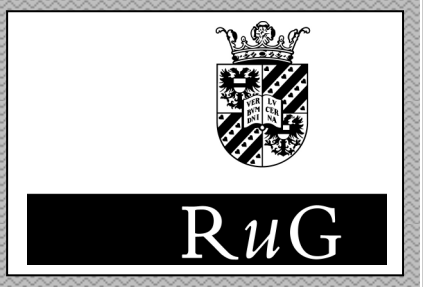
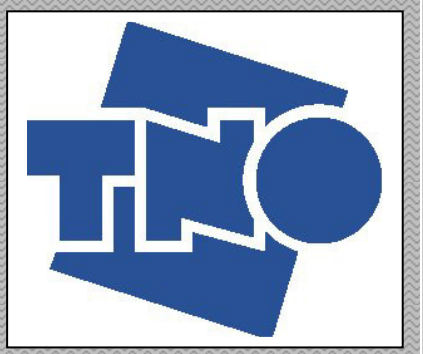
2007

Distributed Semantic Sensor Networks

How to use semantics and knowledge distribution to integrate sensor data of disparate data sources.

In this research project we focused on sensor networks and how semantic web technologies and knowledge sharing can make integration of distributed sensor data possible. To achieve this we have first identified the main challenges to address to allow for data integration. A distributed semantic sensor network framework is proposed that uses semantic web techniques and knowledge sharing to address these challenges. We propose that knowledge sharing in a sensor network is a key aspect of data integration in a dynamic environment, since it allows the network to handle changes in the environment. This project is innovative in that it proposes new ways to handle semantic integration in a distributed environment, by using question federation and data conversion on semantically annotated data and questions. It contributes to current research in that our framework enables data integration of distributed sensor data.





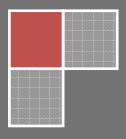
Masters Student

M.J. van der Veen, *Department of Artificial Intelligence, Rijksuniversiteit Groningen, Groningen*

Supervisors

Bart Verheij, *Department of Artificial Intelligence, Rijksuniversiteit Groningen*

Arnoud de Jong, *TNO ICT & Telecommunicatie, Groningen*



People

Masters Student:
Maarten van der Veen

A masters student at the Department of Artificial Intelligence at the Rijksuniversiteit Groningen in The Netherlands. This thesis is the final work in his masters studies. His main interests besides writing this thesis is doing sports, coaching, travelling and making music.

Supervisor: Bart Verheij

A tenured lecturer/researcher (in Dutch: universitair docent) at the University of Groningen, Department of Artificial Intelligence and a member of the ALICE institute. He participates in the Multi-agent systems research program. His research areas are artificial intelligence, logic and law, with emphasis on defeasible argumentation and legal reasoning. A recently added direction of research is agent-based social simulation.

Supervisor: Arnoud de Jong

An innovator at TNO ICT & Telecommunications. He specializes in location based services, semantics and data visualization.

Table of Contents

1 Introduction

Take a look at our world from an information system perspective and you will see a lot of information that once only existed in our heads, but has now been made explicit in digital environments. With the arrival of the world wide web [1], we started to link different information sources on a wide scale through a text interface. With the upcoming of the Semantic Web [2], the possibilities of linking data on the web will be taken to a higher level. Semantics allow machines to interpret data and to connect different data instances by adding semantic metadata to a data instance. For example, one could specify properties of objects, to enable machines to compare objects based on these properties. We can describe an apple as an instance which has a round shape, a colour and which is edible.

An interesting thought is whether it is possible to link our physical world much in the same way as the information sources on the world wide web are linked. Linking sensors in a network results in a sensing web, which has many new capabilities compared to the normal web. An analogue is the addition of sensors to a normal computer. The result is a robot (without actuators), which is aware of its environment and has many more capabilities than a normal computer.

To achieve this, we need to be able to measure the physical world and make these measurements accessible. Physical phenomena can be measured by using sensors. Imagine a network of these sensors. Such a network should allow us to pose questions about the real world phenomena which are measured by the individual sensors. To give a simple example of the possibilities that arise, consider an office building with in every room a temperature sensor. By creating a network of sensors, we could ask the question: “In what rooms is the temperature over 25 degrees Celsius?”. Or maybe: “alert me when the average temperature in the office building gets below 10 degrees Celsius”. More complex questions can be thought of when there are sensors of different types available, for example, pressure sensors and noise sensors. This is interesting because combining measurements of different sensors gives us insight into the interaction between the phenomena in the environment, measured by these sensors.

Since sensor networks are situated in the real world, one should think about how to handle continuous operation in a dynamic environment. Environmental events that change the meaning of sensor measurements but also human intervention in the network are reasons why a network of sensors could become erroneous in the future, making it unusable.

By introducing knowledge sharing in the sensor network domain new possibilities arise to handle continuous operation in a dynamic environment. Knowledge sharing is already a research topic in Peer-to-Peer networks [3, 4] and in multi-agent systems [5] (See Section 3.3 on knowledge sharing). In the sensor network domain knowledge sharing enables sensors to exchange knowledge about sensor measurements and their properties, about changes in the environment or human inflicted changes. Exchanging knowledge among sensors results in an increase of shared knowledge about the environment in which these sensors are situated. This makes it possible to use the sensor network over a longer period of time in a dynamic environment.

The aim of this project is to enable integration of distributed sensor measurements in a sensor network and to allow for dynamic changes in shared knowledge, to allow the network to be useable in the dynamic environment in which the sensors are situated. This results in the following research questions:

- How can semantic web technologies be used in the domain of sensor networks to enable the integration of distributed sensor measurements?
- How does knowledge sharing in a sensor network enable the network to handle both changes in the dynamic environment in which it is situated and human induced changes to the sensors in the network.

1.1 Problem Relevance

Recent technical advancements in creating small, energy efficient sensors have increased the number of applicable areas in which sensor networks can be deployed. Earlier work has made great

advancements in energy consumption in wireless networks [6, 7] and routing efficiency in sensor networks [8-12]. Currently, deployed sensor networks take a centralized approach in which all measurements are routed to a central storage. Such an approach is useful for small homogeneous networks, but with the possibility of linking a large number of heterogeneous sensors in a network, there is a need for a more distributed approach. As a result there is a growing need to add meaning to sensor data, to enable integration of sensor measurements at the source: the sensor.

Different research groups have focused on integration of sensor measurements in distributed sensor networks [13-21]. Some of them have adopted a semantic approach in which semantic metadata is used to add meaning to raw sensor measurements [11, 12, 15, 17]. This allows for the comparison and integration of distributed sensor data. The construction of a distributed sensor network in which semantic metadata is used to integrate sensor measurements is still an ongoing research. Data integration problems can be addressed by looking into the possibilities of semantic web technologies in the sensor network domain.

Moreover, we believe that simply adding meaning to sensor measurements is not sufficient for a sensor network to be useable in an ever changing environment in which the sensors are situated. As a simple example, by changing the location of a sensor, its measurements get a different meaning. We illustrate the relevance of this project with a real world scenario: the IJkdijk.

1.1.1 IJkdijk

The IJkdijk is a collaborative research project between a number of parties, among whom TNO ICT & Telecommunication in Groningen and the Dutch Government. These parties are interested in changes in the condition of embankments in The Netherlands and Germany as a result of weather, temperature, embankment construction and other factors. To enable examination of the condition of the embankment, a number of sensors are placed in the embankment.

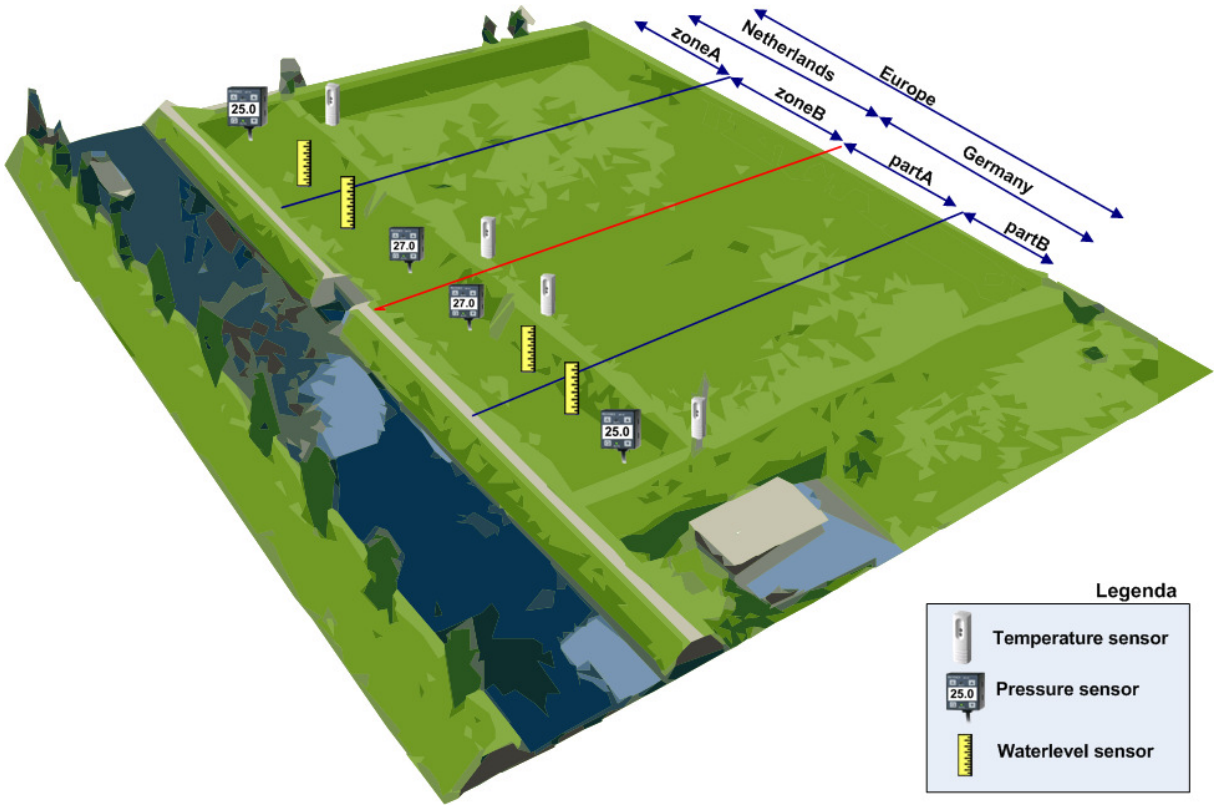


Figure 1: IJkdijk

These sensors are connected in a network, for users to be able to access the sensors, question their measurements and combine sensor readings from different sensors. Figure 1 gives an idea of the

environment around the embankment and a number of sensors that can be placed along the embankment.

Along the embankment are a number of temperature, pressure and water level sensors. Each of these measures a value. Each measurement has a different meaning. For example, a measurement of a temperature sensor is a temperature measurement. Such a measurement could be a degrees Celsius measurement or a degrees Fahrenheit measurement. Adding metadata such as sensor type and the unit of a measurement to the meaning of the sensor increases the knowledge about the sensor. There is a need for a machine interpretable language to describe types, properties and other characteristics such as unit and the relation between them (such as the relation between degrees Fahrenheit and degrees Celsius). This enables sensors and users in the network to exchange knowledge about the environment that is being measured.

The temperature, pressure and water level sensors along the IJkdijk are deployed and maintained by different parties. Among them are a Dutch company and a German company, who both have added temperature and pressure sensors to the network. A European company has added water level sensors to the network. Each company describes its sensor measurements in its own machine interpretable language. As a metaphor, we could say that the Dutch company sensors describe their measurements in Dutch and are able to answer questions in Dutch about these measurements. The same goes for the German and European sensors, but they only "speak" and "understand" German and English respectively. There is a need for translations between these languages to enable the integration of the sensor measurements from these different companies.

Each sensor in the network can do a huge amount of sensor measurements. Having only displayed a few sensors in this example, in fact there could be an unlimited amount of sensors linked in a sensor network. To handle large amounts of sensor measurements it is desired that the sensor data remains distributed, close to the sensor. Adopting a distributed approach eliminates the need to send all the sensor data to a central storage point. The amount of transfer of sensor data in the network is therefore considerably lower. The distributed approach also has the advantages of spreading computation in the network and allowing different parties to add sensors to the network and maintain these sensors by their own standards. A result of distribution however is that a question about sensor measurements can not be answered by a single 'central' information source in the network. The network thus answers a question, by sending the question, or sub-questions, to the sensors individually.

The environment of the IJkdijk is anything but stable. The embankment changes shape continuously as an effect of the water flowing past it. Flooding of the embankment can result in dislocation of sensors and also human intervention can change the position of each sensor. Such events can affect the meaning of a sensor measurement. Also, sensors can be added to or removed from the network. These sensors could measure different phenomena than existing sensors in the network, thus affecting the possible questions that can be answered by the network. Having a fixed meaning for each sensor, a fixed number of sensors or a predefined set of sensor types is thus not sufficient to make the sensor network useable in a changing environment. There is a need for a way to share the knowledge of each sensor, so that sensors can incorporate changes in its environment and also human inflicted changes, such as the addition or removal of sensors to the network.

Typical questions a user would want to ask to the network of sensors include:

- What is the temperature measured by sensor X?
- What are the temperatures measured at the embankment?
- Where along the embankment is the pressure the highest?
- How stable is the embankment?
- What is the water level at locations where the temperature is over 14?
- Warn me when a location on the embankment is about to flood.

Each sensor can perform knowledge acquisition by actively consulting other sensors about measurements in their environment. Sensor to sensor questions could include:

- Give me the water level when it reaches 100 centimetres.

- Always give me the latest temperature measurement at location X.

Besides active knowledge acquisition other types of knowledge sharing are the exchange of processing knowledge, such as knowledge on how to calculate the stability of an embankment, mapping knowledge, which is a translation from one machine interpretable sensor language to another, or data conversion knowledge, such as knowledge on how to convert from a degrees Celsius measurement to a degrees Fahrenheit measurement.

By reasoning over shared knowledge each sensor should be able to infer new knowledge of the environment in which it is situated. It can use this knowledge to keep up to date with changes in the environment and answer user questions about this new knowledge.

1.2 Goal

This project is inspired by the latest research on integration of distributed sensor measurements and the practical problems that have arisen in the IJkdijk project. This project is a design-science research project [22], meaning that we address an unsolved problem in an innovative way. The goal of this project is to enable the integration of distributed sensor data. We will do so by adding semantics to the sensor data and to allow for knowledge sharing about these semantics and the environment. We propose a framework which we will call a **distributed semantic sensor network**. With such a framework it becomes possible for a sensor network to answer questions about distributed sensor measurements, by integrating the sensor data of distributed sensors. Moreover, the framework allows for the **sharing of knowledge** between sensors in the network, allowing them to operate in a dynamic environment.

The IJkdijk scenario will be used throughout this report. However, the aim of the proposed framework is to be applicable in a much wider context than only the IJkdijk scenario. It should be usable in other types of sensor network environments as well and as we will see in the discussion section, it can even be used outside of the domain of sensor networks.

The following paragraph presents a number of challenges that have to be dealt with when trying to create a distributed semantic sensor network as described in the previous paragraph.

1.3 Challenges

In the sensor network domain there are a number of challenges to address. Some of these are of a technical nature, such as energy usage in sensor networks or data communication for data exchange between sensor nodes. Another challenge is reliability of information in sensor networks; when is sensor data outdated and how do we know that the sensor measurements are correct? These challenges apply to the IJkdijk scenario, but are not addressed in this report, since solutions are extensively discussed in earlier work [6-12].

We have identified four challenges from earlier research and the IJkdijk scenario. Each of these challenges should be addressed to design a distributed semantic sensor network with knowledge sharing capabilities. We will now discuss each of the four domain specific challenges. Later we will explain how we are going to address these challenges.

The first challenge was identified by Yao and Gehrke in their survey on sensor networks [23]. The challenge is the construction of a **distributed network architecture** and how to make each sensor data source accessible through the network. This challenge follows from the distribution requirement of the IJkdijk scenario.

A second challenge when working with sensors in a distributed environment is how to **facilitate the integration of sensor data**. Integration of sensor data of two sensors can be accomplished if the smallest known facts: data instances are known by both sensors. The instances should be comparable and used to integrate different properties of a data instance. We know this as the data integration challenge [24, 25], also known as data fusion and more specifically as sensor fusion in the area of Sensor Networks [26, 27]. Data integration is the combining of data of disparate sources in such a way that the resulting information is more completely described, more accurate or has a different meaning altogether.

An important aspect of data integration is adding semantics to data, to enable comparison of data instances [2]. Even if data instances can be integrated by using semantics it is still possible that

one data instance is stored in a different form than another instance. This makes the integration of the two instances faulty. A sub challenge to facilitate the integration of sensor data is to enable data conversion in the network. Data conversion is the process of converting one instance to another, allowing to fuse the converted instances [28, 29]. For example, in the sensor domain one can think of converting a temperature reading in degrees Celsius to degrees Fahrenheit. After conversion, data instances with the same meaning and with the same form or unit can be combined.

A third challenge is to allow the user of a sensor network to pose questions about distributed sensor data in the network as we have seen in the IJkdijk scenario. The fact that the data is distributed makes the combining of sensor data non-trivial. This is a challenge since a single question has to be split into multiple sub-questions which each address a different data source. Also the answers to the sub-questions have to be combined to make data integration possible. The process of splitting a question and combining the answers is what we call **question federation**. As an example, consider a simple sensor network with temperature sensors and pressure sensors at different locations. If we want to know the pressure at locations where the temperature is over 25 degrees Celsius, then we have to split this question into a sub-question to temperature sensors about the location of sensors with a measurement over 25 degrees Celsius and a sub-question to pressure sensors at these locations to get the pressure .

A fourth and final challenge is to enable the **sharing of and reasoning with semantic knowledge** in the network. Examples of semantic knowledge are data pre-processing knowledge, data type conversion knowledge or shared knowledge. By allowing the network to share new ways of combining sensor data and conversions it becomes possible to increase the capabilities of the network and reason with this knowledge to infer new knowledge from the data. This makes it possible to maintain and possibly increase the usability of the network over a long period of time in a dynamic environment. Knowledge sharing also makes it easier to update embedded sensors that are not easily accessible. To summarize, the four main challenges for building a distributed semantic sensor network are:

1. Construction of a distributed network architecture; how to facilitate communication among sensor nodes in a distributed environment
2. Facilitate the integration of sensor data; how to enable data integration by adding semantic web technologies to the sensor network domain and improve upon existing integration solutions
3. Question federation; how to split and federate a semantic question to enable the integration of distributed sensor data
4. Sharing of and reasoning with semantic knowledge; how can knowledge sharing allow a sensor network to operate in a dynamic environment and how reasoning with shared knowledge can increase knowledge at sensor nodes

1.4 Related Work

Different research groups have proposed data integration solutions for sensor networks. We group these solutions by: database oriented sensor networks, agent based sensor networks, and semantic sensor networks. A couple of the most interesting and promising frameworks for sensor networks are discussed.

Database oriented solutions treat a Sensor Network as a database. Among these are TAG and TinyDB [13, 20] and Cougar [30]. They propose a query language that can query distributed sensor data with a single question. These solutions typically use question flooding to send a question to each and every sensor node in the network and a routing tree to send the answer back to the user. These solutions require a fixed data structure to store sensor measurements at sensor nodes and therefore have to be designed for a specific domain, making it difficult to apply the framework to other domains.

Other frameworks treat a Sensor Network as a distributed system in which they use the multi agent paradigm to exchange data between different sensors in the network. The agents use protocols to exchange raw sensor data, which have then to be interpreted internally by the agent. These frameworks allow for discovery of new agents or sensors in the network, making the network more dynamic. Agent frameworks allow agents to reason about the phenomena that are being measured by

the sensors. Agent based frameworks include IrisNet [19], AIGA [14], the Biswas and Phoha architecture [18] and the SWAP framework [15].

A number of frameworks embrace the use of semantics as a solution to data integration. The use of semantics enables these frameworks to be more flexible in changing and extending sensor network environments. Dimitrov et. al [17] propose a distributed information integration system based on semantic web technologies. The framework can send a semantic user question to distributed databases which describe their content in the semantic web description format: RDF. The framework requires the databases to provide a mapping between the databases to allow for the rewriting (or translation) of the user question, so each database can be addressed.

