data. To optimize the performance of the SCB, intelligent caching is introduced to match similar events without the need to reason.

Since the platform is closed, all data traveling through the SCB has been produced in the platform, it can be assumed that each *Context Adapter* or *Service* can only produce a finite number of conceptually distinct messages. Note that only the difference in structure of these messages on a TBox level is considered, not the specific ABox initializations.

The filter rules in the SCB define the high level structure of the expected data. When a filter is triggered, it is possible to map the specific part of the message, responsible for the match, on the filter rule. If this is done on a high level, the presence of the specific structure can be checked with other messages to decide if there is a match.

To determine the data in the message, responsible for the match, the reasoning needs to be reversed and investigated to see which axioms led the reasoner to infer the data as the selected rule, which is an OWL class. The structure of the responsible data is saved in a cache, enabling a simple look-up the next time a similar message passes by. The cache utilizes the Least Recently Use (LRU) strategy, discarding the least used entries first.

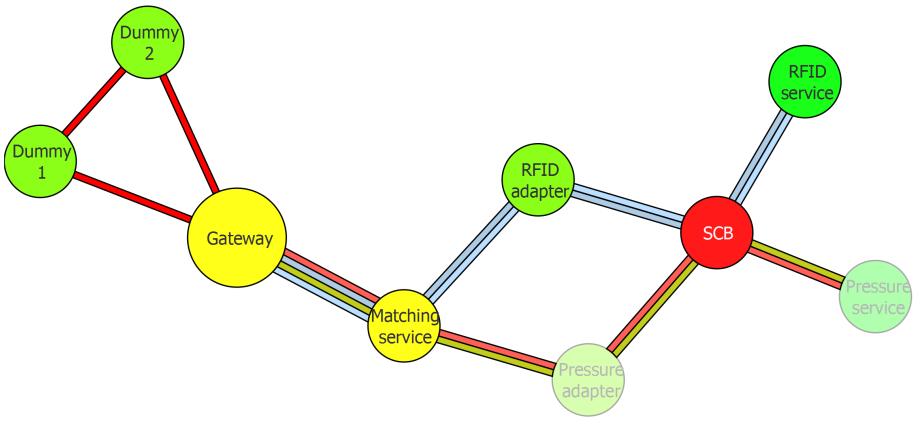

Fig. 4: Visualization of a data flow in the MASSIF platform, visualized as a graph. The vertices represent the components and the edges represent the dataflows between them.

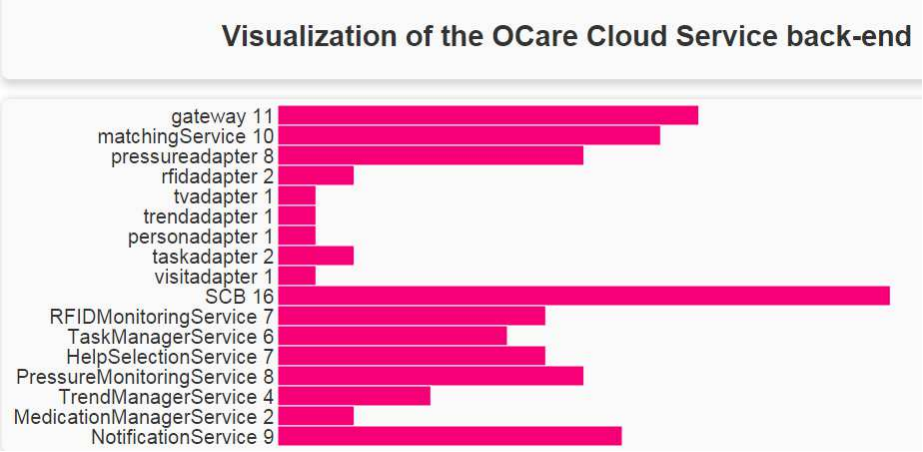

Fig. 5: Visualization of the platform workload. The number of the current processed messages is shown for each component.

3.6.4 Visualization

Visualization tools are available to monitor the data flow through the platform. Keeping a global overview of the data flow when operating in a data-driven, service-oriented environment can become complicated. The use of visual aids simplifies this process.

Figure 4 depicts how the flow through the platform can be monitored on a graphical level. The workflow is visualized as a graph. The vertices represent the services and the edges represent the dataflows between them. Each color represents a different flow of data.

The workload inside the platform is visualized in Figure 5. For each component, the number of messages that are being processed are visualized. This

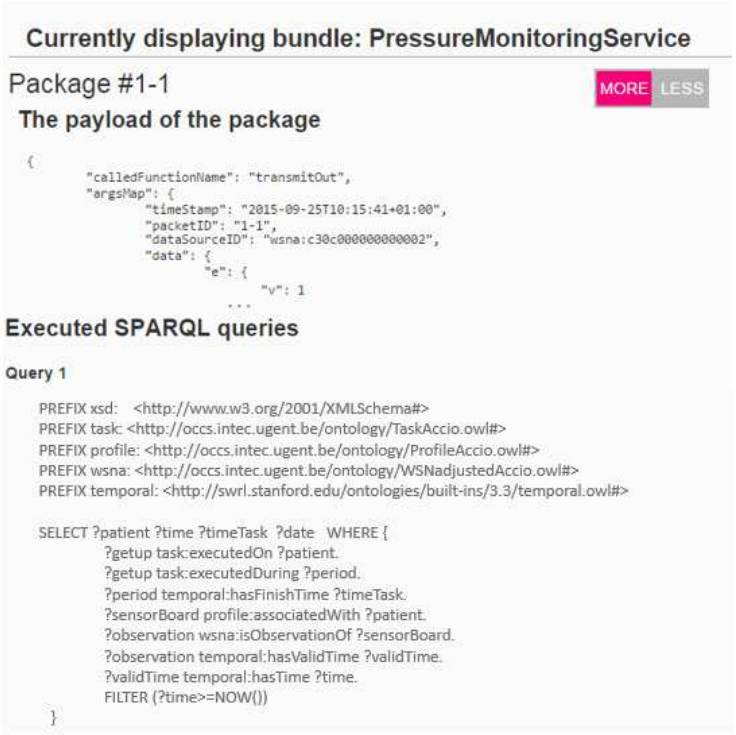

Fig. 6: A detailed inspection of the PressureMonitoringService, showing the initial JSON-message and the executed SPARQL-queries in the *Service*.

provides a visual understanding of the work distribution. Clicking on one of the components enables a deeper inspection. As visualized in Figure 6, the initial JSON-message is shown and the executed SPARQL Protocol and RDF Query Language (SPARQL) [47] query in the selected component. These tools allow a better understanding of the internal flow of messages in the platform.

3.6.5 Message Broker

To enable resilient distributed communication, MASSIF allows the integration of highly efficient message brokers such as RabbitMQ [5] and Kafka [6]. The integration of a message broker allows to communicate with non-semantic services. Communication wrappers have been provided, such that nothing needs to be changed in the implementation of existing *Context Adapters* or *Services*. These wrappers function as an additional layer between the existing components and the message broker and handle all communication. Furthermore, one can choose how to distribute the components over various distributed nodes, e.g., *Services* requiring huge amounts of processing power can be run on separate nodes of a processing cluster. Figure 7 visualizes the message broker integration in MASSIF. As depicted in Figure 7, one

[5] https://www.rabbitmq.com/
[6] http://kafka.apache.org/

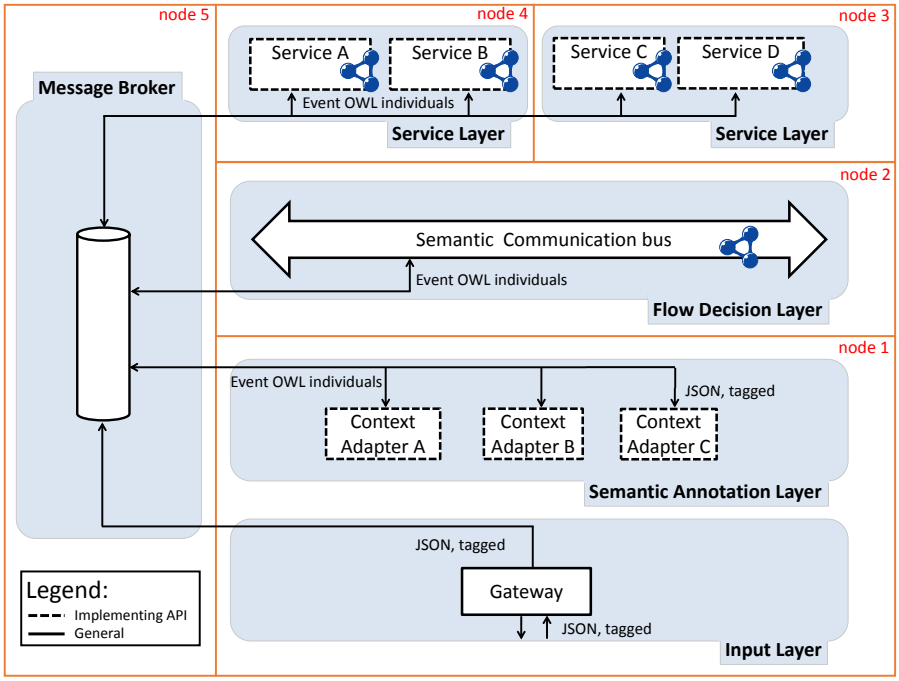

Fig. 7: Architecture of MASSIF with integrated message broker.

can choose to distribute the *Gateway* and *Context Adapters* on one node, since they require low processing, the SCB on another node and to distribute the *Services* over two node since they require most processing resources.