Currently, the work of Dimitrov et. al can be seen as state of the art. This architecture makes it possible to pose questions to distributed data sources. However improvements can be made upon the assumption that each data source stores the same information. Each of the discussed frameworks has its strengths and weaknesses. None of these have incorporated knowledge sharing to handle changes in a dynamic environment. In the discussion section we will compare our distributed semantic sensor network framework with these frameworks to show how we have improved upon these related works.

1.5 Research Method

None of the before mentioned frameworks meet all four challenges discussed in Section 1.3. In our view, all the main components for a proper sensor network framework have been introduced in earlier work. We believe the use of semantics plays a major role in the challenge of data integration. We have used the latest semantic web techniques to formulate semantic questions and annotate sensor data. Moreover, we incorporate our own data conversion solution into semantic questions to improve upon existing data integration capabilities of the semantic query language. To enable fully flexible question federation we propose a federation method based on question splitting. We embrace the multi agent paradigm for our distributed sensor network architecture, both because it is an intuitive approach to distributed systems and because it incorporates useful protocols for knowledge sharing. We will show how knowledge sharing enables the sensors in the network to operate in a dynamic environment and adjust their knowledge given environmental and user induced changed, while remaining in operation.

We have looked in detail at existing semantic web technologies and design decisions in the fields of sensor networks, semantic web technologies and multi agent systems. This knowledge was used to design a framework with which a user is able to integrate sensor data by asking semantic questions to a network of distributed sensors. The design is inspired by the IJkdijk scenario from which we have identified the requirements that constitute the design of our distributed semantic sensor network.

An implementation of the framework is evaluated in a case study of the IJkdijk scenario. To do this we discuss a number of typical questions for the IJkdijk scenario and show how the semantic question is federated and the answer is returned to the user. Moreover, we show how the answers to questions and the capabilities of each sensor change when we share different types of knowledge with the network sensors. No formal evaluation of the framework is possible, since there are no comparable frameworks that incorporate just as flexible data integration and knowledge sharing capabilities.

This research project makes three contributions to the domain of sensor networks. First, a scientific contribution is given with our work on knowledge sharing. We show how knowledge sharing makes it possible to increase the capabilities of the sensor nodes individually and the answering capabilities of the sensor network as a whole. Second, our proposed question federation algorithm makes it possible to ask questions to distributed heterogeneous data sources and our work on data type conversion enables the integration of sensor data in the network by the sensors themselves. Third, a practical contribution is given by providing a framework in the form of an implementation of a distributed semantic sensor network for the IJkdijk scenario, with which actual experiments on a real world sensor network have been performed.

1.6 Thesis Outline

Section 2 gives the reader some background knowledge in the areas of Sensor Networks (Section 2.1), Semantics (Section 2.2) and Multi-agent Systems (2.3). Section 3 explains three ways to improve upon

existing data integration solutions. A query federation method is proposed to handle the splitting and distribution of a question to multiple sensor nodes (Section 3.1). A solution for data conversion is presented in Section 3.2. We give an overview of the types of knowledge that can be shared in a sensor network domain, to enable the sensor network to operate in a dynamic environment (Section 3.3). We propose a framework that addresses the challenges for a distributed semantic sensor network. The framework is discussed methodologically in Section 4, by first analyzing the requirements for a distributed semantic sensor network and based on that we present the design of the framework. In Section 5, after presenting the framework we show how it can be used to implement the IJkdijk scenario and we evaluate the framework against the requirements in a case study with a simulation of the IJkdijk. Finally, in the discussion, Section 6 we discuss the weaknesses and strengths of our framework and how it compares to related work. In the conclusion, Section 7, we review whether we have been able to meet the challenges posed and we suggest directions for future research.

2 Research Background

In this section we will discuss the current literature and techniques on sensor networks, semantics and multi-agent systems. These three topics are the main building blocks for the framework that is proposed in Section 4. With this framework we can evaluate the effect of knowledge sharing in a distributed sensor network. If the reader is not familiar with these topics we advise them to read these sections carefully. Section 2.2 on semantics is a prerequisite to be able to understand Sections 33.1 and 3.2 on question federation and data conversion.

2.1 Sensor Networks

This section discusses sensors and sensor networks and also outlines a number of characteristics of sensor networks in general. These characteristics constitute the main design decisions when choosing a sensor network architecture; which is the first challenge in Section 1.3.

We deliberately left out information about network routing, which is an important research issue in the sensor network field, but not applicable to our research. The interested reader is referred to earlier research [6-12]. This section is one of the three building blocks of the architecture discussed in Section 4. The other two building blocks are Section 2.2 about semantic web technologies and Section 2.3 about multi-agent systems.

Examples in this section are given from the IJkdijk scenario as it was discussed in the Introduction Section.

2.1.1 Sensors

Real world phenomena are observed through sensors. In humans, senses are the physiological methods of perception. Aristotle was the first to make a classification of the human senses; sight, hearing, touch, smell and taste. The mechanical sensors that constitute sensor networks also measure physical conditions and signals. However, where the number of human senses is limited, the different types of mechanical sensors is huge and new sensing methods will be added in the future.

The first mechanical sensors that were to appear sense physical parameters in a passive manner. Nowadays, multiple sensors can be connected to a **sensor node**, which is a fully fledged computer with the capability of processing sensor readings and sharing these readings with neighbouring sensor nodes [10], which is a prerequisite for knowledge sharing in sensor networks.

When a sensor node measures different phenomena through different sensors, we call this sensor node heterogeneous. Consider the following sensors:

- sensors that measure a continues physical signal (temperature, water level)
- sensors that measure a state (availability)
- sensors that measure location (geographical position)

Measuring continuous data seems distinctly different from measuring a state. However, if we generalize the above examples we can treat each case in the same way. The sensor measures a phenomena, but in each of the above cases with a different time interval and a different measure. A sensor that measures a state, for instance the availability of enough energy resources, can be seen as a continues measurement with an interval of minutes. A sensor that measures temperature can measure this value with an interval of seconds. A location sensor also handles data in the same way as a temperature sensor, but instead of measuring temperature, it measures position coordinates.

A single sensor can provide useful information about a single phenomena in the real world. However, complex interactions between phenomena in the environment can only be measured by combining sensor measurements from different sensors. That is why we want to combine these sensors in a network, as discussed in the next section.

2.1.2 Network of Sensors

In this section we will classify sensor networks and address different types of sensor networks. A sensor network is designed to detect events or phenomena, collect and process data and transmit sensed information to interested users [31]. Sensor networks distinguish themselves from other wireless and ad hoc networks by their ability to cooperatively sense a phenomenon with a dynamic network topology of sensors.

Individual sensor readings are useful for monitoring a single phenomena. Moreover, a combination of sensors makes it possible to measure interactions between phenomena. In a sensor network, multiple heterogeneous sensors are connected to make it possible to combine sensor readings. As an example, sensors in a network can have different geographical locations, which makes it possible for a user to pose interesting questions about a wide range of spatially distributed phenomena. There are a number of design decisions to take into account when constructing a sensor network. These will be explained in the following paragraphs.

What is being measured by the network

The type of sensors that are contained in the sensor network determine whether a network is **heterogeneous or homogenous**. If all sensors measure the same phenomenon we call the network homogenous [27]. If the network is capable of measuring different phenomena we call the network heterogeneous.

The following three paragraphs each discuss a different view on architecture. The data architecture determines what data flows through the network. The physical architecture describes how the network architecture is limited by setup of the network in the environment. The communication architecture describes which sensor nodes are allowed to communicate.

Data architecture: Centralized versus Distributed approach

When designing a Sensor Network one has to decide whether the sensor data is going to be stored in a central repository or whether the data will remain distributed close to the source; the sensors. In the IJkdijk scenario we have seen a preference for a distributed approach, however, there are some trade-offs to consider.

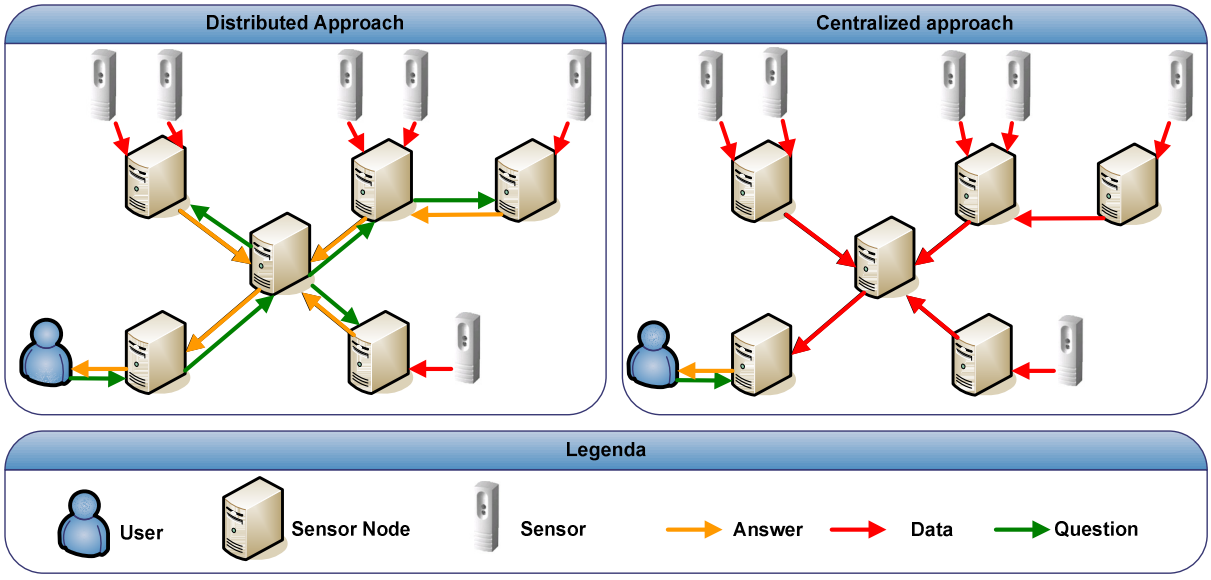


Figure 2: Centralized vs. Distributed

In a centralized approach, a question about sensor data is answered locally on local data. An example of this is shown in Figure 2 on the right. It requires all the data to be constantly sent to a central repository (indicated with a red arrow). The question (green arrow), posed by a user is answered by a single sensor node. Only a single answer has to be computed (orange arrow). This can

be useful if you require all this data, or when a backup is needed of all the sensor data. One has to consider whether this overhead of data transfer is needed for an application. Being able to have all data centrally has advantages when it comes to answering questions about the data. It is not necessary to invent complicated protocols for question splitting and federation.

In a distributed approach (Figure 2 on the left), the question about sensor data is split into sub questions which are sent to each distributed sensor node individually. This approach also goes by the name of in-network processing [21]. One can consider this as bringing the model to the data as opposed to bringing the data to the model. Advantages of this approach are that the data can remain near its source and only answers about this data is sent over the network when a question is received. The network traffic will be considerably lower, however, the distributed sensor nodes will have to have more storage and processing capacities.

As always, there is an in between solution. Partial distribution might help to overcome data traffic and data storage issues. Active knowledge acquisition of sensors in the network can result in semi-distributed aggregation points in the network, where knowledge about the direct environment is more dense.

The choice of each of these three approaches depends on the available bandwidth, storage and processing capacity, number of questions handled by the network, the complexity of the interaction of sensor measurements in the network and the size of the total data set of sensor measurements.

Physical architecture: network topology

The **network topology** describes the physical arrangement of sensor nodes and sensors. There is a distinction between sensor networks with a designed topology and sensor networks with a dynamic topology. A sensor network topology can be designed if the position of the sensors remains fixed. The underlying infrastructure of the area where the network is deployed determines the network topology. By designing a network topology resource-rich sensor nodes can be efficiently placed in the neighbourhood of nodes with a lower capacity. In case of the IJkdijk scenario, most of the sensors will be positioned at a fixed location along the embankment, resulting in a designed topology.

There is also a wide range of situations in which it could be useful to have a more dynamic approach of monitoring. In a field based monitoring task, sensor nodes are randomly scattered. Each node uses communication to determine its neighbours. In this kind of topology it is more difficult to design the interaction between sensor nodes in order to increase efficiency. We can see there is a trade-off between flexibility and network efficiency.

Communication architecture

Besides the physical arrangement of sensors in a network there is also a communication architecture which describes which sensors can talk to each other. This architecture determines how data flows through the network. This influences both question federation and knowledge sharing in the network. Sensors that cannot communicate with each other are unable to share knowledge or send user questions to one another. Although this sounds like a disadvantage for knowledge sharing, limiting

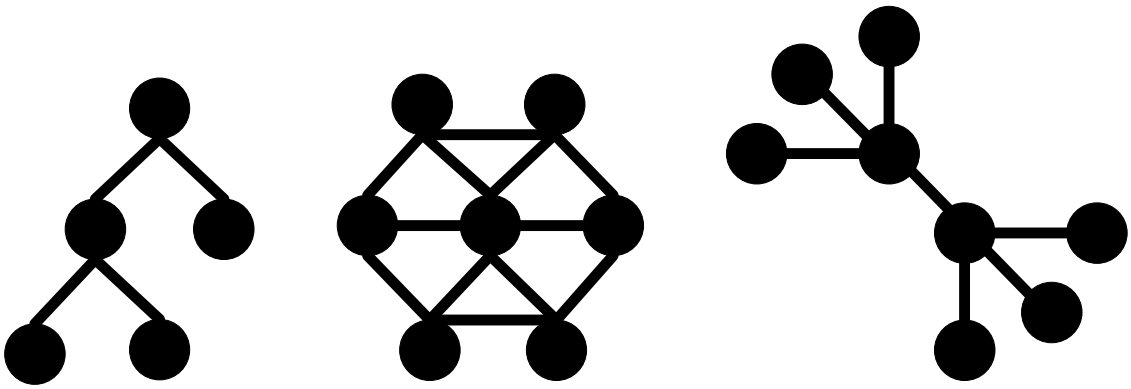


Figure 3

communication possibilities in a network effects the scalability of the network as the amount of sensors in the network increases.

According to Jain et. al. [10] there are two main types of network architectures: the **hierarchical network architecture** and the **flat network architecture**. We think a third architecture should be added, the **clustered network architecture**.

An hierarchical network architecture is organized into a tree-like structure, with a root at the base and multiple leaves at the ends, see Figure 3 (a). This type of network has the characteristic that nodes have few child or parent relationships and that horizontal connections between nodes are not allowed. As data flows from the root to a leave it goes through the nodes in a fixed pattern. An advantage of this approach is that the creation of a hierarchy can simplify the possible ways in which a message flows through the network; it limits the ways in which messages can be routed between nodes.

In a Flat network architecture (or P2P architecture) all nodes are equal and connections are set up between nodes within each other's range, see Figure 3 (b). In such a network, multiple routes from one node to another are possible. This implicates that each node should be able to recognize a message that is distributed in the network as a new message, or one that was seen before. An advantage of the flat network architecture is that it is more robust. Every node can route requests and is thus not depending on a failure prone tree structure. A disadvantage is that message distribution is less efficient. As the amount of sensors in the network grows, the number of connections between sensors in a flat network architecture grows exponentially. This has considerable effect on the scalability of the network since the amount of messages in the network increases exponentially as well.

In a clustered network architecture, see Figure 3 (c), also known as clustered P2P network [32], advantages of both previously discussed architectures are combined. Part of the network is clustered and clusters are connected through a small number (one or more) nodes in each cluster. Messages can now be directed fairly quickly to the correct cluster, in which it is distributed accordingly. A clustered architecture can be used to influence node accessibility top-down. The few nodes that connect clusters together are the only nodes who have access to the other nodes in the cluster and thus restrict access these nodes by functioning as a gateway.

2.1.3 Events Subscriptions and Questions

Having discussed different views on sensor network architectures we can now focus on the messages that flow through the network.

In the domain of sensor networks we can typically think of two types of messages: event subscriptions and questions. A question is initiated by a user and is directly sent to distributed sensor nodes. An event subscription is a message from a user or sensor node to another sensor. The subscription consists of both an event to trigger on and a question about the event. When a local condition at a sensor node changes, the event can trigger the subscription and an answer to the question is sent to the user or sensor node that initiated the subscription.

In a network of temperature sensors one can ask what the average temperature is of the sensors in the network. This question has to be distributed to each temperature sensor and the answer of each of these has to be combined to calculate the average. A user could also wish to be informed whenever the temperature raises over 25 degrees Celsius somewhere in the network and if this happens the user would like to know on what location this temperature raise was measured. The trigger condition, in this case is a temperature over 25 degrees Celsius, and the question, in this case the location of the sensor that triggered the event, are distributed in the network. If the condition is triggered by a sensor, the answer to the question is sent to the user or sensor node that initiated the subscription. Event subscriptions play an important role in actively acquiring knowledge by sensor nodes. A sensor nodes that has event subscriptions with other sensor nodes can require knowledge of changes in the environment that are not directly measured by its own sensors.

2.1.4 Data aggregation

An important issue in sensor networks is where to store data. Data can be stored at the sensor nodes, or centrally as we mentioned before. However, a midway solution is available. It is also possible to add certain aggregation nodes to the network. Such a node aggregates information from other nodes, performs some pre-processing on the data and stores it locally. Aggregation is an aspect of knowledge

acquisition. One could use this approach to reduce resource requirements at sensor nodes and still use a distributed network architecture. This pre-processed data can then be published in the sensor network and used by other nodes. To make the idea clearer, consider the IJkdijk scenario. There are a number of water level sensors. An aggregation node could be created that keeps track of the changes in water level by all the water level sensors. The aggregation node can then calculate the maximum, minimum and average water level along the embankment as a pre-processing step. A question about the average water level along the embankment can now more efficiently be dealt with by addressing the aggregation node instead of the individual water level sensors.

2.1.5 Data sources

So far we have been talking about a network of sensors. We would like to point out that a sensor is nothing more than a device or program that measures a change in the environment and stores this in a local knowledge base. Having said that, we can also connect other data sources to the network. For instance, a weather forecast database that is maintained by a local weather station or a knowledge base that is maintained by a fanatic fisherman who makes estimates of the number of fish in the river along the embankment.

The next section gives an introduction to semantics and current available semantic web technologies. It starts with an anecdote about an application of the semantic web and will work towards using the semantic web techniques in the sensor network domain. Numerous examples in the domain of sensor networks will be given.

2.2 Semantic Web Technologies

Consider the example where you are going to a conference and you would like to get a list of people that are interested in the same research area as you. Moreover, you would like to know if they are speaking on the conference, when and where. You also want to know if there is a time overlap between the talks so that you can attend them all.

Questions like this would be difficult to answer on the current world wide web or any other kind of network. Current keyword based search engines will probably present you the proper web pages to find the program of the conference and the list of speakers and the time of their talks. However, combining these results is up to you. To be able to answer these kinds of questions we have to go beyond keyword searching and add meaning to capture the semantics of data.

With the introduction of the Semantic Web [1], which was originally invented as a new version of the World Wide Web, a lot of possibilities for adding context and understanding to networks have arisen. The Semantic Web builds on the idea that the same resource can be described at multiple locations on the web and is referred to by a unique identifier. The unique identifier is used to link different properties to a single resource, even if they are described by different parties at different locations on the web. As an example, a person Pete is represented by a single resource. On his personal web page a property ‘ email address’ is linked to Pete’s resource. On his company web page a function description property is linked to his resource. The two can be combined by comparing the unique resource identifier of Pete. The combination of the two properties makes up a more complete description of the resource Pete. Annotation with semantic information makes acquisition of meaning for machines possible [33]. Standardization techniques for representing semantic data in combination with a query language makes it possible to combine data from all over a network in order to come up with new information, or to answer questions posed by a user [34].

In this chapter we will look into the different building blocks of the semantic web. We will give examples in the sensor network domain of the IJkdijk scenario, instead of the world wide web domain. First we define semantic knowledge and show which description techniques exist to annotate data semantically (Section 2.2.1). Then we will show how ontologies can be used to link semantic knowledge, a step towards data integration. (Section 2.2.3). In Section 2.2.6 we will mention current limitations when it comes to maintaining a semantic knowledge base with existing techniques. Further, we discuss rule and inference engines that are needed to reason about semantic knowledge to infer new knowledge (Section 2.2.4). This is a new possibility that follows from knowledge sharing. Next, we discuss query languages that can be used to query semantic data (Section 2.2.5). Finally we give a short overview of existing semantic web frameworks and some other useful tools (Section 2.2.7).