4 Two Use Cases

MASSIF has been evaluated in the Organizing Home Care Using a Cloud-based Platform (OCCS)[7] project and the R.A.M.P.[8] project. R.A.M.P. is short for Real-time Automation of Media Production for interactive radio and conferences. The use cases illustrate the strengths and the performance of the platform in two real-life scenarios. Both cases combine low-level data with background knowledge to extract high-level knowledge.

4.1 eHomeCare

The following sections describe the realization of the OCCS project and give a general overview, a description of the used ontology, an overview of the created *Services* and *Adapters* and finally an evaluation of the created system.

[7] http://www.iminds.be/en/projects/2014/04/07/ocareclouds
[8] http://www.iminds.be/en/projects/2014/03/05/ramp

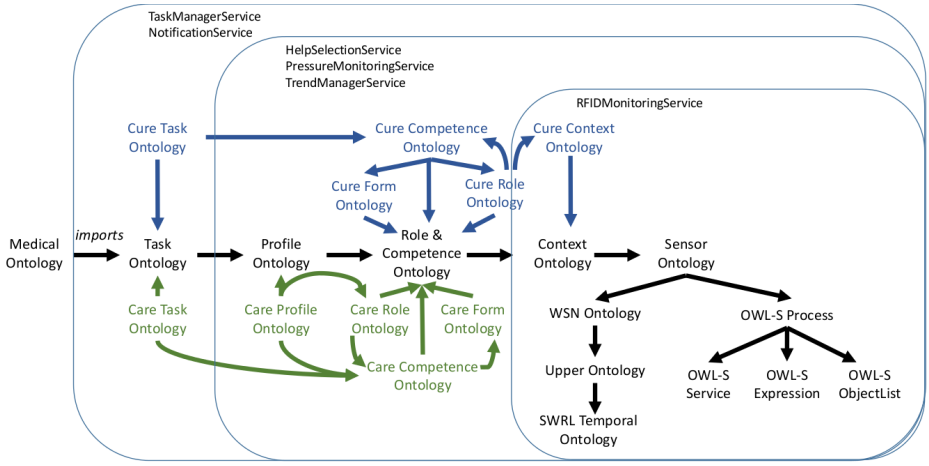

Fig. 8: Import schema of the used ontology.

4.1.1 General Overview

The OCCS project presents a pervasive health use case, demonstrating how healthcare can benefit from the IoT. The OCCS project tackles problems in the home care environment, enabling care organization through cloudy-like services. Hospitals and residential care homes need to cope with the increasing elderly population and the shift from acute to chronic diseases, resulting in a reduced number of available places. An elaborative project description can be found in De Backere et al. [18].

The care receiver's home has been provided with a small amount of discrete sensors and a specialized TV or tablet to interact with. Each caregiver has a smartphone to interact with the platform. By combining the low-level data from the sensors and the smartphones through semantic reasoning, high-level knowledge can be retrieved. From this high-level knowledge, it can, for example, be decided that a care receiver has not been able to get out of bed alone, since the bed pressure sensor is abnormally long active. The system can then decide who might be the designated person to help the care receiver. The selection of the best suited caregiver is based the on the type of relationship the patient has with his helping staff, the competences of the caregivers and their status.

4.1.2 The Ontology

The Ambient-Aware Continuous Care Ontology [40] is used to model all data in the home care environment. Figure 8 shows that it is a modular ontology, containing multiple ontology layers. This allows each distinct *Service* to load only the necessary part of the ontology. An extensive elaboration of the ontology can be found in Ongenae, et al [40].

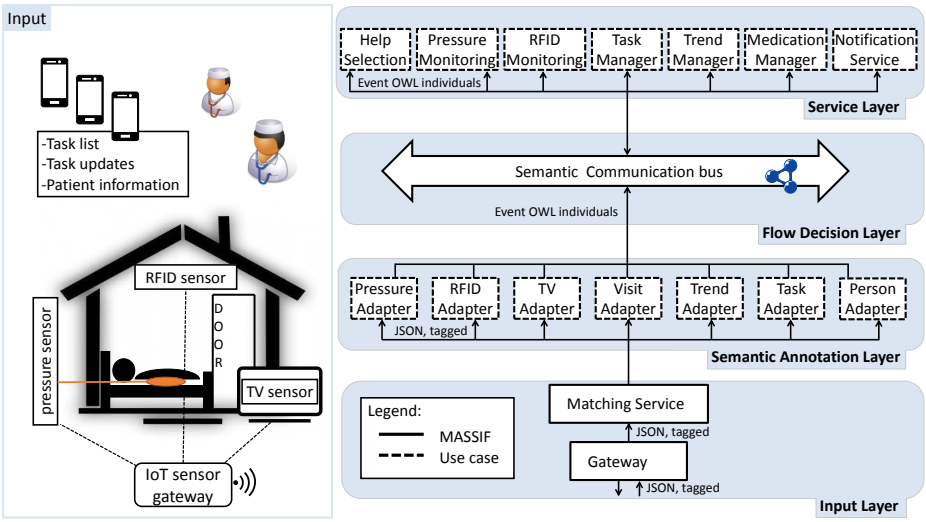

Fig. 9: An visual overview of the designed *Adapters* and *Services* for the OCCS use case. Multiple sensors characterize the environment, enabling context-awareness in the home of the care receiver. These sensors transmit their observation through an IoT sensor gateway to the MASSIF platform.

The ambient-aware continuous care ontology describes the eHealth sector. It has been constructed in collaboration with stakeholders, such as nurses, caregivers and physicians, social scientists and ontology engineers.

- The Upper ontology describes general classes, relations and axioms. The Upper ontology allows data to be related with a unique ID.
- The Sensor & Observation ontology allows the filtering of data. It imports the Wireless Sensor Network (WSN) ontology and extends it with additional eHealth related concepts.
- The Context ontology describes the contextual information regarding the environment. It contains all localization information.
- The Profile ontology models all profile information about the patients and the staff members. Each Person is linked to a Profile, that can either be a basic profile or a risk profile. This has been described as axioms, allowing the risk patients to be automatically inferred.
- The Role & Competence ontology describes roles and competences in the eHealth sector. Roles, based on competences, can be automatically inferred through the use of axioms.
- The Task ontology models the task and call handling.
- The Medical ontology models medical concepts such as observed parameters and pathologies.

Table 2 summarizes the metrics of the proposed ontology.

Table 2: Ontology metrics for the used ontology in the eHomeCare use case.

#Axioms	3065
#Logical Axioms	1736
#Individuals	147
#Classes	409
#Object Properties	185
#Data Properties	53
DL Expressivity	SHOIQ(D)

4.1.3 Designed Adapters

Multiple *Context Adapters* have been implemented, to semantically annotate the raw data. There are seven *Adapters* present in the OCCS project. Figure 9 shows the created *Adapters* and *Services*.

- A *PressureAdapter*, enriches all data transmitted by a pressure sensor.
- A *RFIDAdapter*, translates all received Radio Frequency Identification (RFID)s from caregivers logging in, indicating their presence in the home of the care receiver.
- A *TVAdapter*, annotates all data sent by the TV, capturing the activity of the patient, such as volume and channel changes.
- The *TaskAdapter* handles all data regarding tasks: newly created tasks, task updates, finished tasks. Some examples of possible tasks are: doing groceries, helping the care receiver out of bed, cooking and cleaning.
- The *VisitAdapter* handles all data regarding planned visits: new visits, updated visits and canceled visits.
- The *PersonAdapter* enriches all data describing relations between caregivers and care receivers. For example, when a new caregiver will aid a care receiver, the caregiver is first added to the trust circle of the care receiver. The trust circle indicates which persons the care receiver is familiar with.
- The *TrendAdapter* shows how the data-driven platform can handle specific requests. Trend information about the activity can be requested, based on multiple sensors. The *TrendAdapter* only enriches the request data. The processing of the trend data is done in the *TrendManagerService*. The trend information can give an indication of how active a care receiver is. Some care receivers should do at least some movement during the day.

4.1.4 Specific OCCS Services

Multiple *Services* have been implemented, these are thoroughly discussed below. Note that the *Services* load static data from a database at startup. This data contains the profile information about the caregivers and the care receivers, information about the numerous sensors and devices and recurrent tasks. This data is parsed to the semantic model and loaded in the individual ontologies of these *Services*.

- The *RFIDMonitoringService* registers itself to all semantic RFID-data passing over the SCB. The *Service* receives this particular data by registering the filter rule depicted in Listing 3.

Listing 3: Example filter rule for retrieving RFID data.

```
RFIDFilter ≡ Event and (hasContext some
                  (isObservationOf some (hasSensorPart some RFIDSensor)))
```

The service captures the RFID-data and retrieves the person who is linked to the received RFID, which is not available in the raw sensor data. Finally, it sends an event to the SCB, stating that a specific person has logged in at a given location. Other services, e.g., the *TaskManagerService*, might be interested in this kind of information. Listing 4 shows the generated event from the *Service*.

Listing 4: Resulting Event from the RFIDMonitoringService

```
ClassAssertion(upper:Event regEvent)
ClassAssertion(careTask:RegistrationTask regTask)
ClassAssertion(profile:Person regPatient)
ClassAssertion(profile:Person regHelper)
...
ObjectPropertyAssertion(upper:hasContext regEvent regTask)
ObjectPropertyAssertion(task:assignedTo regTask regHelper)
ObjectPropertyAssertion(task:checkedInWith regHelper regPatient)
...
```

The last axiom indicates that the *regHelper*, which resembles the registered caregiver, has checked in with the patient.

- The *TaskManagerService* handles all task information. The initial tasks are loaded from the database into the ontology of the *TaskManagerService*. These tasks cover all activities a caregiver performs to aid the patient. When a location update, originating from the *RFIDMonitoringService*, arrives, all the tasks that this specific person should perform will be calculated and send, through the *NotificationService*, to the device of the caregiver. These tasks are derived based on the profile of the caregiver. New, updated and finished task information will also arrive in this service, to be able to keep an up-to-date overview of the task lists.
- The *HelpSelectionService* receives all activated tasks and determines who is the most suited person to execute these tasks, depending on location, relation with the care receiver and profile. It links the selected person to the task, labels it as final and sends it to the SCB. A thorough discussion of the task assignment can be found in Bonte, et al [11].
- The *PressureMonitoringService* receives data originating from a pressure sensor, located in the bed of a care receiver. The sensor data indicates the activity of the pressure sensor and thus the presence of the patient. If a patient is longer in bed than usual, without being able to get out alone, a task is generated to get this person out of bed. Listing 5 shows an example query that determines if the patient is longer in bed than usual. Note that the time a patient sleeps on average is calculated before and inserted in the query at query time.

Listing 5: Example query

```
PREFIX xsd:    <http://www.w3.org/2001/XMLSchema#>
PREFIX task: <http://occs.intec.ugent.be/ontology/TaskAccio.owl#>
PREFIX profile: <http://occs.intec.ugent.be/ontology/ProfileAccio.owl#>
PREFIX wsna: <http://occs.intec.ugent.be/ontology/WSNadjustedAccio.owl#>
PREFIX temporal: <http://swrl.stanford.edu/ontologies/built-ins/3.3/
     temporal.owl#>

SELECT ?patient ?time ?timeTask  ?date   WHERE {
?getup task:executedOn ?patient.
```

```
?getup task:executedDuring ?period.
?period temporal:hasFinishTime ?timeTask.
?sensorBoard profile:associatedWith ?patient.
?observation wsna:isObservationOf ?sensorBoard.
?observation temporal:hasValidTime ?validTime.
?validTime temporal:hasTime ?time.
FILTER (?time>=AVGSLEEP())}
```

To better involve the patient in the process, the care receiver is asked if help is necessary. The option to stay in bed is still possible. If help is required, the task is updated, picked up by the *HelpSelectionService* and the most suitable person is asked to help the care receiver out of bed.

– The *TrendManagerService* obtains all kinds of sensor data, enabling trend analysis. Compared to the filter in Listing 3, the *TrendManagerService* registers a filter describing the interest in all types of sensor data, as shown in Listing 6.

Listing 6: Example filter rule for retrieving all sensor data.

```
SensorFilter ≡ Event and (hasContext some
                    (isObservationOf some (hasSensorPart some Sensor)))
```

The *Service* stores all sensor data in an external triplestore[9]. This allows the analysis of the activity of the care receiver. The *Service* allows to request a summary of the activity of one or more sensors. The caregivers can interpret the summary to determine how active the patient has been in a given time interval.

– The *MedicationManagerService* will not receive any data, but will inform care receivers when they should take their medication and the specific amount. This information is retrieved from Vitalink[10], which is an online government platform, containing a complete patient information dossier. The *Service* will analyze the dossier and determine which medications should be taken at what intervals. It will send reminders what medication to take at given time intervals to stimulate the patients to take their medication.
– The *NotificationService* captures knowledge from the SCB that is ready to be shared with the outside world. Other services can indicate that their knowledge can be sent to the caregivers by making it an instance of the *Notification* ontology concept. It will analyze the arriving event and make sure the information is sent to the correct device. The *NotificationService* will register to all *Notifications* on the SCB, as depicted in Listing 7.

Listing 7: Example filter rule for retrieving all Notifications.

```
NotificationFilter ≡ Event and (hasContext some Notification)
```

Additional services can be added to the platform, providing the caregivers and care receivers with supplementary information.

4.1.5 Results

To evaluate the performance of the platform in the described use case, the time necessary to complete one scenario is presented. According to Alshareef et al. [5], an eHealth help system should be able to respond within 5 seconds. The scenario

[9] http://stardog.com/
[10] http://www.vitalink.be/

Table 3: An overview off all average processing times for each component in each system call. The averages (μ) and the standard deviation (σ) are both given in milliseconds.

	Pressure Sensor		Help Needed		Task Accepted	
	μ(ms)	σ(ms)	μ(ms)	σ(ms)	μ(ms)	σ(ms)
Gateway	<1	<1	<1	<1	<1	<1
MatchingService	0.62	0.49	0.54	0.50	0.54	0.50
ObservationAdapter	5.77	1.22	-	-	-	-
TaskAdapter	-	-	3.15	0.82	3.31	0.87
SCB-0	0.62	0.49	0.46	0.50	0.62	0.49
SCB-1	0.50	0.50	0.54	0.50	0.54	0.50
SCB-2	0.54	0.50	-	-	-	-
HelpSelectionService	55.50	14.18	75.96	14.23	57.54	11.24
PressureMonitoringService	230.50	23.17	-	-	-	-
TaskManagerService	32.65	15.96	31.50	13.25	31.62	8.96
NotificationService	19.38	2.76	22.81	12.99	19.62	1.62

consists of an automatic trigger from a pressure sensor, informing the system that the patient is still in bed. The system will notice that the patient is longer in bed than usual and will automatically send a notification to the patient. The patient can inform the system if help is needed. This way, the patient is involved in the automated decision process. If the patient decides that help is required, the system will search for the most suited caregivers to aid the patient. They automatically receive a message with the question if they could go and help the patient out of bed. Note that the message automatically disappears if one of the caregivers accepts the task.

The scenario was evaluated 35 times. The first three and the last two results were dropped to eliminate the influence of the warm-up and cooling down period. The averages are calculated over the remaining 30 iterations. The evaluation was done on a Ubuntu 14.04 server with an Intel Xeon CPU E5520 (16 cores) @ 2.27GHz with 12 GB of memory.

As presented in Table 3, it is clear that the platform overhead (*Gateway*, *Adapters*, *SCB*) has limited influence. The table shows multiple *SCB* entries, this is because the *Services* need to exchange their inferred knowledge and thus messages pass the *SCB* multiple times for each call.

Figure 10 visualizes the time spent in the *Services*, grouped for each call. The time spent in the various *Services* differs, this is due to the fact that each *Service* performs a different reasoning task. The *PressureMonitoringService* performs the most complex task in this scenario, as it needs to decide if help should be requested to aid the patient. The *HelpSelectionService* takes longest in the second call, where the system needs to select the best suited caregivers to aid the patient in the current situation.