2.2.1 Semantic Knowledge

Semantics is the study or science of meaning in language. It describes the meaning or the interpretation of a word, sentence, or other language form. The importance of semantics shows in the domain of sensor networks: how can sensors describe sensor measurements and answer questions about these measurements without an adequate definition of the semantics of these measurements? Therefore a machine interpretable language is required, which facilitates the annotation of sensor data and the communication about these measurements.

As in geographical languages, in machine interpretable languages there are many ways to say the same thing in a different form. Consider the following ways to say something about a measurement of a sensor:

- Sensor T measures 25 degrees Celsius
- 25 degrees Celsius was measured by Sensor T
- The measurement of sensor T is 25 degrees Celsius

How a sensor describes its sensor measurements is specified by semantics. The semantic annotations of sensor data by a sensor constitute the “language” the sensor speaks. Semantically annotated data is normal data which is given some extra context information. For instance, consider the data value 25. This value has no meaning, unless we define what kind of value it is. If we attach to this value the context information that it is a temperature reading, than it becomes knowledge. Moreover, we could annotate this value as being a degrees Celsius value, making it more specific and thus usable in a wider context.

The way a sensor annotates its sensor measurements (the language it speaks) is defined and structured by an ontology. An ontology describes the structure of data (it describes a language) and is a widely used concept in information sciences [35]. An ontology describing a machine interpretable language defines classes and properties, with which sensor data can be structured.

Other types of semantic knowledge exist. A semantic reasoning rule can manipulate and combine semantic data to create new knowledge. For instance in the IJkdijk scenario, if we have a pressure reading and a water level reading, a semantic rule could be: if the pressure on the embankment is over 25 Newton per meter and the water level is lower than 150 centimetres, then there is a high chance of flooding.

Data conversion knowledge describes how to convert from one data type to another. An example of this is the conversion of a degrees Fahrenheit temperature to a degrees Celsius temperature. This conversion can be described mathematically. Other types of conversions are string conversions and higher cardinality conversions, such as one-to-many and many-to-many conversions.

The term knowledge is a term with many possible interpretations. In the rest of this report, when we talk about semantic knowledge we refer to the combination of semantically annotated data, ontological metadata, semantic reasoning rules and data conversion knowledge. The upcoming three paragraphs go into more detail on these variants of semantic knowledge.

2.2.2 Semantic data annotation

So how do we annotate data with semantics? Semantic data can be stored in a statement, which is a triple of subject, predicate and object. Take a look at the following example:

"The measurement of sensor T is 25 degrees Celsius".

- The subject of the statement above is: sensor T
- The predicate is: measurement
- The object is: 25 degrees Celsius

We can plot statements like this in a graph, see Figure 4 (a). The object part of a statement can also be a reference to another subject. This allows for the creation of a graph which connects multiple statements. Figure 4 (b) is an example of such a more complex graph. In this example, a blank node is added, which connects multiple statements. We would translate this graph like: "Sensor T is of type

temperature sensor and has a measurement with value 25, where this value is of datatype degrees Celsius".

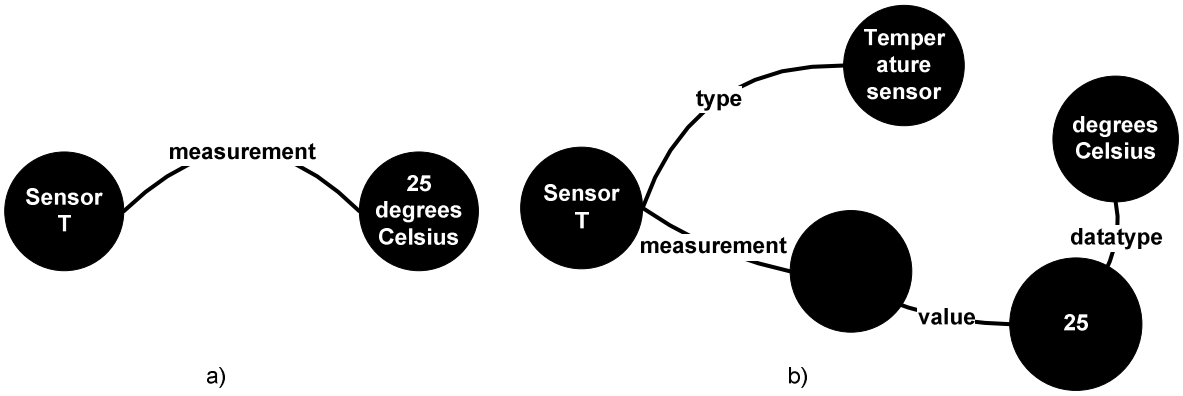


Figure 4: graphs of sensor data annotated with the resource description framework (RDF)

A popular description language for semantic triples is the RDF format (Resource Description Format) [36, 37], which is supported by the W3C. Because RDF is invented as a resource description model for the World Wide Web, it is designed with the purpose of being applicable in a distributed environment.

An RDF resource, which can occur in the subject, property and object position of a graph as shown in Figure 4 is described by a URI. A URI is a unique identifier for a resource. The URI is an important part of the semantic solution to the data integration challenge (Section 1.3), because its uniqueness allows the comparison and integration of resources over multiple distributed sources.

We will give an example of what an RDF description looks like. Standard RDF format is an XML inspired formatting, however a more readable and smaller alternative to RDF XML exists: RDF Notation 3 in which statements are actual triplets without their XML specific hierarchy constraints [38]. Table 1 shows the description of the graphs shown in Figure 4 (b) into the Notation 3 format. The right of Table 1 shows a legend of the specific Notation 3 characters that are used to describe triples. The example of Figure 4 (b) is described two times. In Table 1 on the top left is an example of an original Notation 3 annotation. On the bottom left is a rewritten version of the same example, but here the actual subject predicate object triples are made explicit.

The Notation 3 formatting is advantageous over the XML variant because knowledge bases can be concatenated without having to consider the hierarchy when combining knowledge bases in XML RDF format. Another advantage is that the notation is more compact and can thus save bandwidth during transmission between sensor nodes in a sensor network.

Annotated sensor data in Notation 3 format	Legend for Notation 3	
@prefix : <http://ijkdijk.nl/example#> . @prefix units: <http://ijkdijk.nl/units.owl#> . [ ] a :TemperatureSensor; :measurement [:value 25^^units:DegreesCelsius] . _:b1 a :TemperatureSensor . _:b1 :measurement _:b2 . _:b2 :value 25^^units:DegreesCelsius .	[ ]	A blank node, with a unique identifier
	;	This sign denotes that the next triple starts with the same subject as the current one. The subject is thus omitted in the next triple
	[p o]	The property (p) and object (o) of the triples enclosed in [ ] brackets share the same blank node
	a	Equivalent of rdf:type, denoting that the subject is a class of the object type
	^^	Denotes the rdf datatype of a value

Table 1: Semantic triple annotation with RDF

Now the idea of semantic annotation with triples has been made clear, we discuss ontologies and how they can be used to add knowledge about knowledge.

2.2.3 Ontologies

This section explains the ideas behind ontologies. We kick off with giving a definition of an ontology. Then we will show how ontologies can be described with RDFS and OWL. We will show how ontologies can be used to describe mappings between different semantic knowledge bases and how we can describe deprecation of concepts in an ontology. We end this section with a brief overview of existing ontologies.

An ontology defines a taxonomy and a set of rules. The taxonomy defines classes of objects and relations among them, the rules can be use to reason about semantic data to derive relationships from the taxonomy and the data described by the taxonomy, check completeness of the data or validate the structure of semantic data [2].

As Tim Berners-lee, the inventor of the World Wide Web mentioned in an interview on the 5th International Semantic Web Conference 2006 the term ontology can be used in two ways. First, an ontology can be used for instance to describe the parts of the human body and how they connect. Such an ontology uses instance data to describe relationships. Take a look for example at Figure 5. This **hierarchical ontology** describes a hierarchical relationship between locations in the IJkdijk scenario.

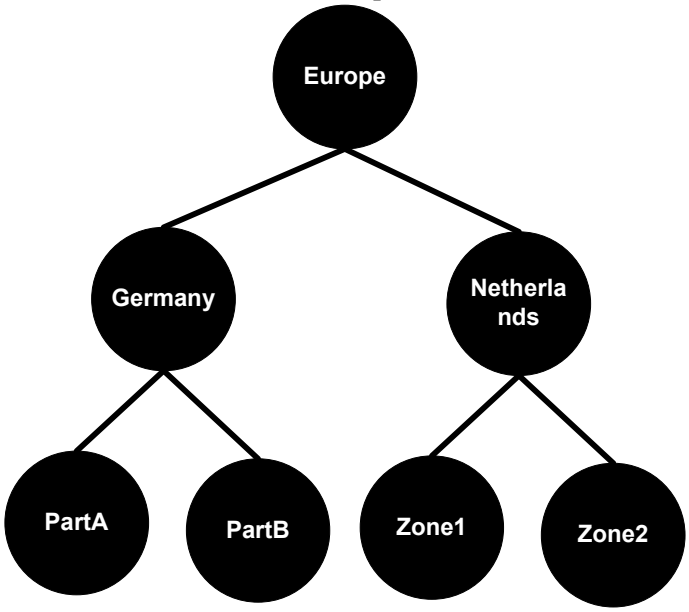


Figure 5: Hierarchical ontology

An **infrastructure ontology** on the other hand describes how to define each resource. It specifies the classes and properties of each class and with this restricts the way semantic data can be annotated. A useful characteristic of such an ontology is that it becomes possible to determine the relevance of an information source, without accessing the underlying data [39]. We will come back to this later in Section 3, where we describe how questions can be federated based on ontology metadata.

In the next two paragraphs we discuss the RDFS schema language and the Ontology Web Language OWL, a popular ontology language for the semantic web.

RDFS

RDF Schema (RDFS) is a light weight ontology language for RDF. This description language is based on RDF itself and consist of a set of primitives to define classes, sub classes, properties and sub properties. RDFS comes with a set of rules that can derive properties and classes of a resource by using sub property and sub class annotations in the RDFS schema.

OWL

The Ontology Web Language (OWL) is designed to enable efficient representation of ontologies that are also amenable to decision procedures [38, 40]. OWL can be seen as a layer on top of RDF and is a more expressive variant of RDFS. OWL comes in three variants: OWL Full, OWL DL and OWL Lite, in which the first is the most complete and complex and the latter are subsets of OWL full. In OWL

one can construct classes where in RDFS you can only name classes, moreover you can limit the cardinality range and value range of classes. Table 2 shows two ontology examples. On the left an OWL description of the hierarchical ontology in Figure 1. On the right an infrastructure ontology description which restricts the way in which in a knowledge base a sensor and its measurements have to be annotated. In Section 2.2.4 we will discuss how these ontologies can be used for semantic reasoning.

OWL comes with a number of predefined properties to define the infrastructure of a semantic dataset. These properties include: subClassOf, subPropertyOf, domain, range, equivalentClass, equivalentProperty, sameAs. DeprecatedClass, DeprecatedProperty, versionInfo and many more. The strength of OWL is that it comes with a set of rules which uses the relations in the ontology described by these properties to derive new facts about the semantic data which is described by the ontology. More will be said about rules later in this section.

Out of the total set of properties defined by OWL, we give some special attention to the domain and range properties. For the remaining properties we refer the reader to [40]. A property can have a **domain restriction** on a class. This restricts the subject value of a triple, with this property as a predicate, to an instance of a class of this domain. As an example, consider the following triples:

```
:sensor1 a :Sensor .
:sensor1 :measurement _:m .
_:m :value “25”.
```

The predicate :measurement is an ObjectProperty in the infrastructure ontology below. This property has a domain restriction to :Sensor, which is an owl:Class. According to the domain restriction in this ontology the subject of :measurement, which is :sensor1, should thus be a resource of class :Sensor. The above example is thus correct given this ontology, since the first triple shows that :sensor1 is of type :Sensor.

The **range restriction** of a property restricts the possible values that can occur in the object of a triple. In the ontology below we see that the range of a :value property is any instance of the class xsd:integer. In the above example we see the value is “25”, which is a valid instance of the integer class.

Both domain and range values in an ontology restrict the subject and object values of triples in a knowledge base described by the ontology. This is helpful since it allows the validation of a knowledge base and also allows to group triples together, as we will see later in the section on Question Federation (Section 3).

Hierarchical ontology description	Infrastructure Ontology description
@prefix : <http://ijkdijk.nl/locations.owl#> . @prefix owl: < http://www.w3.org/2002/07/owl#> . :Continent a owl:Class . :Part a owl:Class . :Zone a owl:Class . :Country a owl:Class . :partOf a owl:ObjectProperty . :Europe a :Continent . :Germany a :Country ; :partOf :Europe . :partA a :Part ; :partOf :Germany . :partB a :Part ; :partOf :Germany . :Netherlands a :Country ; :partOf :Europe . :zoneA a :Zone ; :partOf :Netherlands . :zoneB a :Zone ; :partOf :Netherlands .	@prefix : <http://ijkdijk.nl/sensors.owl#> . @prefix owl: <http://www.w3.org/2002/07/owl#> . @prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> . @prefix xsd: <http://www.w3.org/2001/XMLSchema#> . :Sensor a owl:Class . :Measurement a owl:Class . :measurement a owl:ObjectProperty ; rdfs:domain :Sensor ; rdfs:range :Measurement . :value a owl:DatatypeProperty ; rdfs:domain :Measurement . rdfs:range xsd:integer . :unit a owl:DatatypeProperty ; rdfs:domain :Measurement .

Table 2: Ontology description

Two important aspects for the sensor network domain is how to recognize whether semantic annotations of sensor data at one sensor node is still in sync with the annotations at another node. An OWL ontology incorporates properties for version information and deprecation. Version information is useful to check whether an ontology has changed or not. Deprecation information is useful to find out how the ontology has changed since the previous version. By incorporating version and deprecation information into an ontology, it is possible to allow data integration (see Section 1.3) to continue even when semantic annotations change. Deprecation knowledge is one of the types of knowledge that is useful to share in a sensor network. Sensor nodes can use deprecation knowledge to adjust the communication with each other about shared knowledge.

Ontology mapping

In a distributed environment, if each data source would be annotated following a global ontology infrastructure (each data source speaks the same language), it would be fairly easy to combine annotated data from different sources. However, in Sensor Networks and many other distributed applications, data sources are described by different ontologies (different languages). To still be able to combine the annotated data of each of these sources, we have to provide a mapping between the different ontologies. An ontology mapping defines the equivalences between classes, properties and resources of one ontology with another and can thus be seen as a translation between languages.

Different approaches to ontology mapping have been suggested. Some of these are based on machine learning techniques in which naming conventions and hierarchical information are used to map ontologies automatically [41-44]. Although automatic schema mapping can save a lot of work when mapping huge ontologies, still they do not provide a complete mapping, leaving it to the user to check the validity of the mapping. Another approach is adding and maintaining equivalence relationships with other ontologies during the construction or revision of an ontology. This manual approach has the advantage that ontologies can be mapped partially. Only those equivalences that make sense to map are added to the ontology.

The Ontology Web Language (OWL) has a number of built in properties to describe equivalences with other ontologies in an ontology. The equivalentClass and equivalentProperty properties denote equivalences between classes and properties. The sameAs property denotes equivalences between resources. Table 3 shows a combination of annotating deprecation as was discussed in the previous paragraph and equivalences between classes, properties and resources in an ontology. For simplicity this is a mapping within a single ontology, but the same concept applies when mapping different ontologies, with the one difference that the resource URI is different.

```
@prefix : <http://ijkdijk.nl/example.rdf#> .
@prefix owl: < http://www.w3.org/2002/07/owl#> .
@prefix rdfs: < http://www.w3.org/2000/01/rdf-schema#> .

:Sensor a owl:Class.

:SensorDevice a owl:DeprecatedClass ;
   owl:equivalentClass :Sensor .

:measurement a owl:DeprecatedProperty ;
    rdfs:domain Sensor ;
    rdfs:range Measurement .

:reading a owl:ObjectProperty ;
    rdfs:domain Sensor ;
    rdfs:range Measurement ;
    owl:equivalentProperty :measurement .

: Sensor_1 a :Sensor .

:TSensor_1  a :Sensor ;
  owl:sameAs :Sensor_1 .
```

Class equivalence

property equivalence

Resource equivalence

Table 3: Ontology mapping & Deprecation

Describing a mapping with OWL is possible, but limited. It is not possible to describe a higher cardinality mapping between classes and properties of two ontologies. For example, consider two velocity sensors that both measure the speed and direction in which an object is moving. One of the sensors stores the direction as a product of the direction in the X, Y and Z angle of the object in three dimensional space. The other sensor stores the direction of the X, Y and Z angles separately. Both sensors thus have knowledge of the direction in which an object is moving, but they store it in a different cardinality. To build a mapping between these two annotations, it should be possible to describe a higher cardinality mapping that relates the one value of the first sensor, to the three values of the second sensor, which is currently not possible in OWL.

A final note about ontology mapping. These ontology mapping related issues arise only when the distributed sources are maintained by different parties who use different ontologies. We should note however that these problems do not have to arise when a closed system is being designed by one or a few parties. In a closed system it is easier to make conventions about naming, shared ontologies and changes to ontologies.

2.2.4 Reasoning with semantic knowledge

Reasoning is the process of using knowledge to generate new knowledge. This can be useful in dynamic distributed environments that tend to change a lot. In the sensor network domain, by reasoning with knowledge in a network new knowledge can be derived. In a sensor network environment such as the IJkdijk, this knowledge can be used in several ways. Reasoning can be used to derive new knowledge from already known facts. A sensor could for instance derive that the probability of an embankment flooding is high based on its knowledge on current water levels along the embankment and the pressure of the water on the embankment. A sensor could also derive that another sensor is positioned in the Netherlands if it knows that this is located in Zone A and it has the knowledge that Zone A is part of the Netherlands.

Inference on semantic data can reveal new relationships between data elements. Inference is done through the use of a rule language which can be executed by a rule engine to create new statements by applying the rules on an existing set of semantic statements. There are a number of Semantic Web rule languages like the Semantic Web Rule Language (SWRL), the Semantic Web Services Language (SWSL) and the Web Rule Language (WRL). A rule engine uses rules to deduce new facts or to transform semantic data.

We distinguish two types of rules. OWL rules are a set of ontology related rules that can be used to derive classes and properties from an infrastructure ontology. The set of rules that comes with OWL is consistent for OWL full, DL and lite. Custom rules are all the rules you can think of that have not been defined by OWL. Custom rules can use hierarchical ontologies to derive new facts from existing knowledge. The rule itself contains knowledge in that it describes a relationship between data instances in a semantic knowledge base. Existing rule engines that support reasoning with OWL ontologies include Pellet [45] and Racer [46]. The Jena rule engine [47] also adds the possibility to reason with custom rules in addition to OWL rules.

Rules

The semantic rules that we will discuss are if-then rules [48], which consists of an IF-statement and a THEN-statement The IF-statement specifies the conditions to which the rule should apply in order to make the THEN-statement valid. Below is an example of an OWL rule that is used to derive class information from sub class properties in an infrastructure ontology:

IF X is a temperature sensor and temperature sensor is a sub class of sensor THEN X is a Sensor

The following rule is an example of a custom rule inspired by the IJkdijk scenario:

IF the water level > 150 centimetres and pressure > 15 Newton meter THEN flooding risk is high

A custom rule that uses an hierarchical ontology to derive locations on the embankment:

IF X is part of Y and Y is part of Z THEN X is part of Z

Table 4 shows a translation of these rules in the Jena Semantic Framework rule format [47]. A Jena rule has semantic triples of subject, predicate and object in both the IF and the THEN statement of the rule. The IF triples are matched against a semantic RDF knowledge base and if these triples match, the result of the THEN statement is added to the knowledge base. By using RDF triples, the Jena rule engine understands the semantics of the RDF knowledge base and can change the knowledge base accordingly.

The extensive rule set of OWL operates on infrastructure ontologies, whereas the custom rules operate on the data in the knowledge base or hierarchical ontologies. Both types of rules can increase knowledge for sensors such as those in the IJkdijk scenario.