Figure 11 visualizes the performances of the *Cached SCB* as discussed in Section 3.6.3, compared to the normal SCB. The graph shows the time needed for both buses to execute 15 successive calls. Thus, the x-axis can be seen as a timeline. Note that the warm-up period was not omitted to investigate the performance of the cache over time. At first, the cached SCB is less efficient, because additional reasoning needs to be performed to select the dominant parts of data that should be cached. After a few misses in the cache, only hits occur and the needed time declines to less than 1 millisecond. The normal *SCB* also starts performing better

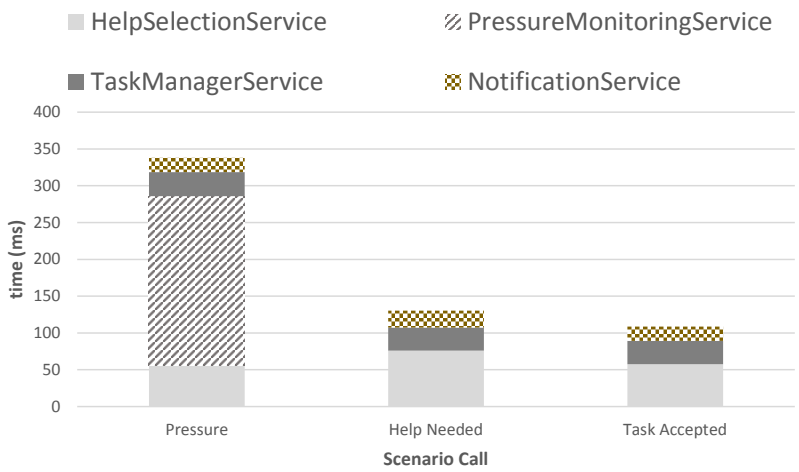

Fig. 10: Evaluation of the designed *Services* in the OCCS use case. The scenario consists of three calls. Pressure: the pressure sensor indicates that the patient is still in bed. If the system sees that the patient is longer in bed than usual, it asks the patient if help is needed. Help Needed: the patient indicates that help is welcome and the system will select the most suited caregivers for this task. Task Accepted: one of the caregivers accepts the task.

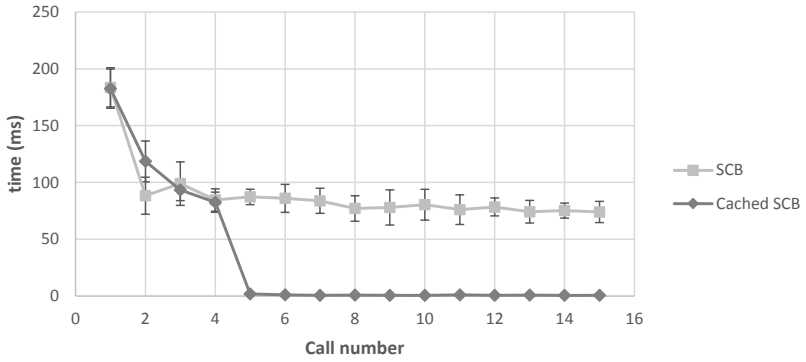

Fig. 11: Evaluation of the *Cached SCB* compared to the normal *SCB*. After a few cache misses, only hits occur and the time spent in the *Cache SCB* gets down to less than one millisecond.

after a few calls, this is due to the performed optimizations inside the reasoner. Eventually the time spent in the *Cached SCB* gets down to less than one millisecond.

4.2 Media

The following sections describe the realization of the media use case and give a general overview, a description of the used ontology, an overview of the created *Services* and *Adapters* and finally an evaluation of the created system.

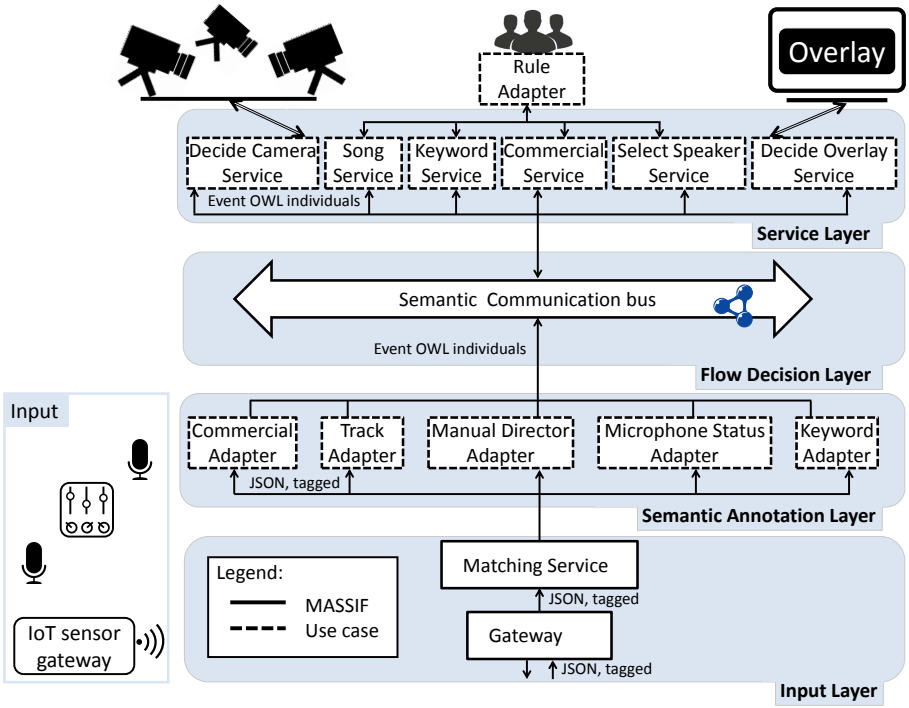

Fig. 12: A visual overview of the designed *Adapters* and *Services* for the R.A.M.P. use case. Multiple sensors characterize the environment, enabling context-awareness in the studio. These sensors transmit their observation through an IoT sensor gateway to the MASSIF platform.

4.2.1 General Overview

The general idea behind the media use case is elaborated below, or more specifically the Real-time Automation of Media Production (R.A.M.P.) project. Both a visual radio and conference use case have been designed, however only the radio case will be elaborated upon. The idea behind both cases is equal: the studio or conference room has been equipped with multiple ubiquitous devices, producing contextual data continuously. Combining these with background knowledge allows to infer high-level knowledge to control various cameras and video overlays. A mapping of the created system to the MASSIF platform is depicted in Figure 12.

4.2.2 Radio

The radio scenario aims at providing visual radio with interactive content, based on the subject of the radio show or the played song in an automated manner, with minimal input from the DJ. The MASSIF platform is the brain of the designed system. It captures and integrates all available data regarding the show and the events inside the studio. The different triggers contain the activity of each microphone, the status of the played songs and commercials, information about certain detected keywords and the configuration of the studio and the cameras.

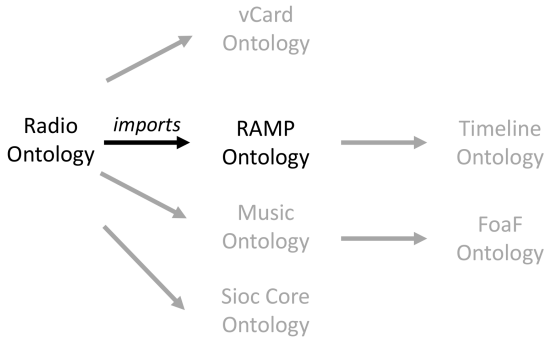

Fig. 13: Import schema of the used ontology. The external imported ontologies are depicted in grey.

These triggers result in the selection of the best suited camera or the activation of certain visual overlays.

4.2.3 The Ontology

The created radio ontology makes use of existing ontologies, as shown in Figure 13. The used ontologies are:

- The Music Ontology[11] describes music related concepts.
- Friend of a Friend[12] (FoaF) defines people-related terms.
- Sioc Core Ontology[13] is an ontology for describing the information in online communities.
- The vCard Ontology[14] describes electronic business cards. They contain names, addresses, phone numbers, email addresses, etc.
- The Time ontology[15] describes the temporal content.

Table 4 summarizes the metrics of the proposed ontology.

Table 4: Ontology metrics for the used ontology in the media use case.

#Axioms	4989
#Logical Axioms	1582
#Individuals	109
#Classes	247
#Object Properties	384
#Data Properties	203
DL Expressivity	SHOIQ(D)

[11] http://musicontology.com/
[12] http://xmlns.com/foaf/spec
[13] rdfs.org/sioc/ns
[14] http://www.w3.org/2006/vcard/ns-2006.html
[15] http://www.w3.org/TR/owl-time

4.2.4 Designed Adapters

To allow extracting useful information from the data originating from multiple sources, the data is first semantically annotated in one of the various created *Context Adapters*.

- The *CommercialAdapter* receives data describing the start and the stop of a commercial and enriches it to the semantic model.
- The *TrackAdapter* annotates data regarding the start and stop of a music track.
- The *MicrophoneStatusAdapter* receives data describing the microphone activity of one of the speakers. This alternates between active and inactive.
- The *KeywordAdapter* receives the detected keywords during the show and enriches the data. The keywords are detected through the use of speech recognition. The detector listens for a list of keywords that have been extracted from the DJ preparation. The DJ prepares a document a few minutes before the start of the show, containing a summary of the various topics handled during the show.
- The *ManualDirectorAdapter* is used for the semantic annotation of the data resulting from the overruling mechanism allowing to show a specific person. This is further explained in Section 4.2.6.

4.2.5 Specific R.A.M.P. Services

The various *Services* react on the received data and decide to manipulate the cameras or visualize additional data if necessary. Each *Service* reasons on the integrated data and generates a *Sequence* of shots that could be shown in the video stream. The creation is based on some precondition, e.g., when the DJ's microphone is active. Semantic Web Rule Language (SWRL)-rules [28] are used to create these *Sequences*. Listing 8 shows this first type of rules in (1). The use of rules allows easy adaptation of the automated process. The rule creates a *Sequence* when the microphone of the DJ is active.

Listing 8: SWRL-rule examples

```
(1) Microphone(?m),capability(?m,DJ),unitState(?m, On)
                                    -> Sequence(Sequence_dj)
(2) Track(?t),capability(?g,MainGuest)
                                    -> member(Sequence_dj,Shot1),show(Shot1,?g)
```

A second type of rule in (2) generates *Shots* to be shown in the *Sequence*. It shows how the *Shot* is added to the created *Sequence*. The *Shot* can show the main guest and can only be added if there is such a guest. The separation of the two types of rules allows multiple combinations in each *Service*, where multiple rules of each type can be active. When an event arrives at one of the *Services*, the reasoner retrieves all instances of the type *Sequence* and *Shot* and their attributes, leaving the creation of the *Sequences* and *Shots* up to the reasoner.

The created *Services* are elaborated below:

- The *Select Speaker Service* gathers all information regarding the microphone activity, the configuration of the room and the information of who is sitting on which seat. Combining the low-level microphone activity with the room configuration allows to determine which exact person is speaking. Note that only the microphone activity arrives as an event at run time, the room configuration

Fig. 14: User interface for the adaptation of the rules by non-technical users.

and the profile information is loaded into the ontology at startup. Based on the defined rules, the outcome of the reasoning task determines who should be selected to show.

- The *Decide Camera Service* captures the possible shots, e.g., from the *Select Speaker Service* and determines what the best suited cameras and camera positions are to show a given person. The service loads the camera configuration at startup, stating the possible presets for each camera. The presets define which seats can be shown with a given certainty and quality. The *Decide Camera Service* will use reasoning to determine the best camera preset for a specific shot in a selected sequence. The sequence gets selected based on its priority. When a sequence of shots arrives with a higher priority, the current sequence will be interrupted and the new sequence will be shown. To provide fluent camera switching towards the viewer, the service will make sure that the same camera with a different preset is never selected sequentially. If this would be the case, the viewer would witness the repositioning of the camera. Therefore, after the selection of a new camera shot, the service will wait until the repositioning has finished before switching shots. Reasoning is performed to enable the camera selection which facilitates fluent camera positioning.
- The *Decide Overlay Service* captures the various activities in the studio and selects the correct overlay based on those activities. For example, different overlays are shown based on the fact that someone is speaking, a keyword has been detected or a song is playing.
- The *Commercial Service* contains all the information regarding the played commercials. It provides rules that can specify how to react on the starting or stopping of a commercial. These rules define who should be shown in case of the described event.
- The *Song Service* captures the information about the played songs. The rules can define how to react when a song is started or stopped.
- The *Keyword Service* is a bit different since it does not allow any manipulation through the use of rules. It receives a spoken keyword as input and will determine which overlay should be shown upon detecting the keyword. This means that the outcome cannot be adapted.

4.2.6 Reasoning Manipulation

To allow control over the automated video composition, a visual *Rule Adapter* is provided which allows end users to adapt the reasoning decisions in the *Services*.

The rules in each *Service* can be adapted to manipulate the automated process. The manipulation of the rules has been abstracted, eliminating the need for the end user to have specific knowledge regarding rules or the ontology. Each *Service* provides high-level generic rules, which can be made more specific. The provided rules are based on the possible events, during the show or conference. As shown in Figure 14, a possible rule might be the fact that a track starts playing and multiple

predefined actions can be chosen for that fact. Listing 9 shows an example of a high-level rule with the possible event as the antecedent in (2) and the predefined actions as consequences in (5) and (7).

Listing 9: High-level rule example

```
(1) {"description": "A track starts playing",
(2)  "antecedent": {
(3)    "rule": ["Track(?t), q:isActive(?t,true) -> Sequence(Sequence_Song)"
    ]},
(4)  "consequences": [
(5)    {"subRule":"Track(?t), capability(?guest, MainGuest)
            -> member(Sequence_Song, Shot1), show(Shot1, ?guest)",
(6)     "description":"Show the main guest"},
(7)    {"subRule":"Track(?t), capability(?dj, DJ)
            -> member(Sequence_Song, Shot2), show(Shot2, ?dj)",
(8)     "description":"Show the DJ"}]}
```

The descriptions in (1), (6) and (8) show how the rules are mapped to readable sentences, alleviating the non-technical user from the technical details.

The antecedent and the consequences contain a (sub)rule, which is a valid SWRL-rule. When the end user selects one (or more) of the predefined consequences, the antecedent rule (3) and the selected subrules (5) or (7) are added to the ontology, allowing the reasoner to incorporate the users preferences.

To provide the viewer a more natural experience, additional properties can be set.

- Priorities: Since multiple rules can be activated at the same time, the video composition can be fine-tuned by specifying which rules have a higher priority than others. For example, the fact that the DJ is speaking might be more important than the fact that a track starts playing.
- Randomness: As depicted in Figure 14, multiple actions (subrules) can be selected for one antecedent. The order in which these actions occur can be specified. However, one can add a randomness factor to mix up the order and provide a more natural flow.
- Timing: The time period each camera shot is selected, can be specified. For example, one could opt to capture the DJ longer than his guests.

The *Decide Camera Service* takes all these options into account when deciding what to show and when to show it. When a high-priority *Sequence* arrives, the current camera shot should be interrupted, even when the specified timing has not been passed.

4.2.7 Reasoning Overruling

A Manual Director tool was designed to explicitly overrule the automatic camera selection. This provides the end users control over the automated process. Through the use of the Manual Director tool, one can manually select the seat that should be captured on camera. This overrules the selection made by the automated reasoning process. A special *Context Adapter* can enrich this data and directly create a shot sequence of one shot with the highest possible priority. This shot sequence will be captured by the *Decide Camera Service*. This *Service* will show the desired seat directly since this sequence has the highest possible priority. In the manual director, it is possible to define how many seconds the manual director should

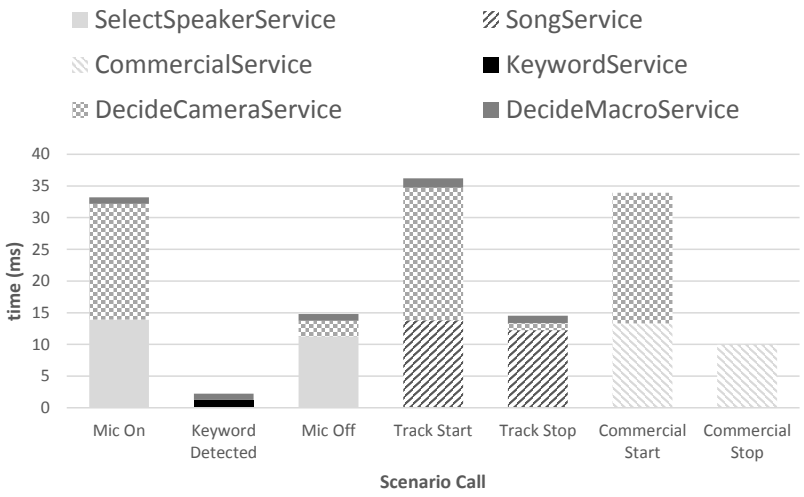

Fig. 15: Evaluation results of the designed *Services* for the R.A.M.P. use case. The scenario consist of seven calls, which are explained in the beginning of this Section.

overrule the system, which corresponds to the time period a shot should be shown in a shot sequence. After the defined period of time, the automatic selection of shots continues.