OWL rule	Custom rule	Custom rule
[rdfs1: (?a rdf:type ?x) (?x rdfs:subClassOf ?y) ->(?a rdf:type ?y)]	[rule1: (?x :waterLevel ?w) (?x :pressure ?p) greaterThen(?w, 150) greaterThen(?p, 15) ->(?x :floodingRisk :high)]	[rule2: (?X locations:partOf ?L) (?L locations:partOf ?C) –>(?X locations:partOf ?C)]

Table 4: Jena Rules

The next section will discuss Semantic Querying as a means to be able to ask questions about semantic data. This is the first step towards addressing the third challenge on query federation as was stated in Section 1.3.

2.2.5 Querying Semantic Knowledge

Semantic annotation of sensor data, ontologies and semantic reasoning make data integration possible in a sensor network environment. An important missing aspect of data integration is how to actually combine semantic data. A semantic question can be used to extract knowledge from a semantic knowledge base. To be able to pose semantic questions about annotated sensor data, we need a query language. A query language can be used to select data from a knowledge base, by specifying the semantic resources, their relations and the conditions to which they should apply.

Since the invention of the semantic web, two main types of query languages for querying semantic data (in RDF format) have been proposed. The most popular query languages for the semantic web are SQL inspired languages like RDQL/Squish, RQL and SPARQL. Other query languages like DQL and RDFQ express the query as an RDF graph. Graph like query languages have the inability to handle blank nodes [49], an example of which was shown in Figure 4 (b). For this reason we will go into more detail on the most popular and expressive SQL inspired language, supported by W3C: SPARQL [50].

The SPARQL query language enables querying over distributed RDF data. It uses the RDF URI principle to combine different resources in an RDF graph and extracts knowledge from them. SPARQL is data-oriented, which means that it only queries the information contained in an RDF dataset. Thus, there is no inference in the query language itself. Recall the following question from the IJkdijk scenario (Section 0):

Where along the embankment is the pressure the highest?

The question is about the knowledge in the knowledge bases of pressure sensors in the IJkdijk scenario. Table 5 gives an example of such a knowledge base. It describes a pressure sensor at location zone A, with a measurement of 25 degrees Celsius.

Example knowledge base for Pressure Sensor

```
@prefix sensors: <http://ijkdijk.nl/sensors.owl#> .
@prefix units: <http://ijkdijk.nl/units.owl#> .
@prefix locations: <http://ijkdijk.nl/locations.owl#> .
@prefix : <http://ijkdijk.nl/sensornode#> .

:Sensor_1 a sensors:PressureSensor;
   sensors:location locations:zoneA ;
   sensors:measurement [
                        sensors:value 25^^units:DegreesCelsius
                       ] .
```

Table 5: Knowledge base with sensor data

Based on this knowledge base a SPARQL query as shown below in Table 6 is created.

```
PREFIX sensors: <http://ijkdijk.nl/sensors.owl#>
PREFIX sensornode: <http://ijkdijk.nl/sensornode#>
PREFIX units: <http://ijkdijk.nl/units.owl#>
PREFIX locations: <http://ijkdijk.nl/locations.owl#>


SELECT
        ?sensor
        ?location
        ?pressure
WHERE
{
        ?sensor  a sensors:PressureSensor ;
                 sensors:location ?location ;
                 sensors:measurement ?m .
        ?m sensors:value ?pressure .
}

ORDER BY DESC(?pressure) LIMIT 1
```

URI prefixes

Knowledge to extract

Conditions on the knowledge base

Order and limit the results

Table 6: Example SPARQL query

A SPARQL query consists of a select and a where part. The where part specifies the conditions that should apply to the triples that are selected from the RDF knowledge base. The variables (recognizable by the question mark) in the where part of the query are replaced by the resource URIs in the knowledge base that match the condition triples. These variables are selected in the where part of the query. The prefixes in the query are used as a shorter way to describe full URIs in the query. The order by clause sorts the results from the query and the limit specifies that we only want the largest pressure measurement returned. This query, executed on the knowledge base in Table 5 returns the following mapping between variables and resource URIs:

sensor	location	pressure
sensornode:Sensor_1	locations:zoneA	25^^units:DegreesCelsius

Besides a select query as shown in Table 6 the SPARQL query language has a construct query. The where part of the query is the same as for the select query, but instead of selecting variables in the

select part, the construct query builds up triples which form a new set of RDF statements. The construct query thus outputs RDF instead of variable bindings.

The SPARQL query language has some additional features that are worth mentioning. A filter clause can be used to restrict the results that a query results. In the above example we could add a 'FILTER(?pressure > 30)', meaning that only results are returned for which the pressure is over 30 (results in an empty result set in the above example). Moreover, the language allows the addition of custom property functions. A **property function** can do something with the semantic data during its lookup in the knowledge base by the query engine. A property function can be added as a normal triple to a SPARQL query. The predicate of this triple is the name of the property function, the subject and object can be variables or values that are evaluated and changed by the property function. In the example below, the apf:splitIRI is a custom property function that splits the URI of the ?location resource into a namespace URI and a local name. These new variables can then be selected in the select part of the query.

```
?location apf:splitIRI (?namespace  ?localname)
```

In the JENA framework, the property function is implemented as a Java method that has as input the subject and object of the triple and can add new results and variables to the result of the SPARQL query. SPARQL property functions are an important prerequisite for our work on Data Conversion, which will be discussed in Section 3.2.

SPARQL is a query language that can select semantic data from a knowledge base, however, it still leaves open the question of how to select semantic data from multiple distributed sources. This is a requirement to integrate distributed semantic knowledge, such as that maintained by the sensor nodes in the IJkdijk sensor network. To enable distributed question answering Query Federation is needed. This process splits queries based on information available at the different sources in the network. A full explanation of our work on Query Federation is given in Section 3.

2.2.6 Maintaining a Semantic Knowledge Base

A number of issues arise when a sensor node has its own knowledge base that has to be maintained. These issues include: handling new sensor data, changes to ontology infrastructure of data and permissions on data.

The semantic web techniques are useful to annotate sensor data with semantic metadata. However, the current RDF specification does not incorporate tools or methods to handle the maintenance of an RDF dataset. Other data storage techniques, such as databases, have supporting query languages such as SQL that make it possible to insert data into the database, replace existing data or remove data from the database. The most advanced query language for handling an RDF knowledge base: SPARQL, is currently limited to a select language only. This means that the language does not incorporate the ability to update, replace or delete semantic data from a knowledge base.

Handling new sensor data is thus not obvious and is mainly a design decision when implementing a sensor node. A possible temporary solution to this lack of support in SPARQL is to utilize ontology metadata to determine when to add new knowledge and when to replace. Since an ontology defines the infrastructure of a knowledge base, we can add metadata about this structure to define what fields are unique and should thus be replaces, and what fields can have multiple instances. By adding such metadata it becomes possible to handle new sensor data semi-automatically.

The internal knowledge infrastructure of a sensor node can change, due to changes in the environment or user inflicted changes to the sensor node. When the knowledge base maintained by a sensor node changes, then the ontology that describes this knowledge base should change as well. By maintaining deprecation information in the ontology of a knowledge base a sensor node can keep track of changes in the description of its knowledge. This deprecation information is knowledge worth sharing with other sensor nodes.

Finally, in a sensor network, such as the IJkdijk, it might be of importance that sensor nodes are able to limit the availability of their knowledge to only a number of other sensors in the environment. There is currently no standard way of describing access permission for semantic knowledge. A possible solution could be to add metadata which describes permissions on each semantic resource in the knowledge base. This could be done by adding an :accessibleBy property to

the ontology, which is then used by the sensor node to allow or deny incoming questions from other sensor nodes. An architecture constraint on accessibility was discussed in Section 2.1.2 on Network Architecture.

2.2.7 Semantic Web Frameworks

There are a number of tools and frameworks available to work with semantic web technologies. We already mentioned Jena, which is a semantic web framework for semantic RDF data. It can use a number of different back ends for RDF data, such as files and databases. There are a number of SPARQL implementations, for Java, Lisp, PHP, Python and C.

Other interesting semantic web applications include D2R, a database to RDF mapping engine [51]. The ontology editors Swoop and Protégé. The Pellet reasoning engine and the Ginseng natural language to SPARQL rewriting application.

Now we have discussed the basics of semantics web technologies and have shown several techniques to combine semantic sensor data, we can start focusing more on the distributed aspect of sensor networks. We will do so by introducing Multi-agent Systems in the Section 2.3, followed by our own work on Question Federation (Section 3.1) and Data Conversion (Section 3.2), with which we complete the picture of the data integration challenge. The next section on Multi-agent Systems is the main building block to address the knowledge distribution and is an appealing approach to building a distributed sensor network architecture as was identified as one of the challenges in Section 1.3.

2.3 Multi Agent Systems

A network of sensors is by definition a distributed system. However, there are different architectures for distributed systems, as was discussed in Section 2.1.2. A multi-agent system is in principle an architecture for distributed systems [52], but can also be used as a centralized or clustered architecture. It treats each node in the sensor network as an entity with communicative and reasoning capabilities, which we call an agent. These two capabilities make agents well suited for addressing our fourth challenge on knowledge distribution as was stated in Section 1.3.

In the domain of sensor networks, an agent could be instantiated on a sensor node. A sensor node is a full fledged computer, with the ability to run an agent. By adding agents to sensor networks each sensor node can be regarded as an entity that can answer questions about sensor measurements and actively share knowledge.

In this section we will first define agents and agent behaviours in Section 2.3.1. Section 2.3.2 discusses how agents can communicate and Section 2.3.3 introduces interaction protocols to enhance the communication between agents and facilitate knowledge sharing. We end this section on Multi-agent Systems with a short overview of the JADE Agent Framework, the most widely used multi-agent framework. In each of these sections examples are given of how agents can be used in the sensor network domain.

2.3.1 Agents

An agent is anything that can be viewed as perceiving its environment through sensors and acting upon that environment through effectors [48]. According to [52] an agent is a computer system that is capable of independent action on behalf of its user. Roughly an agent can be attributed the following capabilities: an agent is **reactive**, which means that it perceives its environment and is able to respond to changes in it. An agent is **proactive**, which means that an agent can take initiative in order to exhibit goal directed behaviour. Finally, an agent has **social ability**, which means that it is able to communicate with other agents to share knowledge or answer questions.

These capabilities make agents well suited for a sensor network environment. In a sensor network, multiple sensors can be connected to a sensor node. If a sensor node runs an agent, then it can use the connected sensors to perceive its environment and react to measured changes accordingly. Moreover, a sensor node has a proactive component by its need to address other sensor nodes in the network in case the agent wants to know something that is known by another sensor node. As was stated in Section 1.3, knowledge distribution is a main challenge for a sensor network. The social ability of agents make them well suited to share knowledge among each other.

In literature, agents are thought to be intelligent entities that show interesting behaviour. However, agents can be very simple; by definition, showing interesting behaviour is not even a prerequisite for agents. In the sensor network domain an agent on a sensor node could have the capability to reason with semantic knowledge about sensor data (Section 2.2.4) to increase its knowledge and it can answer and pose questions about this knowledge.

Agents use behaviours to give them their reactive, proactive and social abilities. A behaviour is an action that is executed as a result of a reaction to a change in the environment, a wish to perform a proactive action or as an answer to a question in a social conversation. The following paragraph explains behaviours and how they can be used to enable the agent to be both reactive and proactive.

Behaviours

Imagine a network of sensors in which different users pose questions about sensor measurements. A sensor node can thus receive multiple questions simultaneously. Some of these questions are direct questions and can be answered right away. Other questions only receive an answer when an event condition is triggered (see Section 2.1.3 on questions and events). An agent can receive many questions simultaneously and some of these cannot be answered strait away. These questions are then put on hold, while others are being handled. An agent is able to execute actions in parallel, giving it the ability to handle multiple simultaneous questions.

An agent is able to handle simultaneous tasks by implementing behaviours. Each behaviour is able to perform a certain task and multiple instances of the same behaviour can run in parallel. A behaviour scheduler decides which behaviour can execute and when it should hold to allow other behaviours to execute.

For a sensor node agent in the sensor network domain we can think of specialized behaviours for answering semantic questions, federating questions, handling answers to federated questions and sending answers about questions to the agent that posed the question. Different behaviours can be thought of. These will be discussed in more detail in Section 4.2.7.

In the next section we will show how communication between agents gives the agents their social ability.

2.3.2 Agent communication

It is important to make a distinction between direct method calls in object oriented programming and communication in agent systems. Agents are independent and distributed, which means they do not have direct access to each other's state and knowledge base. Communication through a common language is the only way to exchange information. This might seem cumbersome in an application designed by a single party. However, the use of agents by multiple parties makes it possible to exchange information without having to know the architectural differences between software paradigms. The only thing needed is agreement on the communication protocol. This makes agent communication well suited for a sensor network environment where different parties maintain the sensors, such as the IJkdijk.

Before we discuss agent communication languages, we introduce the Speech Act Theory of Searle [53]. This theory is based on the assumption that communication is performed by agents just like any other action. This theory provides the foundation for communication between two agents by defining a framework for message exchange between a speaker and a hearer. By classifying communication acts as actions it is possible for an agent to deliberate on communication actions just as it does on other types of actions. This is useful in a sensor network where communication is the most occurring event.

We will now specify how agents can communicate using a communication language. An agent communication language specifies what information should be sent from sender to receiver. For instance, it describes who is the sender and who is the receiver. It also specifies the content of the message and it states the language that this content is written in. Different languages have been proposed, like KIF and KQML, but none of these have become a standard. In 1995 the FIPA has started to develop standards for agent systems [54]. The FIPA-ACL communication language has now become the most widespread protocol for agent communication. It prescribes a number of parameters that can be send with a message between agents. Table 7 gives an overview of these parameters.

Parameter	Category of Parameter
performative	Type of communicative acts
sender	Participant in communication
receiver	Participant in communication
reply-to	Content of message
content	Description of Content
language	Description of Content
encoding	Description of Content
ontology	Description of Content
protocol	Control of conversation
conversation-id	Control of conversation
reply-with	Control of conversation
in-reply-to	Control of conversation
reply-by	Control of conversation

Table 7: Fipa ACL parameters

The FIPA specification does not specify how the content of a message should be created. This depends on the type of information that has to be communicated. By specifying the language in which the content was written, it is possible for agents to parse messages in different languages and thus be able to communicate with a wide range of agents on different topics.

The ontology parameter can be used to specify about which ontology the agent is asking a question. An ontology is characterized by a version number. In Section 2.2.3 we already mentioned how version information can be used to decide whether a question about an ontology is still valid. By sending the expected version number in the ontology parameter, agents can check whether there is a mismatch between the expected ontology and the local ontology of the receiving agent.

The sender and receiver parameters are used to specify which agent has sent the message and who should receive the message. In the next section we will go into more detail on the performative and protocol parameters of the FIPA ACL message.

2.3.3 FIPA Interaction Protocols

In Section 2.1.3 we have mentioned two types of message interaction in a sensor network: questions and events. In a multi-agent system message interaction can be described by a series of actions that constitute a conversation between two agents. Such a conversation is controlled by a protocol.

Protocols define what types of messages agents can expect from another agent with which they are having a conversation, when they can expect to receive these messages and when it is their turn to send a message. Using a protocol makes a conversation more explicit and more efficient. Moreover, agents can use a protocol to deny request from other agents.

Regarding questions and events. We can map a direct question about a sensor measurement onto the request protocol as shown in Table 8 on the left. This protocol uses a request message to initiate a question and expects an inform message with the answer or refuse or failure message in case the question was already received earlier or could not be understood by the participant.

An event maps onto the FIPA Subscribe protocol as shown in Table 8 in the top right. A subscription starts with a subscribe message which contains a event condition to trigger on and a question which should be answered when the event triggers. The subscription can be refused or agreed upon by the participant agent. Each time the event condition triggers due to a local change at the participant, an inform message is sent to the initiator agent. When the initiator agent wants to cancel the subscription it has to send a cancel message, as shown in Table 8 in the bottom right.

We will now introduce the JADE Agent Framework which is the most widely used agent framework for distributed environments.

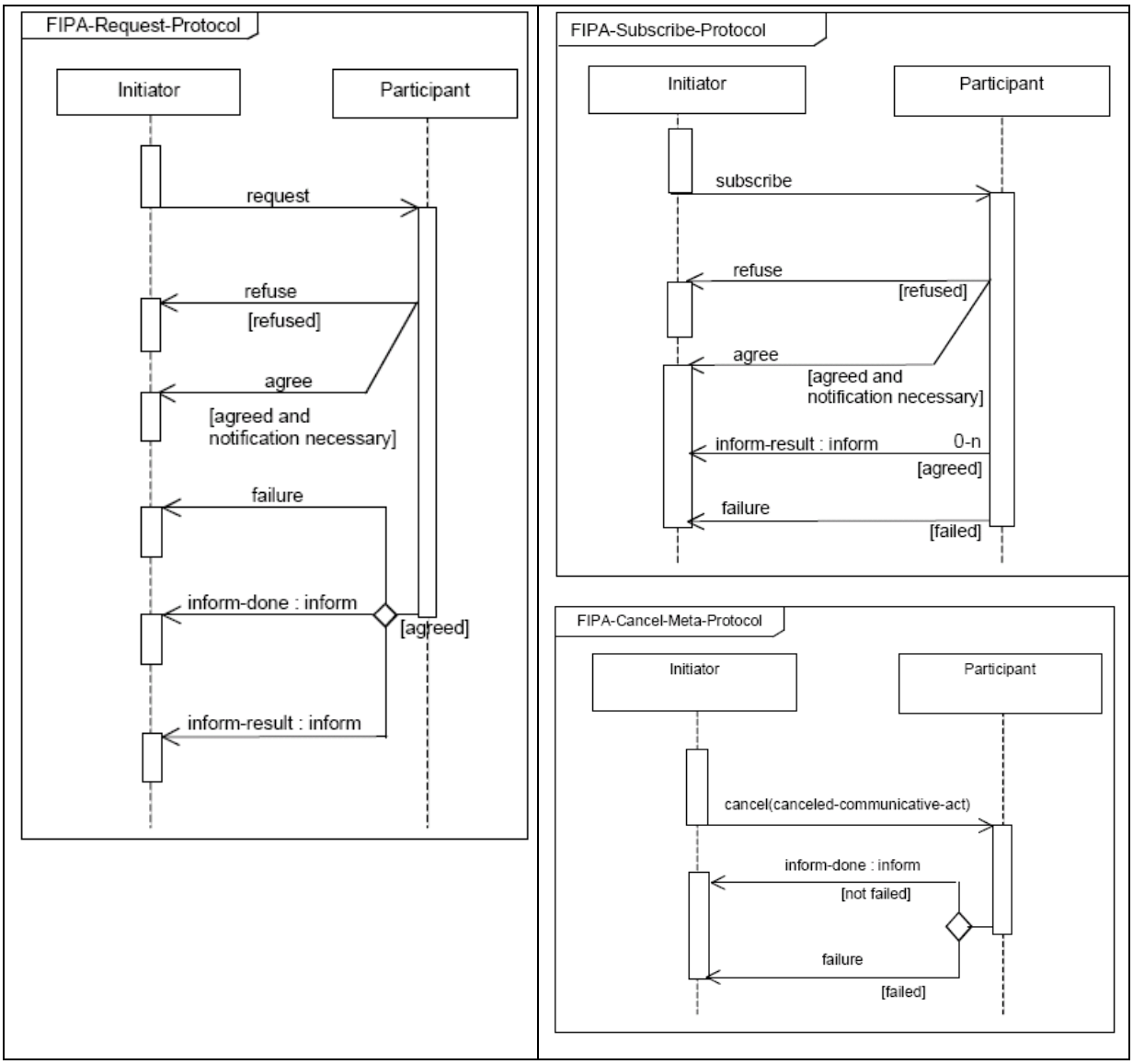


Table 8: FIPA Interaction Protocols

2.3.4 JADE Agent Framework

The JADE Agent Framework [55], is a Java implementation of a behaviour based agent architecture. It implements the FIPA Interaction Protocols described in the previous section. As the Java programming language is cross-platform and since the framework is able to run on the Java Micro Edition CLDC, it is well suited to be used in a sensor network environment, even if the sensor nodes have limited resources. The JADE framework is interoperable with other programming languages than Java. For many languages there exists a bridge with Java to be able to send messages to a JADE agent. An example of such a bridge is the JadeLeap extension that enables the use of JADE from a .NET application.