4.2.8 Results

To evaluate the performance of the platform in the described use case, the time the system needs to coordinate a typical radio show was evaluated. Since this coordination should be visually appealing, the system should react within 100 milliseconds. According to Card et al. [14], a delay of 100 milliseconds is not troublesome for the human perception. The show consists of the following events:

1. The DJ turns on his microphone and talks for 5 seconds.
2. A special keyword is detected.
3. The DJ turns off his microphone.
4. A new track starts playing
5. The track stops playing.
6. A commercial starts playing.
7. The commercial stops.

The scenario was evaluated 35 times. The first three and last two results were dropped to eliminate the influence of the warm-up and cooling down period. The averages are calculated over the remaining 30 iterations. The evaluation was done on a Ubuntu 14.04 server with an Intel Xeon CPU E5520 (16 cores) @ 2.27GHz with 12 GB of memory.

Table 5 presents the overall average times for all components in each call. It is clear that the *Services* have the greatest impact on the system, since these com-

Table 5: An overview off all average processing times for each component in each system call. The averages (μ) and the standard deviation (σ) are both given in milliseconds.

	Mic On		Keyword Detected		Mic Off		Track Start		
	μ(ms)	σ(ms)	μ(ms)	σ(ms)	μ(ms)	σ(ms)	μ(ms)	σ(ms)	
Gateway	<1	<1	<1	<1	<1	<1	<1	<1	...
MatchingService	0.31	0.46	0.38	1.00	0.23	0.42	0.38	0.49	...
MicrophoneAdapter	1.85	0.82	-	-	1.81	0.83	-	-	...
KeywordAdapter	-	-	1.58	0.57	-	-	-	-	...
TrackAdapter	-	-	-	-	-	-	1.50	0.57	...
CommercialAdapter	-	-	-	-	-	-	-	-	...
SCB-0	0.46	0.50	0.42	0.49	0.42	0.49	0.50	0.50	...
SCB-1	0.58	0.49	-	-	0.54	0.50	0.69	0.46	...
SelectSpeakerService	13.92	7.52	-	-	11.27	2.30	-	-	...
SongService	-	-	-	-	-	-	13.81	2.67	...
CommercialService	-	-	-	-	-	-	-	-	...
KeywordService	-	-	1.23	0.50	-	-	-	-	...
DecideCameraService	18.27	3.61	-	-	2.46	0.63	20.92	8.13	...
DecideOverlayService	1.00	<1	1.02	<1	1.08	0.47	1.46	0.50	...

	Track Stop		Commercial Start		Commercial Stop	
	μ(ms)	σ(ms)	μ(ms)	σ(ms)	μ(ms)	σ(ms)
...	<1	<1	<1	<1	<1	<1
...	0.31	0.46	0.19	0.39	0.23	0.42
...	-	-	-	-	-	-
...	-	-	-	-	-	-
...	1.65	1.00	-	-	-	-
...	-	-	1.50	0.57	1.88	0.97
...	0.69	0.72	0.42	0.49	0.42	0.57
...	-	-	0.81	0.39	-	-
...	-	-	-	-	-	-
...	12.23	2.55	-	-	-	-
...	-	-	13.31	3.11	9.92	1.47
...	-	-	-	-	-	-
...	1.12	0.51	20.62	5.37	-	-
...	1.19	0.62	-	-	-	-

ponents perform reasoning.

Figure 15 visualizes the average time spent in each *Service* in each step of the scenario. The other components have negligible impact and have thus been omitted. It shows that the *Decide Camera Service* needs about 20 milliseconds when the microphone gets turned on, a track starts or when a commercial starts. However, when the microphone is turned off, only a small fraction of time is spent in the *Decide Camera Service*. This is due to the fact that the system is configured (through the use of the rules) to show an overview camera shot when nobody is speaking. The overview shots do not change and are cached for performance measures. In the other scenario steps (Keyword detected, Track stops and Commercial stops), there is no camera activity, because these *Services* have been configured not to manipulate the cameras as a results of its incoming triggers. Note that this can be easily changed by updating the rules. It is notable that the *Decide Overlay Service* is quite efficient. This is due to the fact that most of the overlay extraction has been done as a preprocessing step. During the show this is only a simple look-up.

Furthermore, it is clear that the system is efficient and causes a maximum delay of less than 35 milliseconds. This is more than acceptable, since a delay of less than 100 milliseconds is not troublesome for the human perception [14]. It is important for visual radio that delays are minimized as much as possible to provide a fluent flow to the end user. The high performance results in a scalable platform that enables the possibility for multiple concurrent scenarios.

5 Discussion

We have presented the MASSIF platform, a platform that can successfully enable dynamic and high-level coordination between IoT services. Through two use cases, we have shown that the platform can efficiently handle generated IoT data and also allows to perform complex reasoning. However, there are some known limitations that will be addressed in our future work.

MASSIF is an event-based system, in the sense that it can perform advanced reasoning on event data. Processing of continuous flows of streaming data is currently not the focus of MASSIF. Examples of such streaming data are Facebook and Twitter streams or current measurements of streaming sensors. Each of these streams would be annotated by its own Context Adapter. Currently a single adapter can annotate more than 100 messages per second, as presented in Table 3 and 5. However, more than 500 messages per second would flood the Context Adapter. Since each component can run on a separate node of a processing cluster, the rest of the system might not suffer from this congestion. Once the data is annotated, the Services need to process the data, these can again congest if the arrival rate is higher than the processing rate. Similarly, if the MASSIF platform is deployed in a distributed fashion, the rest of the platform will not be hindered when one service congests, at least if they are not dependent on the outcome of the congested service. One way to mediate this congestion is to subscribe the filter rules, which indicate the data the services will consume, more intelligently. One could opt to filter most of the data in the SCB, limiting the data arrival rate in the services. The uptake of stream reasoning is a high priority in our future work. However, current stream reasoning systems do not support complex reasoning.

A second limitation is the use of the tag in the low-level sensor data. In the presented use cases, the sensor integration is supported by the DYAMAND platform [39], which was developed at the IBCN[16] research group. The DYAMAND platform allows the detection and integration of sensors and devices. It utilizes a model to make a distinction between sensors. The tag is a result of this model. Similar functionality could be offered by mapping the internal model used by other sensor gateways on the tags. In the future, we wish to offer an API to sensor (gateway) developers that allows them to request the available tags in the system. These tags would be accompanied by a human-readable description. This would allow these developers to choose the appropriate tags for the data they send to the platform. Offering such an API enables the integration of data into the platform that has not been semantically annotated in an easy and straightforward way. When the data does not have a tag, it is filtered by the gateway and thus not processed by the MASSIF Platform. In the future, we wish to make the Matching Service more

[16] https://www.ibcn.intec.ugent.be/

intelligent by incorporating machine learning and text analysis algorithms that allow to automatically process the incoming data and choose the most appropriate Context Adapter. As such, the platform would be able to handle data that is not tagged.

6 Conclusions & Future Work

The number of connected devices will know a rapid increase due to the rising popularity of the IoT. The need to capture, transform and process the produced data by these devices grows. Moreover, the number of services processing the produced data will also increase.

In this paper, the MASSIF platform is presented. It allows semantic annotation of IoT data and the high-level coordination between semantic IoT-services. The platform is fully data-driven and by representing the data semantically, *Services* can indicate their input data on an abstract level. Each *Service* can process the received data and share its gained knowledge with other *Services* through the use of the Semantic Communication Bus. This allows the creation of data-driven workflows that can fulfill complex reasoning chains. By defining their input data, *Services* operate on a subset of the available data, achieving more efficient reasoning. The applicability of the platform has been shown by presenting two concrete use cases: an eHomeCare case and a media case. The platform has also been thoroughly evaluated by means of the same two use cases. These use cases demonstrate the performance of the platform. Furthermore, they indicate that the platform can be extended to cope with additional data producers and new *Services* to provide extra processing capabilities.

In our future work, stream reasoning techniques will be incorporated for the efficient processing of data streams for less complex reasoning scenarios. To be able to annotate unknown sensor data, machine learning techniques will be investigated enabling the platform to learn how to annotate unknown data. Furthermore, load balancing techniques and automated duplication of the *Services* will be investigated to provide a truly scalable system.

Acknowledgment

OCCS was partly funded by iMinds and the IWT project nr. 110113. It involves Boone, Familiehulp Gent, Telecom-IT, Televic Healthcare and TP Vision.

R.A.M.P. was partly funded by iMinds and involves Media Innovatie Centrum, Small Town Heroes, Televic Conference and Vlaamse Media Maatschappij.

References

1. Abowd, G.D., Dey, A.K., Brown, P.J., Davies, N., Smith, M., Steggles, P.: Towards a Better Understanding of Context and Context-Awareness. In: Handheld and ubiquitous computing, pp. 304–307. Springer (1999)
2. Al-Fuqaha, A., Guizani, M., Mohammadi, M., Aledhari, M., Ayyash, M.: Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications. Communications Surveys & Tutorials, IEEE **17**(4), 2347–2376 (2015)

3. Al-Jadir, L., Parent, C., Spaccapietra, S.: Reasoning with large ontologies stored in relational databases: The OntoMinD approach. Data & Knowledge Engineering **69**, 1158–1180 (2010)
4. Ali, M.I., Ono, N., Kaysar, M., Griffin, K., Mileo, A.: A Semantic Processing Framework for IoT-Enabled Communication Systems. In: The Semantic Web-ISWC 2015, pp. 241–258. Springer (2015)
5. Alshareef, H., Grigoras, D.: First responder help facilitated by the mobile cloud. In: Cloud Technologies and Applications (CloudTech), 2015 International Conference on, pp. 1–8. IEEE (2015)
6. Atzori, L., Iera, A., Morabito, G.: The Internet of Things: A Survey. Comput. Netw. **54**, 2787–2805 (2010)
7. Baralis, E., Cagliero, L., Cerquitelli, T., Garza, P., Marchetti, M.: CAS-Mine: providing personalized services in context-aware applications by means of generalized rules. KAIS **28**(2), 283–310 (2011)
8. Barnaghi, P., Wang, W., Henson, C., Taylor, K.: Semantics for the Internet of Things: Early Progress and Back to the Future. Int. J. Semant. Web Inf. Syst. **8**, 1–21 (2012)
9. Bechhofer, S., van Harmelen, F., Hendler, J., Horrocks, I., McGuinness, D.L., Patel-Schneider, P.F., Stein, L.A.: OWL Web Ontology Language Reference. Tech. rep., W3C, http://www.w3.org/TR/owl-ref/ (2004)
10. Bergamaschi, S., Castano, S., Vincini, M., Beneventano, D.: Semantic Integration of Heterogeneous Information Sources Using a Knowledge-Based System. Data & Knowledge Engineering **36**, 215–249 (2001)
11. Bonte, P., Ongenae, F., Schaballie, J., De Meester, B., Arndt, D., Dereuddre, W., Bhatti, J., Verstichel, S., Verborgh, R., Van de Walle, R., et al.: Evaluation and Optimized Usage of OWL 2 Reasoners in an Event-based eHealth Context. In: OWL Reasoner Evaluation Workshop, vol. 4. CEUR (2015)
12. Byun, H.E., Cheverst, K.: Utilizing Context History To Provide Dynamic Adaptations. Appl. Artif. Intell. **18**, 533–548 (2004)
13. Calbimonte, J.P., Sarni, S., Eberle, J., Aberer, K.: XGSN: An Open-source Semantic Sensing. Middleware for the Web of Things. . Terra Cognita and Semantic Sensor Networks p. 51 (2014)
14. Card, S.K., Robertson, G.G., Mackinlay, J.D.: The Information Visualizer, an Information Workspace. In: Proceedings of the SIGCHI Conference on Human factors in computing systems, pp. 181–186. ACM (1991)
15. Compton, M., Barnaghi, P., Bermudez, L., GarcíA-Castro, R., Corcho, O., Cox, S., Graybeal, J., Hauswirth, M., Henson, C., Herzog, A., et al.: The SSN Ontology of the W3C Semantic Sensor Network Incubator Group. Web Semantics: Science, Services and Agents on the World Wide Web **17**, 25–32 (2012)
16. Crockford, D.: The Application/JSON Media Type For Javascript Object Notation (JSON). Internet informational RFC 4627 (2006)
17. De, S., Elsaleh, T., Barnaghi, P., Meissner, S.: An Internet of Things Platform for Real-World and Digital Objects. Scalable Computing: Practice and Experience **13**(1), 45–58 (2012)
18. De Backere, F., Ongenae, F., Vannieuwenborg, F., Ooteghem, J.V., Duysburgh, P., Jansen, A., Hoebeke, J., Wuyts, K., Rossey, J., Van den Abeele, F., et al.: The OCareCloudS project: Toward organizing care through trusted cloud services. Informatics for Health and Social Care (0), 1–19 (2014)
19. Dey, A.K., Abowd, G.D., Salber, D.: A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-Aware Applications. Comput.-Hum. Interact. **16**, 97–166 (2001)
20. Ensan, F., Du, W.: A knowledge encapsulation approach to ontology modularization. KAIS **26**(2), 249–283 (2011)
21. Famaey, J., Latré, S., Strassner, J., De Turck, F.: An Ontology-Driven Semantic Bus for Autonomic Communication Elements. In: Lecture Notes in Comput. Sci., vol. 6473, pp. 37–50. Springer Verlag Berlin (2010)
22. Forkan, A., Khalil, I., Tari, Z.: CoCaMAAL: A cloud-oriented context-aware middleware in ambient assisted living. Future Generation Computer Systems **35**, 114–127 (2014)
23. Gray, A.J., García-Castro, R., Kyzirakos, K., Karpathiotakis, M., Calbimonte, J.P., Page, K., Sadler, J., Frazer, A., Galpin, I., Fernandes, A.A., et al.: A Semantically Enabled Service Architecture for Mashups over Streaming and Stored Data. In: The Semanic Web: Research and Applications, pp. 300–314. Springer (2011)

24. Gruber, T.R.: A Translation Approach to Portable Ontology Specifications. Knowl. Acquis. **5**, 199–220 (1993)
25. Haarslev, V., Hidde, K., Möller, R., Wessel, M.: The RacerPro Knowledge Representation and Reasoning System. Semantic Web **3**, 267–277 (2012)
26. Hogan, A., Harth, A., Polleres, A.: Saor: Authoritative reasoning for the web. In: The Semantic Web, pp. 76–90. Springer Berlin Heidelberg (2008)
27. Horridge, M., Bechhofer, S.: The OWL API: A Java API For OWL Ontologies. Semantic Web **2**, 11–21 (2011)
28. Horrocks, I., Patel-Schneider, P.F., Boley, H., Tabet, S., Grosof, B., Dean, M.: SWRL: A Semantic Web Rule Language Combining OWL and RuleML. Tech. rep., World Wide Web Consortium (2004)
29. Huertas Celdran, A., Clemente, G., Felix, J., Gil Perez, M., Martinez Perez, G.: SeCoMan: A semantic-aware policy framework for developing privacy-preserving and context-aware smart applications (2013)
30. Hustadt, U., Motik, B., Sattler, U.: Data Complexity of Reasoning in Very Expressive Description Logics. IJCAI International Joint Conference on Artificial Intelligence pp. 466–471 (2005)
31. Indra: IoT Interoperability Platform with a Big Data approach (2016). URL `sofia2.com`
32. Kang, J., Park, S.: Context-Aware Services Framework Based on Semantic Web Services for Automatic Discovery and Integration of Context. International Journal of Advancements in Computing Technology **5**(4) (2013)
33. Kostelnik, P., Sarnovsk, M., Furdik, K.: The semantic middleware for networked embedded systems applied in the Internet of Things and Services domain. Scalable Computing: Practice and Experience **12**(3), 307–316 (2011)
34. Li, X., Eckert, M., Martinez, J.F., Rubio, G.: Context Aware Middleware Architectures: Survey and Challenges. Sensors **15**(8), 20,570–20,607 (2015)
35. McGuinness, D.L., Van Harmelen, F., et al.: OWL Web Ontology Language Overview. W3C recommendation **10**, 2004 (2004)
36. Mineraud, J., Mazhelis, O., Su, X., Tarkoma, S.: A gap analysis of Internet-of-Things platforms. arXiv pp. 1–7 (2015)
37. Motik, B., Grau, B.C., Horrocks, I., Wu, Z., Fokoue, A., Lutz, C.: OWL 2 Web Ontology Language Profiles. W3C recommendation **27**, 61 (2009)
38. Motik, B., Grau, B.C., Horrocks, I., Wu, Z., Fokoue, A., Lutz, C.: OWL 2 Web Ontology Language: Profiles. W3C recommendation **27**, 61 (2009)
39. Nelis, J., Verschueren, T., Verslype, D., Develder, C.: DYAMAND: DYnamic, Adaptive MAnagement of Networks and Devices. In: Local Computer Networks (LCN), 2012 IEEE 37th Conference on, pp. 192–195. IEEE (2012)
40. Ongenae, F., Bleumers, L., Sulmon, N., Verstraete, M., van Gils, M., Jacobs, A., Zutter, S.D., Verhoeve, P., Ackaert, A., Turck, F.D.: Participatory Design of a Continuous Care Ontology - Towards a User-driven Ontology Engineering Methodology. In: KEOD, pp. 81–90. SciTePress (2011)
41. OSGi Alliance: OSGi Service Platform Release 4. http://www.osgi.org/. Accessed 9 September 2015 (2009)
42. Palmisano, I.: JFact DL Reasoner. http://jfact.sourceforge.net/. Accessed 1 July 2015 (2014)
43. Palmisano, I.: Reasoners, OWL API Support, papers about the OWL API. https://github.com/owlcs/owlapi/wiki/Reasoners,-OWL-API-Support,-papers-about-the-OWL-API. Accessed 23 April 2015 (2014)
44. Patkos, T., Chrysakis, I., Bikakis, A., Plexousakis, D., Antoniou, G.: A Reasoning Framework for Ambient Intelligence. In: Artificial Intelligence: Theories, Models and Applications, pp. 213–222. Springer (2010)
45. Perera, C., Zaslavsky, A., Christen, P., Georgakopoulos, D.: Context Aware Computing for The Internet of Things: A Survey. IEEE Commun Surv. Tut. **16**, 414–454 (2014)
46. Pinto, H.S., Martins, J.P.: Ontologies: How can They be Built? KAIS **6**(4), 441–464 (2004)
47. PrudHommeaux, E., Seaborne, A., et al.: SPARQL Query Language for RDF. W3C recommendation **15** (2008)
48. Shearer, R., Motik, B., Horrocks, I.: HermiT: A Highly-Efficient OWL Reasoner. In: OWLED, vol. 432 (2008)
49. Simperl, E.: Reusing ontologies on the Semantic Web: A feasibility study. Data & Knowledge Engineering **68**, 905–925 (2009)