In this section on Multi-agent Systems we have shown why this paradigm is well suited to be used in the sensor network domain. The fact that agents are autonomous and can handle multiple tasks simultaneously is a big advantage. Interaction protocols make communication between distributed sensor nodes in a sensor network more explicit and can be used to share knowledge between sensor nodes, thus addressing the fourth challenge on knowledge distribution as was stated in Section 1.3.

In the next section we discuss a number of ways to improve upon existing solutions for data integration.

3 Integration of distributed knowledge

In the research background we have shown the different architectures of sensor networks and how these effect communication and knowledge sharing between sensor nodes in a sensor network. The semantic web technologies where shown to be of use when it comes to the integration of sensor data. Semantic annotation with RDF and semantic querying with SPARQL make it possible to describe sensor measurements and extract sensor knowledge from sensors. The multi-agent paradigm proved to be of good use when it comes to knowledge sharing, since it provides useful interaction protocols that can be used by sensor nodes to exchange knowledge and ask questions about each others knowledge.

From the IJkdijk scenario and the state of the art in current semantic sensor network frameworks we identified a number of challenges that have to be addressed in order to successfully integrate distributed sensor knowledge. In this section we will discuss three missing aspects to address these challenges.

First, since data in a sensor network is distributed, the acquisition of knowledge in the network implicates that it should be possible to address multiple sensor nodes to receive an answer to a single question. Question federation is the process of splitting a semantic question and send it to multiple distributed sensor nodes.

Second, an enhancement to existing data integration techniques of the semantic web can be realized when it becomes possible to convert data at sensor nodes from one form to another. Data conversion is the process of converting data at a sensor node when it is not in a form that is required for integrating this data with sensor data from other sensor nodes. The main advantage is that sensor data can be integrated in the network, close to the sensors, resulting in less redundant data transfer through the network. Another advantage is fewer post-processing operations are required by the user or sensor node who posed the question to the network.

Third, knowledge sharing between sensor nodes in a sensor network enables the sensor node to update their knowledge of the environment. This improves the network capability to integrate distributed sensor measurements. An example of this is the sharing of a translation between ontologies (ontology mapping), allowing the sensor nodes to answer questions that are posed in a different language (posed in terms of another ontology).

In the following sections we discuss how to make question federation and data conversion possible with current semantic web techniques. We show what types of knowledge are worth sharing in a sensor network such as the IJkdijk and how sharing this knowledge effects the network.

3.1 Question Federation

In the IJkdijk scenario, which was presented in the introduction section, a number of example questions were given. Some of these questions can only be answered by combining sensor data from multiple sensors. Consider the following question:

What is the pressure at locations where the temperature is over 14 and the pressure over 30?

A possible approach to answer this question in the setting of the IJkdijk sensor network is to first address all temperature sensors and for each temperature sensor where the temperature is over 14, address a pressure sensor that is measuring pressure at the same location as the temperature sensor. The final answers from the pressure sensors are then limited to those where pressure values are over 30.

Such a question about sensor measurements could be asked by a user, or by another sensor node. As the size of the sensor network grows it is not plausible to require a user, or sensor node, to know which sensor nodes to address with sub-questions of the complete question. A requirement for a sensor network framework is thus that it allows a user or sensor to pose a complete question, such as the one above. Based on knowledge in the network this question can then be split into sub-questions, which can be answered by different sensors in the network. The combined answer is then sent to the user or sensor node, without having to known how the question was split into sub-questions in the first

place. The process of splitting a question based on network knowledge is what we call **question federation**. To be able to federate a question in a network a meta description of the knowledge that is maintained by each sensor node has to be available. Based on this meta description, a sensor node can decide where to send the question, or sub-questions. Ontologies, as discussed in Section 2.2.3, are such a meta description of a knowledge base. In the remainder of this section the ontology of a knowledge base is used as a basis for question federation.

Methods for question federation

There are two ways to handle question federation, a question rewriting method was introduced by Dimitrov et. al. [17] and a question splitting is being introduced in this report. Both methods can be used on SPARQL queries as we will show. The **question rewriting** method is to rewrite a question so that it can be understood by different sensor nodes. Question rewriting can be achieved by maintaining an ontology for each sensor node and rewrite questions based on a mapping between the ontologies. A question to a sensor network is posed in terms of the ontology of one of the sensor nodes and a mapping between ontologies describing sensor nodes is used to rewrite the question so that it can be understood by other sensor nodes as well. As an example, consider the ontologies describing knowledge bases of two different sensor nodes in Table 9.

Ontology describing knowledge of sensor node 1	Ontology describing knowledge of sensor node 2
@prefix : <http://ijkdijk.nl/sensornode1#> . @prefix sensor2: <http://ijkdijk.nl/sensornode2#> . @prefix owl: < http://www.w3.org/2002/07/owl#> . @prefix rdfs: < http://www.w3.org/2000/01/rdf-schema#> . :Sensor a owl:Class; owl:equivalentClass sensor2:Sensor . :Measurement a owl:Class ; owl:equivalentClass sensor2:SensorMeasurement . :measurement a owl:ObjectProperty ; owl:equivalentProperty sensor2:reading ; rdfs:domain Sensor ; rdfs:range :Measurement . :value a owl:DatatypeProperty ; owl:equivalentProperty sensor2:value ; rdfs:domain :Measurement . rdfs:range xsd:integer .	@prefix : <http://ijkdijk.nl/sensornode2#> . @prefix sensor1: <http://ijkdijk.nl/sensornode1#> . @prefix owl: < http://www.w3.org/2002/07/owl#> . @prefix rdfs: < http://www.w3.org/2000/01/rdf-schema#> . :Sensor a owl:Class ; owl:equivalentClass sensor1:Sensor . :SensorMeasurement a owl:Class ; owl:equivalentClass sensor1: Measurement . :reading a owl:ObjectProperty ; owl:equivalentProperty sensor1:measurement ; rdfs:domain Sensor ; rdfs:range :SensorMeasurement . :measurement a owl:DatatypeProperty ; owl:equivalentProperty sensor1:value ; rdfs:domain :SensorMeasurement . rdfs:range xsd:integer .

Table 9: ontologies with mapping information, for query rewriting example

A question about the measurements of these two sensor nodes could be:

What are the measured values of the sensors?

This question is translated into a SPARQL query as shown in Table 10 on the left. This question is posed in terms of the ontology of sensor node 1 (Table 9 on the left). The question is rewritten for sensor node 2 based on the mapping with this ontology in the ontology of sensor node 1. The rewritten query is given in the Table below on the right. These questions are sent in parallel to both sensor nodes and the results are combined to make up the final answer.
Dimitrov et.al have incorporated this technique in their distributed information integration system [17]. They use the MiniCon Algorithm [56], which uses ontology mappings for rewriting SPARQL queries. The rewritten queries are executed by the different sensor nodes.

The main advantage of question rewriting is that it becomes possible to ask questions about knowledge that is stored with a different cardinality at different sensor nodes. If one sensor node for instance stores a value in two semantic properties and the other sensor node in just a single field. By

rewriting the query it becomes possible to address both sensor nodes and still combine the results. The problem with question rewriting as the question federation method is that it only allows for querying distributed sources which contain the same kind of information. In a sensor network this would require all the sensor nodes to have knowledge about the same type of sensors. However, in practice, there are many different sensors in the network. Any number of sensors and sensor types can be connected to a sensor node. This results in a heterogeneous sensor network, in which question federation through question rewriting is not suitable.

SPARQL query for sensor node 1	Rewritten SPARQL query for sensor node 2

PREFIX sensor1: <http://ijkdijk.nl/sensornode1#>
 	
SELECT
?value
WHERE
{
?sensor a sensor1:Sensor ;
sensor1:measurement ?m .
?m sensor1:value ?value .
}
	PREFIX sensor2: <http://ijkdijk.nl/sensornode2#>
	
	SELECT
	?value
	WHERE
	{
	?sensor a sensor2:Sensor ;
	sensor2:reading ?m .
	?m sensor2:measurement ?value .
	}

Table 10: example of question rewriting

The second method for question federation is **question splitting**. This method splits a semantic question into multiple sub-questions that can be answered by different distributed sensor nodes. The question splitting is also based on ontologies. Each sensor node is required to describe its data with an ontology. These ontologies have to be shared with the other sensor nodes. Each time a question is received, the ontologies of neighbouring sensor nodes are used to determine which of these nodes can answer a part of the question.

In the rest of this section we propose a question splitting algorithm that uses ontological metadata to determine how to split a semantic SPARQL question into multiple sub-questions and how the answers to the sub-questions are recombined. We start by showing the different ways to send sub-questions to distributed sources and we clarify this with some examples. Our question splitting algorithm is the first to use ontological metadata to split semantic SPARQL questions. It is inspired by Darq [57], which is an implementation for the JENA framework that allows to send queries to distributed data sources based on metadata extracted from each source.

3.1.1 Question federation, a formal explanation of splitting a question

There are two ways to send sub-questions to sensor nodes. If multiple sources have knowledge about the same kind of semantic data, then each of these sources has to be sent the sub-question. We will call this **parallel federation** of a sub-question. If a question is split into multiple sub-questions, but one sub-question has to be answered before the next can be sent, then we use **serial federation**.

The following example will clarify the terms we have just introduced. Take a look at Figure 6. This figure shows a simplified version of the IJkdijk scenario, describing a sensor network with two temperature sensors and a single pressure sensor. In the top left you see a sensor network architecture which contains four sensor nodes, four temperature sensors, one pressure sensor and a user who asks questions to the network. On the right side of the figure there is an explanation of the knowledge that is maintained by each sensor node. In the bottom there is a legend of the items used.

This network contains knowledge about temperature measurements on along an embankment, where two locations are identified: zone A and zone B. Furthermore, there is a node in the network that knows about the measured pressure of both zone A and zone B. Each of the sensor nodes has a knowledge base, which contains sensor readings.

Take a look at Table 11. On the left hand side is a possible semantic knowledge base for sensor node R1 or R2. It contains semantic data about sensor measurements for two temperature sensors. On the right hand side is a semantic knowledge base of data that could be stored at sensor node 3. A user now wants to know from the network the answer to the following question:

Wh*at is the pressure at locations where the temperature is over 14 and the pressure over 30?*

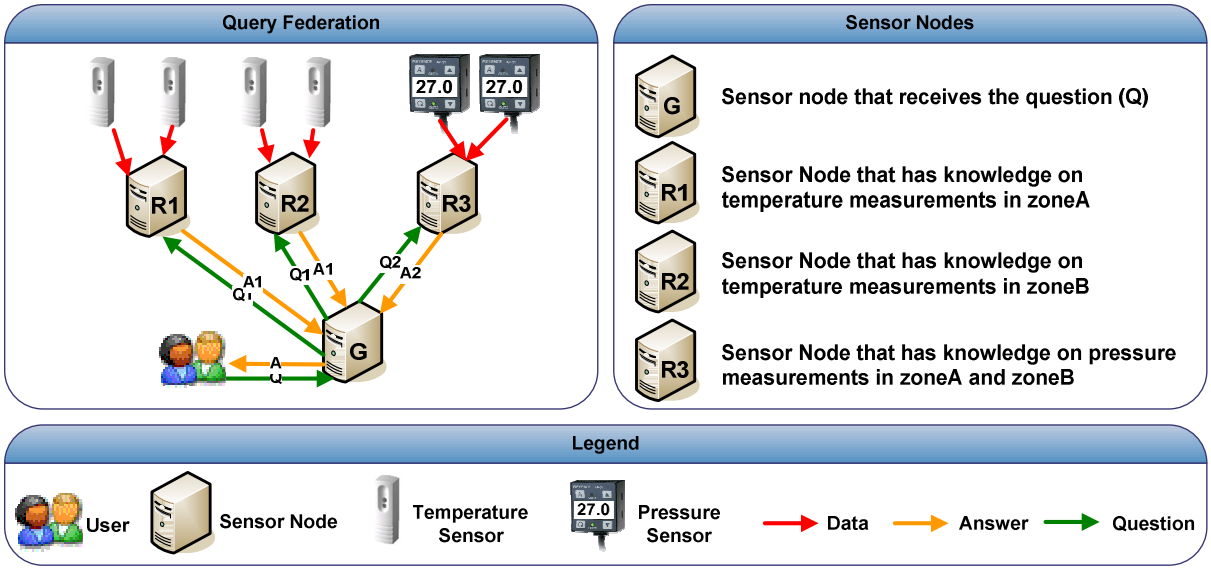


Figure 6: Question Federation example

Knowledge base of sensor node R1 or R2	Knowledge base of sensor node R3
@prefix : <http://ijkdijk.nl/service1#> . @prefix sensors: <http://ijkdijk.nl/sensors.owl#> . @prefix locations: <http://ijkdijk.nl/locations.owl#> . :Sensor_1 a sensors:TemperatureSensor ; sensors:location locations:zoneA ; sensors:measurement [a sensors:Measurement ; sensors:value “25”] . :Sensor_2 a sensors:TemperatureSensor ; sensors:location locations:zoneB ; sensors:measurement [a sensors:Measurement ; sensors:value “23”] .	@prefix : <http://ijkdijk.nl/service3#> . @prefix sensors: <http://ijkdijk.nl/sensors.owl#> . @prefix locations: <http://ijkdijk.nl/locations.owl#> . :Sensor_1 a sensors:PressureSensor ; sensors:location locations:zoneA ; sensors:measurement [a sensors:Measurement ; sensors:value “32”] . :Sensor_2 a sensors:PressureSensor ; sensors:location locations:zoneB ; sensors:measurement [a sensors:Measurement ; sensors:value “36”] .

Table 11: Service Knowledge bases for query federation example

Given the knowledge bases in Table 11 we can create a SPARQL query, such as the one in Table 12, which is a semantic equivalent of the question above. This question is sent to a sensor node in the network. In the example in Figure 6 this is sensor node G.

```
PREFIX sensors: <http://ijkdijk.nl/sensors.owl#>
PREFIX locations: <http://ijkdijk.nl/locations.owl#>

SELECT
  ?location  ?temperature ?pressure
WHERE
{
  ?tsensor  a sensors:TemperatureSensor ;
      sensors:location ?location ;
      sensors:measurement ?m .
  ?m sensors:value ?temperature .
  FILTER(?temperature > 14)
  ?psensor  a sensors:PressureSensor ;
      sensors:location ?location ;
      sensors:measurement ?m2 .
  ?m2 sensors:value ?pressure .
  FILTER(?pressure > 30)
}
```

What is the location of temperature sensors with a measurement over 14

What is the pressure measured by pressure sensors at these locations if it is over 30

Table 12: Query federation example query

Sensor node G now has to retrieve the answer to this question. As we already know, none of the sensor nodes know the complete answer to this question, since the knowledge to answer this question is distributed in the network. The question thus has to be split into sub-questions which should be sent to the correct sensor nodes.

At the moment, sensor node G is completely unaware of the knowledge its neighbouring sensor nodes have. To be able to send questions to the correct sensor nodes, sensor node G thus has to know something about the type of knowledge that is stored at each sensor node. Sensor node G needs to have a meta description of the knowledge available at other sensor nodes. An ontology can provide such a meta description of a semantic knowledge base. Let us assume that each sensor node has an ontology describing its knowledge base. Ontologies for the knowledge bases in Table 11 are shown in Table 13. Such ontologies describe classes and their properties as we discussed in Section 2.2.3. Sensor nodes can share their ontologies with other sensor nodes through the principals discussed in 2.3.1 on knowledge sharing with agents.

Ontology of sensor nodes R1 & R2	Ontology of sensor node R3
@prefix : <http://ijkdijk.nl/service1#> . @prefix sensors: <http://ijkdijk.nl/sensors.owl#> . @prefix locations: <http://ijkdijk.nl/locations.owl#> . @prefix owl: < http://www.w3.org/2002/07/owl#> . @prefix rdfs: < http://www.w3.org/2000/01/rdf-schema#> . sensors:TemperatureSensor a owl:Class . sensors:Measurement a owl:Class . sensors:measurement a owl:ObjectProperty ; rdfs:domain sensors:TemperatureSensor ; rdfs:range sensors:Measurement . sensors:value a owl:DatatypeProperty ; rdfs:domain sensors:Measurement . sensors:location a owl:ObjectProperty ; rdfs:domain sensors:TemperatureSensor ; rdfs:range locations:Location . :Sensor_1 a sensors:TemperatureSensor ; sensors:location locations:zoneA . :Sensor_2 a sensors:TemperatureSensor ; sensors:location locations:zoneA .	@prefix : <http://ijkdijk.nl/service3#> . @prefix sensors: <http://ijkdijk.nl/sensors.owl#> . @prefix locations: <http://ijkdijk.nl/locations.owl#> . @prefix owl: < http://www.w3.org/2002/07/owl#> . @prefix rdfs: < http://www.w3.org/2000/01/rdf-schema#> . sensors:PressureSensor a owl:Class . sensors:Measurement a owl:Class . sensors:measurement a owl:ObjectProperty ; rdfs:domain sensors:PressureSensor ; rdfs:range sensors:Measurement . sensors:value a owl:DatatypeProperty ; rdfs:domain sensors:Measurement . sensors:location a owl:ObjectProperty ; rdfs:domain sensors:PressureSensor ; rdfs:range locations:Location . :Sensor_1 a sensors:PressureSensor ; sensors:location locations:zoneA . :Sensor_2 a sensors:PressureSensor ; sensors:location locations:zoneB .

Table 13: Sensor node ontologies for query federation example

If all the sensor nodes in Figure 6 share their ontology with each other, then sensor node G now has the ontologies of all the other sensor nodes. Consequently it now knows about what knowledge is available at each sensor node and it can use this to split the question it received from the user (see Table 12). In the next paragraph we discuss how a SPARQL question can be split by using ontology metadata.

Question Splitting Procedure

In this section we will explain how ontology meta data can help us to gain insight where to send parts of a question. Take a look again at the SPARQL question in Table 12. From what we know about the knowledge in the network, the question can be split into two parts. The first part is a question about temperature measurements (marked green), the other is a question about pressure measurements (marked red). We can also see that these two parts are linked by the ?location variable. To federate such a query we should first send a question to the sensor nodes that know something about temperature sensors and only when we know the locations of these sensors we can send the second part about the pressure at that location. Table 14 shows how these two parts can be formulated as SPARQL queries in a sub-question.

Sub-question to sensor nodes R1 & R2	Sub-question to sensor node R3

PREFIX sensors: <http://ijkdijk.nl/sensors.owl#>
PREFIX locations: <http://ijkdijk.nl/locations.owl#>
 	
SELECT
?temperature
?location
WHERE
{
?tsensor a sensors:TemperatureSensor ;
sensors:location ?location ;
sensors:measurement ?m .
?m sensors:value ?temperature .
FILTER(?temperature > 14)
}
	PREFIX sensors: <http://ijkdijk.nl/sensors.owl#>
	PREFIX locations: <http://ijkdijk.nl/locations.owl#>
	
	SELECT
	?location
	?pressure
	WHERE
	{
	?psensor a sensors:PressureSensor ;
	sensors:location ?location ;
	sensors:measurement ?m2 .
	?m2 sensors:value ?pressure .
	FILTER(?pressure > 30)
	}

Table 14: Sub-questions

So how are the sub-questions distributed in the network? Given the fact that both sensor nodes R1 and R2 describe temperature sensor and its properties in their ontology, the first sub-question, marked green, should be sent in parallel to both temperature sensor nodes R1 and R2. This parallel federation is shown in Figure 6 with the green arrows Q1. Both R1 and R2 now execute this question on the local knowledge base and return the answers through connection A1, indicated by the orange arrow. The answers from R1 and R2 are combined into a single result at sensor node G. For each location in the result of question Q1, sensor node G sends a sub-question to sensor node R3, with the ?location variable substituted by the location from the result of the temperature sensors. Table 14 on the right shows such a question where ?room could be substituted by locations:zoneA or locations:zoneB.

The second sub-question (Q2) about the pressure at these locations is now sent to sensor node R3, since this node describes pressure sensors and measurements in its ontology. Node R3 executes the question on the local knowledge base and returns the answer A2. This answer is combined with the results from the other sub-questions by sensor node G, who now finalizes the answer A to question Q.

3.1.2 Question splitting, a systematic approach

We will now explain how to use the ontologies from sensor nodes to go from a question as in Table 12 to the sub-questions in Table 14. A SPARQL question like the one in Table 12 consists of a number of subject – predicate – object triples; on each line a new triple. In the following it is assumed that these triples are evaluated one by one, from top to bottom, to decide which sensor nodes can possibly answer them. The evaluation of a triple starts with the identification of the predicate and after that, the subject and object are used to restrict the possible destination sensor nodes.

There are a number of ways to analyze triples and to group them together to be sent to a specific sensor node:

- class and property availability
- domain and range restrictions on properties
- shared variables in the semantic question
- static ontology metadata, or: **individuals**

We will discuss each of these in the following paragraphs.

Class and property availability

When evaluating a triple, we can look at the subject, predicate and object of the triple. The predicate is the easiest to start with, since it links the subject and the object and thus has the most specific semantic meaning in a triple. If the predicate is an rdf:type, then this means that the object of this triple is a class name. We thus have to check which of the ontologies of the sensor nodes describe this class. If the predicate is not an rdf:type, then we have to check the ontologies to see which of them contain the

predicate as a property of a class. We will illustrate this with a simple example. Consider the first two triples in the question in Table 12:

```
?sensor  a sensors:TemperatureSensor .
?sensor sensors:location ?location .
```

In the first triple the predicate is "a" which is the equivalent of rdf:type. We thus look in each of the ontologies of sensor nodes R1, R2 and R3 for the owl:Class with resource sensors:TemperatureSensor. In Table 13, we see that both sensor nodes R1 and R2 describe such a class, but node R3 does not. The second triple has the predicate sensors:location. We can see that both R1 and R2 describe this resource as an owl:ObjectProperty in the ontology, and R3 as well. To summarize, based on class and properties the first triple can be sent to both R1 and R2 and the second triple to R1, R2 and R3. The rest of the triples are evaluated just like this. Some additional constraints apply when determining to which sensor nodes each triple should be sent, as will be discussed next.

Domain and range restrictions on properties

In Table 13 we have made explicit the matches between domain restrictions (purple) and range restrictions (orange). These restrictions give information about which properties belong to which classes. This makes it possible to group triples in a question by using the domain and range information from the ontology. To give a simple example, a triple with the predicate sensors:measurement should be sent to the same sensor node as a triple about sensors:TemperatureSensor because it has a domain restriction on the sensors:TemperatureSensor class. We also know that a property sensors:value should be sent to the same sensor node as a triple containing a sensors:Measurement class. In the ontology we can see that the sensors:measurement triple can only contain objects of class sensors:Measurement. From this we learn that a triple with predicate sensors:value should be sent to the same sensor node as a triple with predicate sensors:measurement. The domain and range restrictions can be used to limit the number of services to which a triple should be sent.

Shared variables in the semantic question

A semantic SPARQL question is created by a user. During the creation, the user has added useful information, in the form of variables (such as ?sensor), to the question that can be used to split the question into sub-questions. A SPARQL query contains a number of triples. In a triple, variables can occur in the subject and object position, but not in the predicate position. These variables can bind to variables in other triples of the SPARQL query. Using the same variables in different triples indicates that these triples are related somehow. As an example. Consider these triples again:

```
?sensor  a sensors:TemperatureSensor .
?sensor sensors:location ?location .
```

The ?sensor variable binds the first triple to the second triple. This indicates that the sensor location relates to the temperature sensor, and it is more likely that these triples can be sent to (and answered by) a the same sensor node in the network. Consider the following two triples:

```
?tsensor  a sensors:TemperatureSensor .
?psensor a sensors:PressureSensor .
```

These two triples do not share variables, making it thus less likely that they can be sent to the same sensor node. We use shared variables only as an indication of relation, since variables can bind sub-questions as well (meaning that the triples of sub-questions are sent to different sensor nodes). We will show later in the discussion on the algorithm how shared variables together with domain and range restrictions are used to group triples into sub-questions.

Static ontology metadata, or: individuals

The ontologies in Table 13 contain some extra information about the sensors at the sensors nodes, namely a sensors:location property. The meta information about these sensors is static information and will not change over time. By adding this to the ontology we also restrict the subject and object resources of a triple.

The example question that is considered here does not contain any static information. It would if the location of the sensors was limited to zoneA for instance. Consider the following triple being added to the question in Table 12:

```
?sensor  sensors:location locations:zoneA ;
```

The object resource of this triple is bound to locations:zoneA. We can use this bound object to limit the set of sensor nodes to which this triple should be sent. Only those ontologies that contain an individual with a property sensors:location that is bound to locations:zoneA will be sent this triple. In this case all R1, R2, and R3.

Property Functions, Filters, Order by clauses and Limits

Some of the elements of a SPARQL query are not normal triples or not triples at all. Take a property function. Its predicate is meaningless since it is merely the name of a programming method to change the result of a query. Other examples are filters, order by clauses and limits, which do not have the form of a triple. When it comes to splitting questions and federating the sub-questions to sensor nodes the following decisions were made. If a sub-query contains a variable which is also used in a property function, filter or order by clause, then these elements of the original SPARQL query are added to the sub-question. The limit element is always sent to the same sensor node as the order by clause. Consider the following example of an order by clause:

ORDER BY DESC(?pressure) LIMIT 1

If added to a SPARQL query, it orders the results from the query by a descending pressure value and returns only the first result (meaning the highest pressure). This order by clause will be sent to every sensor node to which a triple will also be sent that contains the ?pressure variable. As an example, take a look at the sub-questions in Table 14. The sub-question on the left, about temperature measurements does not make a reference to a pressure measurement (none of the triples has the ?pressure variable in either the subject or object position). The sub-question on the right about pressure measurements does make a reference to the ?pressure variable. The order by statement above is thus appended to this sub-question and not to the sub-question about temperature measurements.

Question Splitting Algorithm

The four ways to restrict where a triple is sent are incorporated into the SplitQuery algorithm, which is shown in Appendix A. This algorithm gets a list of triples from a SPARQL query as input, together with the ontologies describing the sensor nodes in the network. It outputs a list of sub-questions in the form of SPARQL queries and the sensor nodes to which they should be sent. With this, the sub-questions can be federated to the correct sensor nodes. An example how this algorithm is used to analyze and split a semantic SPARQL question is also shown in Appendix A. Figure 6 showed the difference between serial and parallel federation of sub-questions. If a question is split into multiple sub-questions then these are federated serially, in which the first sub-question from the algorithm is sent first, and only after the results of that question are received, the next sub-question is sent. However, each sub-question that has multiple sensor nodes as a target is sent in parallel to these targets. This makes parallel and serial federation possible.

Recombining sub-question results

We have shown that questions can be sent both in serial and in parallel to multiple sensor nodes. The way results are combined is different between the two. Parallel questions are identical. The variables from the SPARQL sub-questions are thus the same between these questions. The results to these questions are concatenated as a SQL like union.

A next sub-question is sent and some of the variables in this question might have been bound by results from the previous sub-question. The results of this sub-question apply to all the results from the previous question. The results are thus not simply concatenated. For each result of the previous sub-question and each result of the current question a new result is created in which the two sub-results are combined.

As an example, consider the sub-questions in Table 14. The question on the left is sent in parallel to R1 and R2. Their knowledge bases are those used throughout this section and can be found in Table 11. Reasoning was performed on these knowledge bases with the ontologies from Table 13, enabling the questioning of both sensor measurements and metadata from the ontology, such as location. The result received from R1 and R2 is concatenated as shown below:

location	temperature
locations:zoneA	25
locations:zoneB	23

The results from R1 and R2 are thus concatenated as a union. The next sub-question, shown in Table 14 on the right is sent to R3. This question is sent twice, with the ?location variable substituted by the location values of the results of the previous sub-questions: locations:zoneA and locations:zoneB. Two results are received serially from R3. Below on the left is shown the result about the pressure at zoneA and on the right the result of the pressure at zoneB.

location	pressure
locations:zoneA	32

location	pressure
locations:zoneB	36

These results are now combined into a result as shown in the table below. This makes up the final answer to the question in Table 12.

location	temperature	pressure
locations:zoneA	25	32
locations:zoneB	23	36

Before moving on to the next section on data conversion we want to make a few notes on question splitting using the method we have proposed in this section. This approach to question splitting assumes that if a sensor node describes in its ontology to have knowledge about a class and its properties, then the knowledge base should be complete, meaning that all the properties of a class need to have a value in the knowledge base. This is due to the fact that the ontology metadata is used to split and federate questions. The ontology gives information about the knowledge that is present at a sensor node in the network. If a question is sent to a sensor node and it turns out that the sensor nodes knowledge base is missing some properties or classes (which are described in the ontology of this knowledge base), then the SPARQL question to this sensor node will return an empty result. The Semantic Web SPARQL language assumes that if a triple can not be found in an RDF dataset, that the triple should return an empty result set. Questions that are split into sub-questions thus behave in the same way. If a sub-question returns an empty answer, and the other sub-questions depend on this question, then the complete SPARQL question returns an empty result. Completeness is an acceptable assumption in the Database Management Systems world. Also, tools such as d2r, a database to RDF mapper work under this assumption. We do not feel that it is a weak point of the SPARQL specification. However, it does implicate that ontology and knowledge base have to be complete and correct if question splitting is used.

In this section we have discussed the different methods to question federation. We have proposed a new algorithm to federate questions based on question splitting, instead of question rewriting. For this to work we have used ontology meta data to describe distributed sensor nodes. We showed how a question is federated and how the results are combined. With this we addressed the third challenge on question federation as was stated in Section 1.3. In the experiments section (Section 5) we evaluate the question splitting algorithm and point out the main advantages and shortcomings of this approach.

The next section will discuss our work on data conversion, which is one of the building blocks of addressing the second challenge on data integration, as was stated in Section 1.3.

3.2 Data Conversion

In Section 1.3 we identified data integration as one of the main challenges for creating a distributed semantic sensor network. Two important elements of the solution for the data integration challenge are the addition of semantics to data and the ability to convert between data with the same meaning but of a different form. Adding semantics to data makes merging of data possible, by comparing semantic meaning of data. Data conversion is the process of converting one data instance into another [28, 29] and allows the conversion between data of the same semantic meaning, but in a different form.

Consider two sensor nodes that both collect temperature measurements of temperature sensors. Sensor node 1 stores the temperature values in a degrees Celsius form and sensor node 2 stores the values in a degrees Fahrenheit form. Both knowledge bases of sensor nodes 1 and 2 are shown in Table 15. The form, or unit of the sensor value is annotated with a ^^ character in the N3 format of RDF. It denotes that the resource has a specific datatype. For sensor node 1 this datatype is units:DegreesCelsius and for sensor node 2 the datatype is units:DegreesFahrenheit.

Knowledge base of sensor node 1	Knowledge base of sensor node 2
@prefix : <http://ijkdijk.nl/sensornode1#> . @prefix sensors: <http://ijkdijk.nl/sensors.owl#> . @prefix units: <http://ijkdijk.nl/units.owl#> . :Sensor_1 a sensors:TemperatureSensor ; sensors:measurement [a sensors:Measurement ; sensors:value “25” ^^ units:DegreesCelsius] . :Sensor_2 a sensors:TemperatureSensor ; sensors:measurement [a sensors:Measurement ; sensors:value “15” ^^units:DegreesCelsius] .	@prefix : <http://ijkdijk.nl/sensornode2#> . @prefix sensors: <http://ijkdijk.nl/sensors.owl#> . @prefix units: <http://ijkdijk.nl/units.owl#> . :Sensor_3 a sensors:TemperatureSensor ; sensors:measurement [a sensors:Measurement ; sensors:value “77” ^^ units:DegreesFahrenheit] . :Sensor_4 a sensors:TemperatureSensor ; sensors:measurement [a sensors:Measurement ; sensors:value “59” ^^units:DegreesFahrenheit]

Table 15: Data conversion example

The semantics of the knowledge bases at both sensor nodes allow us to integrate the sensor measurements of both nodes. We could ask a question to give all sensor values of all temperature sensors. In the previous section on question federation (Section 3) we have shown how to collect all these measurements by splitting the question and use parallel and serial federation to integrate data from distributed sources. However, even though the sensor values from the sensor nodes are temperature values, each is stored in a different form. Combining these values in a single answer to a semantic question gives unexpected results. Therefore, the sensor values from both nodes should be converted to the same form.

The remainder of this section describes how unit / form information can be annotated in a semantic knowledge base. We will then explain how we have chosen to describe a conversion between different forms. Finally we show how the conversion can be used within a semantic question.

Unit annotation

If we want to convert one data instance to another we need to know the form, or unit, in which they are stored in the semantic knowledge base. This requires semantic annotation of unit. There are two approaches to annotate unit. One could add a new property for each value for which data conversion should be possible. This is shown on the left in Table 16. The main advantage is that this approach is

intuitive, since the unit is a property in the knowledge base and ontology. Another approach is to add datatype information to the value directly[1]. This is shown on the right in Table 16.

Unit annotation as property	Unit annotation as datatype
@prefix : <http://ijkdijk.nl/sensornode1#> . @prefix sensors: <http://ijkdijk.nl/sensors.owl#> . @prefix units: <http://ijkdijk.nl/units.owl#> . :Sensor_1 a sensors:TemperatureSensor ; sensors:measurement [a sensors:Measurement ; sensors:value “25”; sensors:unit units:DegreesCelsius] .	@prefix : <http://ijkdijk.nl/sensornode2#> . @prefix sensors: <http://ijkdijk.nl/sensors.owl#> . @prefix units: <http://ijkdijk.nl/units.owl#> . :Sensor_3 a sensors:TemperatureSensor ; sensors:measurement [a sensors:Measurement ; sensors:value “77” ^^ units:DegreesFahrenheit] .

Table 16: Unit annotation

A datatype describes the kind of data that is represented by a resource. Resources can be typed as integer values, string values, but also more specific types by using user-defined datatypes, for example, seconds, meters, inch, etcetera. The main advantage is that this approach allows to easily annotate each resource without having to introduce new properties into the ontology for each value that should have a unit property. Another advantage is that datatyping in RDF is a standard approach for adding type information to data, whereas adding unit as a property requires each resource to use the same semantic annotation for these unit properties. Datatyping is thus the better approach when multiple parties annotate sensor data in a network differently. In the following we will annotate units as a datatype. In the next paragraph we will show how conversion between units can be handled.

Conversion Description

To convert one data form to another we need a way to describe the conversion. Such a conversion takes as input a value of a specific form and outputs a converted value in a different form. In the example in Table 16, if we want to convert all measurements to a degrees Fahrenheit unit, then we would convert the value “25” to its equivalent in Fahrenheit, which is “77”.

A conversion is a form of semantic knowledge. In a distributed sensor network domain it can be useful to share conversions function between different sensor nodes. This allows for the introduction of new datatypes and conversion functions even after the network in its original form was started, thus allowing the network to be usable over a longer period. To allow for knowledge distribution we need to describe the conversion function in an exchangeable format. All the knowledge discussed so far was in the RDF format, which is an open format that can be interpreted by many different engines. We should thus look for a way to describe a conversion function in a similar way. There are two appealing approaches to describe the conversion between datatypes in an exchangeable format. An intuitive method to describe mathematical conversion is MathML. MathML is a representation scheme for mathematical functions in XML format. Although MathML is based on similar principles as RDF, allowing binding of variables both internal to the document and by URI, it still lacks proper interoperability with RDF variables, which makes it difficult to embed in a semantic knowledge base. Another disadvantage of MathML is that it is more a representation language then it is executable, although a few execution engines exist, non of these make it easy to combine it with other semantic web techniques. Finally, probably the most compelling reason not to use MathML is that it only allows to describe a mathematical conversion and not any other conversion, such as string conversions.

Another, less obvious method to exchange conversion knowledge is by using a very expressive scripting language which is interpretable by different programming environments. One such scripting language is JavaScript. For many different programming languages there are interpreters for JavaScript, which makes it possible to add conversion functions on the fly, without having to recompile the sensor node. JavaScript however does not allow for semantic annotation.

[1] See RDF Datatyping (http://infolab.stanford.edu/~melnik/rdf/datatyping/) and XSD schema datatypes (http://www.w3.org/TR/swbp-xsch-datatypes/)

```
@prefix units:   <http://ijkdijk.nl/units.owl#> .
@prefix conversions:   <http://ijkdijk.nl/conversions.owl#> .

[] a conversions:Conversion ;
        conversions:from units:DegreesFahrenheit ;
        conversions:to units:DegreesCelsius;
        conversions:function "

            function convert(mapping){
              if(mapping.inputValues.length == 1){
                 var degreesFahrenheit = mapping.inputValues[0];
                 degreesCelsius = (degreesFahrenheit - 32) * 5 / 9;
                 mapping.setOutput(mapping.outputUnits[0], degreesCelsius;
              }

              return mapping;
            }

        " .
```

Table 17: Annotated conversion function

By embedding JavaScript in an RDF format we can describe the input and output datatypes in a semantic way and still be able to distribute the conversion function in an open format. An example of this is shown in Table 17. The JavaScript conversion function is embedded in an semantic Conversion, which has a from and to property. This Conversion resource can be described by a shared ontology and thus used to exchange conversion functions between sensor nodes in a network. This function describes a conversion from degrees Fahrenheit to degrees Celsius. The convert function in the JavaScript code receives the JSON object: mapping. A JSON object is an interchangeable JavaScript object which can be interpreted by other programming languages such as Java. The mapping object contains the input value and its datatype.

The conversion function then adds the output value for the given output datatype to the mapping object. By using a semantically annotated JavaScript conversion function we are flexible in describing the conversion from one datatype to another. Due to the fact that the whole conversion is in an exchangeable format, this is a well suited approach to unit conversion in a distributed environment.

There is an issue regarding the precision of data conversion methods. If a degrees Fahrenheit value is converted to a degrees Celsius value, the conversion function will have to decide how to round the Celsius value. Precision of converted values makes it difficult to compare these values with other values. As an example: are the following values equal: 25.0 and 25.0001? At first sight one would say the values are different, however, it all depends on how we round the values. We could say these values are almost identical, but not exactly.

Conversion Execution

Thus far we have only mentioned how to describe a conversion between two datatypes with an annotated JavaScript function. We will now explain exactly how and when the conversion is executed when we ask a semantic question to a distributed sensor network. Take a look at the semantic SPARQL question in Table 18. This question asks for all temperature sensors and their sensor values, in the degrees Celsius form. We implemented the conversion execution as a SPARQL property function (See 2.2.5). The input to the function is the value of the variable ?v, the output is a value bound to the variable ?value.

This question is executed on a semantic knowledge base such as the one in Table 15. The datatype of the sensor value ?v can be found in the knowledge base. If this datatype is already in a degrees Celsius form, then the ?value content is exactly the same as the ?v content. If the datatype is different, then the conversion execution function searches in the local knowledge base for a conversion function that converts from the datatype of ?v to a degrees Celsius datatype, by using the “from” and “to” properties as shown in Table 17. A conversion function such as the one in Table 17 will be found and the JavaScript function is executed on the value bound to ?v. The resulting output will be bound to the variable ?value in the question.

```
PREFIX sensors: <http://ijkdijk.nl/sensors.owl#>
PREFIX locations: <http://ijkdijk.nl/locations.owl#>
PREFIX pf: <java:com.hp.hpl.jena.query.darq.pfunction.library.>

SELECT
  ?sensor
  ?value
WHERE
{
  ?sensor  a sensors:TemperatureSensor ;
      sensors:measurement ?m .
  ?m sensors:value ?v .
  ?v pf:convert (units:DegreesCelsius ?value) .
}
```

Table 18: Example SPARQL conversion question

3.3 Knowledge sharing

The IJkdijk, or any other sensor network, is susceptible to changes in the environment or network topology. Such changes include the addition or removal of sensors to sensor nodes, a displacement of sensors by a human operator or environmental events, or a change in semantics with which a sensor node, or a group of sensor nodes, describes its sensor measurements. To be able to deal with these changes in a dynamic environment, the sensor nodes require the ability to exchange knowledge about these events. Sharing knowledge enables sensor nodes in the network to stay up to date about the environment and the knowledge of other sensor nodes.

Knowledge sharing is used in different domains, such as peer-to-peer networks and multi-agent systems to enable a decentralized sharing and administration of knowledge in a distributed environment [3, 5]. Castano et. al. share ontologies to facilitate communication among peer nodes in a peer to peer network. We will show that there are other types of knowledge that are worth sharing in a distributed sensor network, such as the IJkdijk.