50. Sirin, E., Parsia, B., Grau, B.C., Kalyanpur, A., Katz, Y.: Pellet: A practical OWL-DL reasoner. Web Semantics: Science, Services and Agents on the World Wide Web **5**, 51–53 (2007)

51. Soldatos, J., Kefalakis, N., Hauswirth, M., Serrano, M., Calbimonte, J.P., Riahi, M., Aberer, K., Jayaraman, P.P., Zaslavsky, A., Žarko, I.P., et al.: OpenIoT: Open Source Internet-of-Things in the Cloud. In: Interoperability and Open-Source Solutions for the Internet of Things, pp. 13–25. Springer (2015)

52. Strang, T., Linnhoff-Popien, C.: A Context Modeling Survey. In: Workshop Proceedings (2004)

53. Tsarkov, D., Horrocks, I.: FaCT++ Description Logic Reasoner: System Description. In: International Joint Conference on Automated Reasoning, pp. 292–297. Springer-Verlag, Berlin, Heidelberg (2006)

54. Tsarkov, D., Palmisano, I.: Chainsaw: a Metareasoner for Large Ontologies. In: ORE, vol. 858. CEUR Workshop Proceedings (2012)

55. Wang, F., Hu, L., Zhou, J., Zhao, K.: A Survey from the Perspective of Evolutionary Process in the Internet of Things. International Journal of Distributed Sensor Networks **2015**, 9 (2015)

Author Biographies

Pieter Bonte graduated from Ghent University, Faculty of Engineering and Architecture in the summer of 2013. In September, he joined the Internet Based Communication Networks and Services (IBCN) research group of Piet Demeester as a Research Engineer and started working on the research of knowledge discovery and management for IoT services, using semantic technologies. In September 2014 he started working as a PhD student in the same department focusing on scalable reasoning solutions for IoT applications. During his research, he has participated in several interdisciplinary research projects, providing personal and optimized care for patients both in home as hospital settings.

Femke Ongenae graduated from Ghent University, Faculty of Engineering in Summer 2007. A month later she joined the research group of Piet Demeester, IBCN as a PhD student. On the 1st of January 2009, she received a PhD grant from the IWT, Institute for the Support of Innovation through Science and Technology, to work on the research of knowledge discovery and management for eHealth applications using ontologies. During this time, she worked on several eCare projects to improve the continuous care of patients in institutionalized care settings. She received her PhD in August 2013, and she is currently working as a postdoctoral researcher at the IBCN research group. She does research in the area of knowledge management and discovery, specifically focusing on the use of semantic web technologies and ontologies for the optimization of continuous care and the application of IoT.

Femke De Backere graduated from Ghent University, Faculty of Engineering and Architecture in the summer of 2009. In August of that year, she joined the Internet Based Communication Networks and Services (IBCN) research group of Piet Demeester as a PhD student and started working on the research of knowledge discovery and management for pervasive eHealth and eCare services, using semantic technologies. During her research, she has participated in several interdisciplinary research projects focusing on facilitating independent living at home for elderly and individuals with chronic diseases. She obtained her PhD in June 2016 and will continue working on these topics, while also researching how semantic technologies can be used in institutional care.

Jeroen Schaballie graduated from KaHo Sint-Lieven high school, specialization Electronics-ICT in summer 2010. After working two years in a private company as a Software Engineer, he joined the research group of Piet Demeester, IBCN as a Research Engineer and is associated with several projects assisting in network based projects, web service applications and eCare projects. During the last two years, he was focusing on how to improve the continuous care of patients in institutionalized care settings and is currently working on the use of semantic web technologies and ontologies for the optimization of continuous care and the application of IoT.

Dörthe Arndt studied Mathematics and Computational Linguistics at the University of Bonn and Polytechnic University of Catalonia in Barcelona. She graduated in 2010. For her Master thesis in the field of mathematical logic she co-developed the Naproche proof checker. After that she worked for 2 years as a risk manager in an international insurance group. Since May 2013, she is a researcher at Data Science Lab Ghent University iMinds. Her main research interests are semantic web logics, reasoning and and rule languages, in particular Notation3 Logic. Her current work mainly focuses on the formal semantics of Notation3 and its relation to other logics.

Stijn Verstichel graduated magna cum laude at Ghent University in 2005. He joined the research group of Piet Demeester, IBCN working on two European Projects. Geant2, consists of the development and deployment of a new-generation pan-European Research Network, and corresponding monitoring software. InteGRail focuses on the development of specific semantic software for the railway industry. Its aim is to create a holistic, intelligent and integrated information and data sharing platform for the European Railway Network. In 2007, he received a PhD grant to work on research in distributed reasoning techniques for the Semantic Web. He successfully defended his PhD, titled “Distributed Reasoning for Context-Aware Services” in 2011. He is currently still with IBCN-iMinds as a post-doctoral researcher, performing research on semantics through a number of iMinds projects. He is author or co-author of more than 30 papers and is a regular reviewer for conferences and journals in his research field.

Erik Mannens is Professor @ Ghent University - Data Science Lab / Member of the Senior Management Team @ iMinds / Research Manager @ iMinds - Data Science Lab since 2005 where he has successfully managed +50 projects. He received his PhD degree in Computer Science Engineering (2011) at UGent, his Masters degree in Computer Science (1995) at K.U. Leuven University, and his Masters degree in Electro-Mechanical Engineering (1992) at KAHO Ghent. He heads the Data Science team of +50 Semantic Technologies & Artificial Intelligence Researchers. The primary objective of our Data Science Lab team is to advance research and technology in the sweet spot of the fusion of Semantics & Artificial Intelligence and to widely apply this research in large-scale use cases. His major expertise is centered around metadata modeling, semantic web technologies, big data analytics, broadcasting workflows, iDTV and web development in general.

Rik Van de Walle received master and PhD degrees in Engineering from Ghent University, Belgium in July 1994 and February 1998, respectively. His PhD was about Magnetic Resonance Imaging, and was prepared in the context of a close collaboration between the Department of Radiology and the Department of Electronics and Information Systems. After a post-doctoral fellowship at the University of Arizona he returned to Ghent, became a full-time Lecturer in 2001, and founded the Multimedia Lab at the Faculty of Engineering and Architecture. This research group is one of the founding teams of iMinds. In 2016, it became Data Science Lab. Currently, the Data Science Lab has about 80 members. In 2004 he was appointed Full Professor, and in 2010 he became Senior Full Professor at Ghent University. His personal research interests include multimedia content delivery, coding of multimedia data, metadata technology, content adaptation, interactive (mobile) multimedia applications, and e-health, systems and signals.

Filip De Turck leads the network and service management research group at the Department of Information Technology of the Ghent University, Belgium and iMinds (Interdisciplinary Research Institute in Flanders). He (co-) authored over 450 peer reviewed papers and his research interests include telecommunication network and service management, efficient big data processing and design of large-scale Internet of Things systems. In this research area, he is involved in several research projects with industry and academia, serves as vice-chair of the IEEE Technical Committee on Network Operations and Management (CNOM), chair of the Future Internet Cluster of the European Commission, and is on the TPC of many network and service management conferences and workshops and serves in the editorial board of several network and service management journals.