Thus far we have seen different types of knowledge in a sensor network. Both private ontologies of sensor nodes and shared ontologies among sensor nodes tell the sensor node something about knowledge infrastructure and hierarchical relationships of semantic concepts in the environment. These ontologies can be used to successfully split and distribute questions to sensor nodes. Both rules and data conversion functions can be used by a sensor node to further improve its data integration capabilities. These types of knowledge can be shared among sensor nodes, enabling them to increase their understanding of the environment and their ability to answer questions about the sensor measurements in the network.

In the experiments section later on we show how the behaviour and capabilities of a sensor network change as a result of introducing and sharing new knowledge in the network.

With this section we finalize our work on the building blocks of a distributed semantic sensor network. We have addressed all the challenges as were stated in Section 1.3, which we now briefly summarize:

1. Construction of a distributed network architecture; how to facilitate communication among sensor nodes in a distributed environment
2. Facilitate the integration of sensor data; how to enable data integration by adding semantic web technologies to the sensor network domain and improve upon existing integration solutions
3. Question federation; how to split and federate a semantic question to enable the integration of distributed sensor data
4. Sharing of and reasoning with semantic knowledge; how can knowledge sharing allow a sensor network to operate in a dynamic environment and how reasoning with shared knowledge can increase knowledge at sensor nodes

We have shown that the Multi-agent framework is well suited as a sensor network architecture, because it is based on the assumption that agents are distributed entities. Such an agent can represent a sensor node in the network and answer questions about semantic data stored at the sensor node.

The data integration challenge was split into two. First, we explained how semantic annotation in RDF format of data can help to compare and combine sensor data. Furthermore, we have shown that data conversion can help to merge data that has the same semantic meaning, but is stored in a different form or unit at the sensor node.

We have explained how a semantic SPARQL question can be used to query semantic RDF data. This is necessary to extract and combine the knowledge contained in a semantic knowledge base. With our work on question federation we have discussed how a question can be split and federated to multiple sensor nodes. This work is essential to be able to combine semantic data which is distributed over multiple sources.

Finally we have shown in a number of ways that the distribution of semantic knowledge, such as ontologies, conversion functions and semantic rules, can be useful to increase the capabilities of the sensor network. A Multi-Agent framework is well suited to be used for knowledge exchange since it incorporates a number of useful protocols to manage conversations between agents at sensor nodes.

In the upcoming section a framework is proposed that implements the requirements for a distributed semantic sensor network in which knowledge can be exchanged among sensor nodes.

4 Framework Design

The previous sections have all given the reader the baggage to understand this section in which we propose a framework for a distributed semantic sensor network. With such a framework it becomes possible to ask questions about sensor measurements to a network of distributed sensor nodes. This framework addresses data integration, question federation and network architecture challenges. Moreover, it facilitates knowledge distribution in a sensor network, to enable long term use of the network.

In section 1.3 of the introduction we stated a number of challenges a distributed semantic sensor network framework would have to address. These, together with the requirements from the scenario in the introduction section, the research background of Section 2 and of course our work on question federation (Section 33.1) and data conversion 3.2) result in a set of requirements and ideas.

This part of the project is design-science research [22]. The proposed framework is thus a result of a search process for a design. In this section we follow a software engineering methodology to guide this search process and to elicit the requirements for a sensor network architecture to come to a more formal specification of the architecture. With these requirements and an multi-agent system methodology we come to the design of a framework. We discuss the different components of the framework and show how together they meet all the requirements.

With this framework we will implement the IJkdijk scenario as was presented in the introduction and describe the sensor network architecture, semantic questions and knowledge bases for each of them in Section 5. But first, the design of the framework, starting with the requirements.

4.1 Requirements

In the IJkdijk scenario a number of requirements for the system where already given. Inspired by the Sommerville software engineering methodology [58] we make a distinction between **functional requirements** and **non-functional requirements**. Functional requirements are statements of services that the architecture should provide, how the architecture should react to particular inputs and how it should behave in particular situations. Non-Functional requirements are constrains on the services or functions offered by the architecture. These include constrains on privacy, usability, efficiency and others.

4.1.1 Functional requirements

The functional requirements are directly linked to the four challenges that we identified in Section 1.3. We will discuss the main requirements of each of the following four challenges. These requirements are a specification of the challenges and refer to the work discussed in Sections 2.1 about sensor networks, 2.2 about semantics and 2.3 about multi-agent systems.

1. Construction of a distributed network architecture; how to facilitate communication among sensor nodes in a distributed environment
2. Facilitate the integration of sensor data; how to enable data integration by adding semantic web technologies to the sensor network domain and improve upon existing integration solutions
3. Question federation; how to split and federate a semantic question to enable the integration of distributed sensor data
4. Sharing of and reasoning with semantic knowledge; how can knowledge sharing allow a sensor network to operate in a dynamic environment and how reasoning with shared knowledge can increase knowledge at sensor nodes

The following functional requirements are explained with an example from the IJkdijk scenario to illustrate its necessity.

Construction of a distributed network architecture

1. A sensor node is an entity

Looking at the IJkdijk scenario and the challenges that need to be addressed by the framework, there are a number of reasons why a sensor node should be an entity. A sensor node should make autonomous decisions about whether or not to answer questions, based on local availability of data, resource computational limitations and characteristics of the requesting entity. A sensor node should be able to reason with knowledge or acquire knowledge about the environment by questioning others.

This requirement follows from the scenario, since in there is a requirement for a sensor node to answer questions about its local sensor measurements. It also relates to the work on Multi-agent Systems (Section 2.3) and Semantics (Section 2.2), in that an agent allows for reasoning and semantics make it possible to understand and answer questions about local data.

An example can be found in the IJkdijk scenario. If there is a sensor node that is very low on energy resources it might decide to answer some questions and drop others, based on the properties of the entity that sent the question.

2. The architecture allows for any degree of distribution

This requirement relates to the discussion on a distribution versus centralized approach as discussed in Section 2.1.2 on networks of sensors. As we can learn from the IJkdijk scenario, different degrees of distribution might be necessary. In the IJkdijk scenario the architecture is distributed since different parties place and maintain their sensors. However, centralized aggregation nodes can greatly reduce the complexity of the questions to the network. This results in a clustered architecture.

3. Events and Questions

In the IJkdijk scenario both normal questions and notifications or alerts, also known as events are used to exchange knowledge and question the network sensors. These were already discussed in Section 2.1.3 and are useful in any network in which sensor measurements should trigger an event.

An example of this was already given in the IJkdijk Project: "*Give me the water level when it reaches 100 centimetres.*"

4. In network data aggregation

In a high usage environment, in network data aggregation can result in reduction of data transfer and computational complexity. Sensor nodes with a specific aggregation functionality can be added to the network. These sensor nodes can be addressed by other resources, making the aggregated data available to the network. This requirement follows from the idea of data aggregation in sensor networks as discussed in Section 2.1.4.

As an example, in the IJkdijk scenario, a sensor node could be instantiated that aggregates water level and pressure measurements from other sensor nodes and calculates the risk of flooding based on these measurements. This resource can then be queried by other resources who need the risk of flooding at different locations.

Facilitate the integration of sensor data

5. A mutual understanding

To be able to ask questions to a network of distributed sensor nodes, they need a mutual understanding of the knowledge domain. The one to pose a question should thus know how the one to receive the question semantically describes his knowledge.

As an example, in the IJkdijk scenario, to be able to ask questions about temperature measurements of multiple temperature sensors, there should be an agreement on how to annotate temperature measurements and the possible datatype forms for temperature, such as degrees Celsius and degrees Fahrenheit.

6. Recognize inconsistencies in shared knowledge

As soon as a sensor node has an incorrect view of sensor data represented by another node, then communication about shared concepts between these two nodes can lead to ambiguous interpretation.

Conflicts in interpretation must be monitored and the node must be made aware about a possible conflict.

How inconsistencies to shared knowledge can be found was discussed in Section 2.2.3 on ontologies. This requirement is valid for the IJkdijk scenario, since maintaining a mutual understanding in a distributed environment requires sensor nodes to alert one another whenever a mutual understanding becomes invalid.

For instance, in the IJkdijk scenario, a sensor node addresses another sensor node that has knowledge about temperature measurements along the embankment. Assume that the temperature sensor node has changed its annotation of temperature measurements. The requesting node still assumes that the answer it will receive from the temperature aware sensor node contains this information in the old form. The temperature aware sensor node will be unable to answer the question of the addressing node, since they do not share the same concepts anymore. The requesting node thus has to be notified about the change of representation beforehand, so it can change the way it addresses the temperature sensor in the future.

Question federation

7. Complex semantic data combinations

The network must be able to answer complex questions about information stored in the sensor network. The question mechanism has to allow for parallel and serial addressing of multiple sensor nodes to receive parts of the answer to the question from each node. These partial answers then have to be combined into a single final answer. The background on question federation was discussed in Section 3. This is a requirement for the IJkdijk scenario, since the sensor nodes in the network are heterogeneous; measure different phenomena.
As an example, consider the question from the IJkdijk scenario:

"*What is the pressure at locations where the temperature is over 14 and the pressure over 30*".

First, the locations with a temperature measurement have to be addressed in parallel, then for each of these locations the pressure has to be found.

Sharing of and reasoning with semantic knowledge

8. Sharing of knowledge

Whenever a resource lacks the specific knowledge to answer a question of other sensor nodes then it should be able to request this knowledge from other nodes. Moreover, sensor nodes might share knowledge with other nodes beforehand, as soon as they learn about this knowledge from another source.

We have shown different types of semantic knowledge in Section 3.3 and we discussed how these can be exchanged between sensor nodes, by adopting a multi-agent communication paradigm, as was shown in Section 2.3.2.

As an example, in the IJkdijk scenario, an entity requests the temperature, in a degrees Celsius form, measured at different locations along the embankment. Some of the temperature nodes store temperature measurements in a degrees Celsius form, others in a degrees Fahrenheit form. Those sensors that measure in degrees Fahrenheit are unable to answer this question if they do not have the knowledge of how to convert from degrees Fahrenheit to degrees Celsius. By sharing this knowledge among sensor nodes, all nodes are able to answer the question.

9. Gaining knowledge by reasoning

The knowledge at a resource changes continuously. From this changing knowledge base, new knowledge can be derived through reasoning. This requirement is useful for IJkdijk scenario, since the answering capabilities increase when new knowledge is inferred.

Knowledge reasoning was discussed in Section 2.2.4, in which we have shown how to use semantic ontologies, rules and a semantic knowledge base to increase knowledge at a sensor node.

As an example, in the IJkdijk project, consider the following example question: "*Warn me when a location on the embankment is about to flood.*"
A reasoning rule could infer whether the embankment is flooding by combining facts known about the embankment, such as water level and pressure, which it can acquire from other sensor nodes through event subscriptions.

In the next section we discuss the non-functional requirements of a semantic sensor network architecture.

4.1.2 Non-functional requirements

Following the Sommerville methodology, the non-functional requirements have been split into product, organizational and external requirements. Non-Functional requirements are constrains on the services or functions offered by the architecture. There is no ranking in these requirements and all of these will be addressed in the proposed architecture.

Product requirements

Usability requirements

A user that wants to question the network should be able to easily formulate a question based on the semantics, that describe the knowledge made available by the sensor nodes in the network. The returned answer should be displayed in an easy to understand format.

Efficiency requirements

Due to the possibility of adding resource poor nodes to the network, the chosen architecture should be lightweight in processor usage, memory, disk space and bandwidth. Moreover, it should be possible to dynamically assign functionality to sensor nodes.

A question asked to the network should return an answer within an appropriate amount of time. This reply-time depends on the complexity of the question but should depend as little as possible on the size of the network.

Reliability requirements

Whenever a question is asked to the network a reply should be given. Questions asked to the network should be replied to with the correct answer or at least an approximation of the correct answer. If a question cannot be answered the user should be notified about this as well.

Portability requirements

Since different parties add custom sensor nodes to the network, the chosen architecture has to be platform independent.

Organizational requirements

Due to the multiple application domains for this architecture, there are no specific constraints on delivery, implementation language and standards.

External requirements

Interoperability requirements

Due to the nature of sensor networks, different sources of information must be queried. These resources could be supplied by a number of parties and presented using different formats. It is required that the chosen architecture is able to retrieve information in a wide number of formats and still be able to interpret the sensor data at different sensor nodes.

It should be possible to extend the sensor network by adding new sensor nodes. This could implicate that information available in the network also changes. All parties and nodes have to be notified in case of changes to the knowledge available in the network.

Privacy and Security requirements

Sensors can measure data which should not be made public. The architecture should make it possible to specifically state who should have access to specific knowledge. Furthermore, encryption of transmitted data should be possible in the chosen architecture.

4.2 Design

In this section we work towards a framework for a distributed sensor network, which addresses both the functional and the non-functional requirements from the previous section. First some high level design decisions are made, in which a number of existing techniques and frameworks are chosen to help address some of the requirements. Then following a **multi-agent methodology** the design of the framework is described from a multi-agent perspective. Each of the components of the network is explained. This section ends with an example of the framework in a sensor network setting.

4.2.1 Design Decisions

This section analyzes the functional and non-functional requirements and how different techniques and architectures can be used to address them.

Functional Requirement 1 implicates that sensor nodes are autonomous entities with control over a local knowledge base and with the ability to communicate with other nodes. Each resource should be able to perform tasks to accomplish data aggregation (Requirement 4). Moreover, sensor nodes should share knowledge with other nodes (Requirement 8).

These three requirements point us toward using a sensor network architecture based on the **multi-agent paradigm** (See Section 2.3). Agents are entities that can perform tasks and have control over their own knowledge base. An agent system can handle any kind of distribution, depending on the number of agents and their specific implementation (Requirement 2). Multi-agent systems make use of communication protocols which make it possible for agents to exchange knowledge (Requirement 8) and also interact in a subscription protocol (Requirement 3). Also parallel message exchange between multiple agents is possible with a communication protocol, meeting Requirement 7 (See Section 2.3.2).

The non-functional requirements require the system to allow for the exchange of messages in different formats. Agent communication using the FIPA protocols allow agents to exchange messages and interpret messages written in different languages, by specifying the content language and ontologies. Communication protocols can also determine when a conversation fails and notify the user about this, meeting the reliability requirement. The efficiency requirements are met by using an agent based architecture, because agents are lightweight entities and agents with specific functionality can easily be instantiated on a network node. Agent communication can be encrypted and each agent can determine whether or not to send data to a requesting agent, thus meeting the privacy requirements. Agents can be implemented on any kind of operating system, which means they are independent of the platform, meeting the portability requirements.

The functional requirement 5 requires sensor nodes to have a common understanding of the data that is represented by the other nodes. By explicitly stating what type of data is provided by a sensor node, through annotation of semantic meta data, it becomes possible to combine and aggregate sensor readings (requirements 4 and 7). Moreover, the semantic meta data can be used to compare data and notice a change in data representation using ontology deprecation information, meeting requirement 6.

The semantic **resource description framework**, RDF (See Section 2.2), provides a rich framework for representing data and semantic meta data. Moreover, it incorporates a query language that allows querying of semantic data (Requirement 7). Semantic reasoners can use semantic rules to reason about semantically annotated data and convert resource data, meeting requirement 9. Our own work on question federation (Section 3) can be used to make complex combinations of sensor data possible, thus addressing requirement 7. Our work on data conversion helps to make a mutual understanding of knowledge possible between different sensor nodes, by addressing the data integration problem (Requirement 5).

The framework which will be discussed in the coming sections is thus based on a multi-agent system. It uses semantic web techniques such as the resource description framework (RDF) and a semantic query language (SPARQL) as a basis to solve data integration problems in a distributed

sensor network. Moreover, the framework incorporates our work on question federation and data conversion to increase the capabilities of this framework. The combination of these enables the framework to do things that existing frameworks cannot.

To make the implementation of the framework feasible we have decided to use the JADE multi-agent framework [55] as a basis for implementing the multi-agent system. We use the JENA semantic web framework [47] to enable description and querying of semantically annotated sensor data. The next section describes our proposed framework, by using a multi-agent methodology to identify the agents, agent communication and internal agent components.

4.2.2 Agent Analysis

The framework we are about to discuss is based on a multi-agent system. We decided to use a multi-agent systems design methodology to identify agents, agent interaction and agent responsibilities [59]. Figure 7 describes the different steps of the methodology and how they contribute to the deliverables. In the following sections we walk through the steps of the methodology. We have made minor adjustments to the way the methodology was used, since it did not always fit our situation. As you can see in the figure, the methodology is an iterative process. To keep it simple, we will not describe the iteration and present the results as they were found after the final iteration. The first phase is to describe a use case diagram for our scenarios.

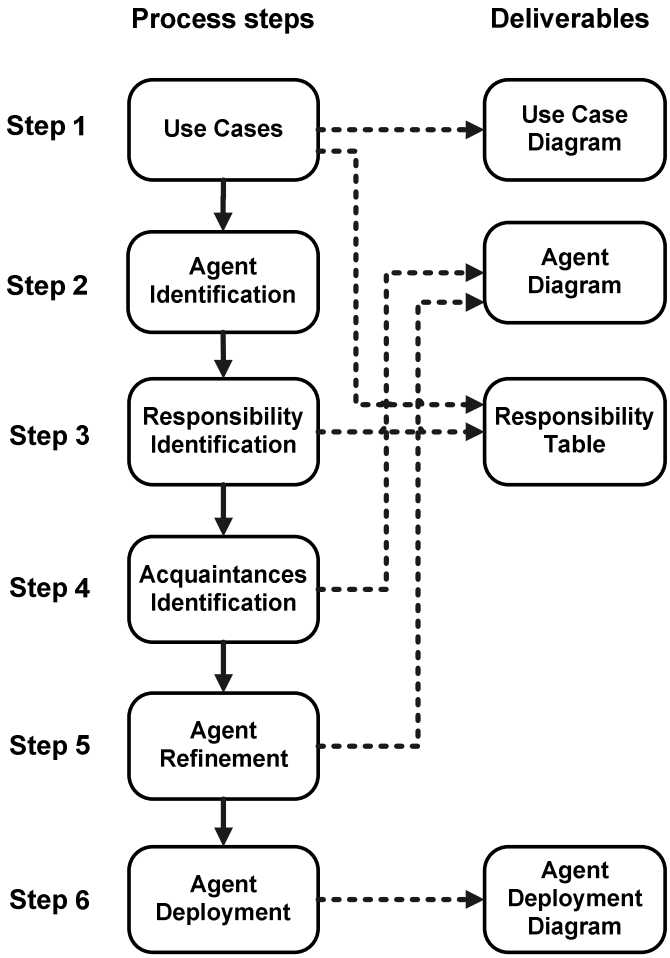


Figure 7: Multi-Agent Design Methodology

4.2.3 Use Cases

According to the methodology, use cases describe a specific scenario. They provide information about the different tasks a multi-agent system has to perform. From the use case diagram the different agents that are needed can be identified.

The IJkdijk is a possible example sensor network for which the proposed framework could be implemented. However, the framework should be applicable to many more sensor network scenarios. Therefore we decided not to model the IJkdijk scenario in a use case, but instead add the requirements from the requirements section to a use case diagram (Figure 8). Figure 8 shows how both a user and a sensor interact with the sensor network and which components are identified given the requirements.

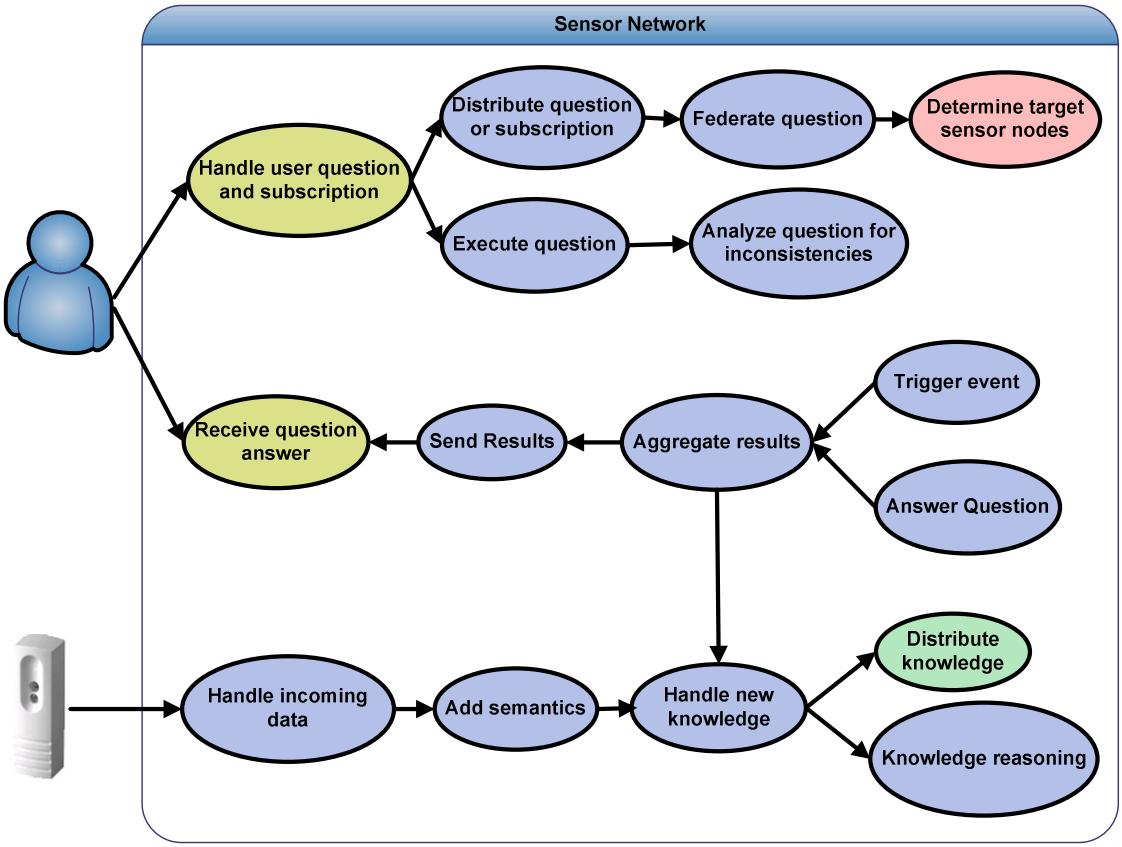


Figure 8: Use case diagram

These are the most common tasks of the network. The next phase of the methodology is to identify the type of agents that need to be implemented to create a network with the above capabilities.

The next phase in the methodology is the agent identification phase, which results in an agent diagram.

4.2.4 Agent type identification

As indicated earlier, the methodology is an iterative process. We started by identifying a single agent. By using the methodology we experienced that the best design for this framework consisted of four different agents. Figure 8 shows which of the use cases is attributed to each agent. Figure 9 shows the legend of Figure 8, indicating the four different types of agents and the use cases for which they are responsible. We will now explain why we have chosen to introduce these four types of agents into the framework.

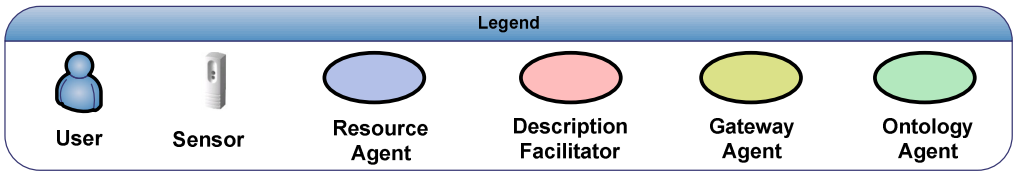


Figure 9: Agent Diagram and legend of figure 7

Most of the use cases can in principle be handled by a single agent. This agent is responsible for handling questions about sensor measurements. We identified this agent as the **resource agent.** The resource agent is an agent that represents and maintains a semantic knowledge base containing sensor data. It is an entity that answers questions posed by other agents and it can itself actively acquire knowledge by sending event subscriptions to other resource agents. A resource agent can be implemented on each sensor node. The resource agent is the main building block of the sensor network.

In Figure 8 one can see a use case in which there is a need to determine sensor nodes (or agents) in the network to whom to send a question. Each resource agent thus needs a way to know which other resource agents are available in the network. The **description facilitator** is an agent which maintains a list of agents that have registered to the network. It works like a yellow-pages service. If an agent wants to be found by others, it registers to the description facilitator. Other agents, such as the resource agents, can subscribe to the description facilitator to be notified about addition or removal of agents to the network.

For the interaction of a user with the network we identified a third agent: the **gateway agent**. The gateway agent acts on behalf of a user. It sends questions, subscriptions and new knowledge from a user application or endpoint into the network. Once the network has figured out an answer, has triggered an event, or has new knowledge, the gateway agent forwards this data back to the user application or endpoint. This agent is necessary to bridge the gap between an application and the sensor network and can thus not be directly identified by looking at the requirements. Although the gateway agent shares some capabilities with the resource agent, we still decided to divide their responsibilities between two agents. The main reason for this is simplicity. When two sets of responsibilities are sufficiently distinct, then it is a good idea to identify two different agents to implement these responsibilities. This makes the agents more lightweight and simpler.

A fourth agent was identified to facilitate the distribution of knowledge in the network. The **ontology agent** plays a main role in this responsibility. It acts in a similar way to the description facilitator in that other agents can register their ontologies, semantic rules and conversion functions with the ontology agent. Other agents that have a subscription to this information are then notified about updates of the knowledge. The reason why the ontology agent and the description facilitator agent were separated in the design is that the used JADE framework already provides an existing description facilitator agent, with tools to analyze its knowledge. However, the description facilitator could not be changed to add the responsibilities of the ontology agent and thus this second agent was identified..

4.2.5 Responsibilities Identification

This section specifies the responsibilities of each agent. In this phase we are only interested in the communicative responsibilities of the agents, since this tells us something about the interaction between agents and the protocols with which this communication is managed.

Table 19 gives an overview of the communicative responsibilities of each agent. These responsibilities are extracted from Figure 8. For each communicative responsibility the interaction protocol is given. The role of the agent in the protocol: responder or initiator gives information on the stage the protocol is in during that responsibility. The *with* column indicates the agents with whom the conversation can take place. The *when* column specifies when this responsibility applies.

Agent	Responsibility	Protocol	Role	With	When
Resource Agent (RA)	Answer a question by another resource agent	Request	R	RA	A question is received
	Send a question to another resource agent	Request	I	RA	A question is unanswered
	Respond to subscription of another agent	Subscribe	R	RA,GA	Changed data matches subscription
	Initiate a subscription with an agent	Subscribe	I	RA, DF, OA	Wish to be notified by change in the network
Description Facilitator (DF)	Receive registrations of agents	Request	R	RA,OA,GA	An agent enters the network
	Receive subscriptions of agents who want to receive registration updates	Subscribe	R	RA,GA	An agent initiates a subscription
	Send notifications to agents that subscribed to updates of registrations	Subscribe	I	RA,GA	An agent enters the network
Ontology Agent (OA)	Receive ontologies, rules and conversion functions from agents	Request	R	RA	A knowledge update was sent to the ontology agent
	Receive subscriptions of agents who want to receive knowledge updates	Subscribe	R	RA, GA	An agent initiates a subscription to knowledge updates
	Send notifications to agents that subscribed to updates of knowledge	Subscribe	I	RA, GA	A knowledge update was received
Gateway Agent (GA)	Receive questions and subscriptions from a user application	Request	R	USER	A user sends a message
	Send questions into the network	Request	I	RA	A question was received from a user, or an event was triggered
	Send subscription into the network	Subscribe	I	RA	A subscription was received from a user
	Receive answer to a question	Request	R	RA	An answer is received

	Receive a trigger from a subscription	Subscribe	R	RA	An event was triggered in the network
	Send answers from question to the user	Request	I	USER	An question or subscription was finished

Table 19: Responsibility Table

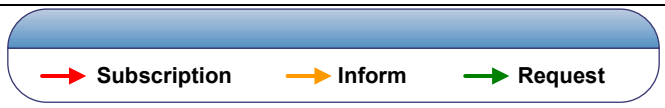
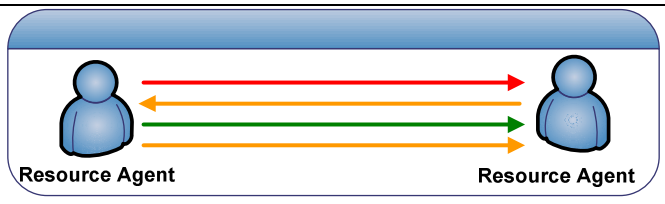
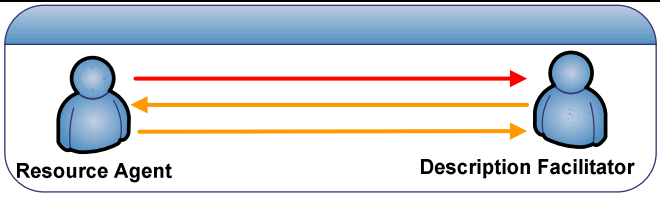
(R = responder, I = initiator, RA = resource agent, OA = ontology agent, GA = gateway agent, DF = description facilitator, USER = a network user)

According to [59], agent responsibilities can be directly translated into agent behaviours (See also Section 2.3.2, on agent behaviours). For each of these responsibilities a behaviour can be instantiated which allows an agent to perform the task. In JADE there are existing implementations of the FIPA request and subscribe protocols (See Section 2.3.3 for an overview of these protocols). These protocols can be added as behaviours and handle the agent interaction from there on. Each behaviour specifies what should be done with incoming messages and when to send a message to another agent.

To make the interaction between the agents in the network clearer to the reader, the following section goes into more detail on the possible interactions between the resource agents, gateway agent, ontology agent and the description facilitator.

4.2.6 Acquaintance Identification

In this phase of the methodology we determine whom interacts with whom. We go a bit further than the original methodology in that we not only describe the interaction partners, but also the kind of messages they exchange (for information on agent message exchange read Section 2.3.2). Table 20 shows the interaction between the agents identified in Section 4.2.4. For simplicity we have not added the interaction between all agents in a single figure, as the methodology proposes, but split the interaction into those between each two agents. On the left side there is an explanation of the interaction between two agents and on the right there is a figure to clarify the interaction and the types of messages that are exchanged.

Legend	
A resource agent can have a subscription with another resource agent, to be notified about a change in local knowledge at the other agent. A resource agent can also request some knowledge from another resource agent. It receives an inform message with the answer. A resource agent can also request the ontology of the local knowledge base of another resource agent.	
The resource agent has a subscription with the DF agent, to receive inform messages when an agent enters or leaves the network It needs this to know who its neighbours are. *It also sends an inform to the DF agent to notify that it has entered the network*	

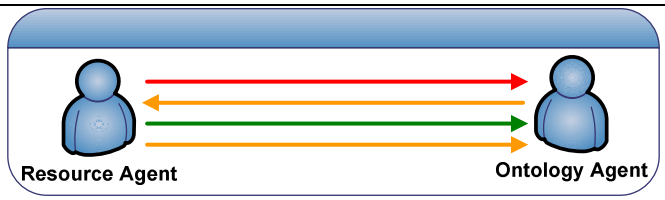
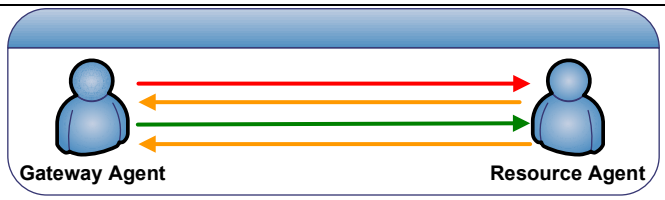
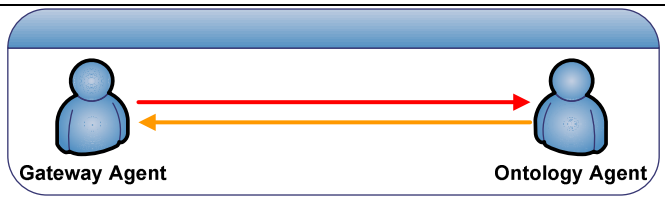
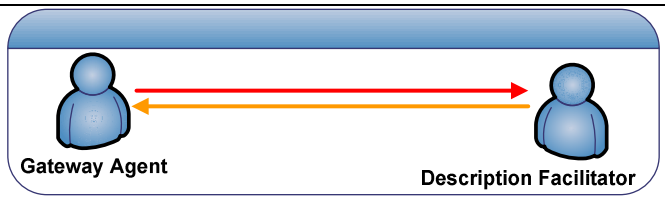
The resource agent sends his knowledge (ontology, rules, conversions) to the ontology agent with an inform message (this message triggers a notification to other agents). It requests current knowledge and also initiated a subscription with the ontology agent to be notified about new knowledge. This knowledge is then used for reasoning and question federation.	
The gateway agent can forward a subscription or request from a user to a resource agent. It receives inform messages for both. A gateway agent can also request the ontology of the local knowledge base of a resource agent. These ontologies are used to allow the gateway agent to federate questions.	
The gateway agent has a subscription with the ontology agent to be notified about new knowledge in the network. This knowledge is used for reasoning, to be able to federate questions more specifically.	
The gateway agent has a subscription with the DF agent to be notified whenever an agent enters or leaves the network. It forwards this knowledge to a user. The user then knows what knowledge has become available and what has disappeared.	

Table 20: Agent Acquaintances

Now the interaction between agents was discussed, we continue to the actual implementation of the agents. The next section describes the components of the agents and how these make sure the requirements for our distributed semantic sensor network are met.

4.2.7 Agent Design

This section goes into more detail on the design phase of the methodology. We describe the implementation of the individual agents that constitute the distributed semantic sensor network. This section is based on the design of the resource agent. This agent is the most complex of the four. The gateway agent is a simpler version of the resource agent. The description facilitator agent and the ontology agent are simple yellow-book services and only implement a subscription protocol behaviour.

Resource Agent

Figure 10 shows the components of the resource agent. It describes which components interact with one another, and what interaction there is with other agents through input and output of the resource agent.

Components are grouped into modules. These modules will be explained separately because they implement key behaviours of the agent. Take a look at Figure 11, it describes the modules of Figure 10.

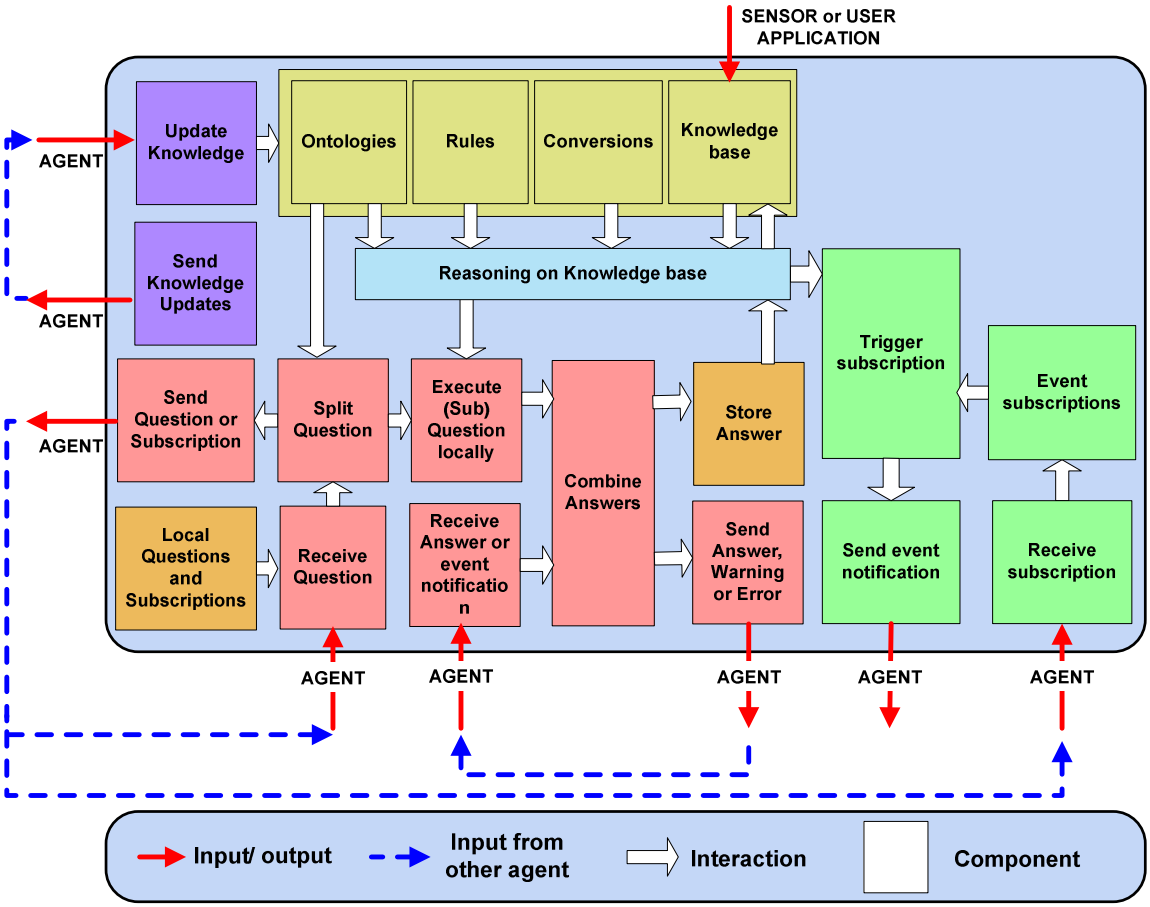


Figure 10: Resource Agent: Components and interaction

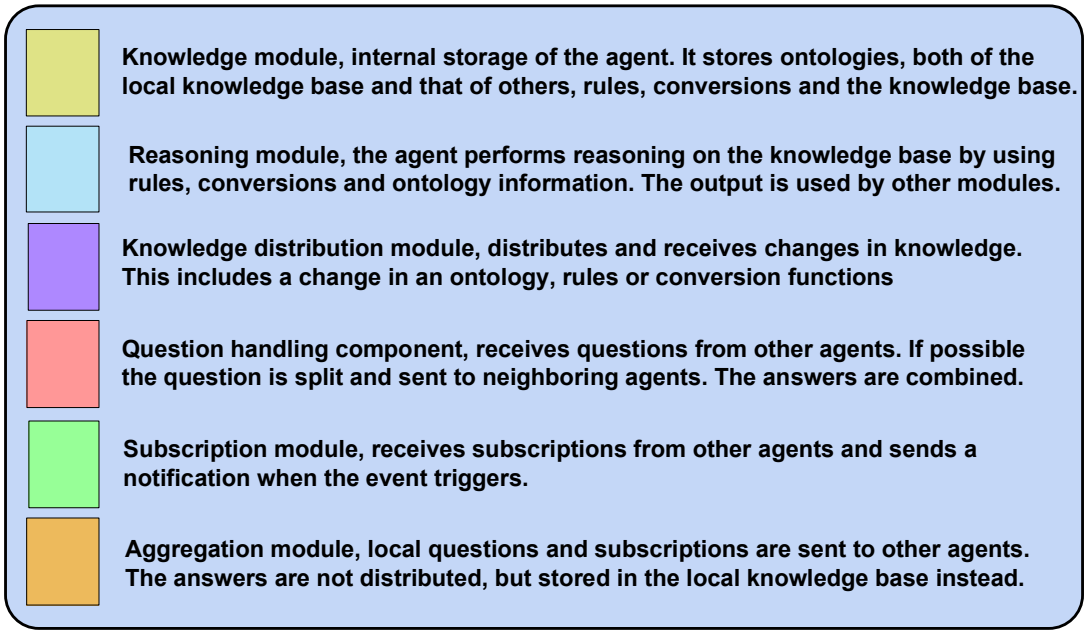


Figure 11: Resource agent components

The following paragraphs briefly discuss the modules of the resource agent. They show how the building blocks such as semantics, reasoning, query language and our work on data conversion and

question federation can be used to get the requirements into the resource agent. The section finishes with a short discussion on the gateway agent, ontology agent and description facilitator.

Knowledge Module

The knowledge module includes all the knowledge the agent has. The main chunk of knowledge for a resource agent is the knowledge base, containing the sensor measurements. This knowledge is semantically annotated in RDF (Section 2.2.2). The semantics with which this knowledge base is described is stored in an ontology (2.2.3). This ontology, together with the ontologies of other resource agents forms the ontology component of this module. The rules component contains all the semantic rules this agent has received from other agents in the network (Section 2.2.4). The conversion component is a collection of conversion functions from input format to output form (3.2).

The knowledge base can be filled with sensor measurements. Figure 12 shows how this interaction is implemented.

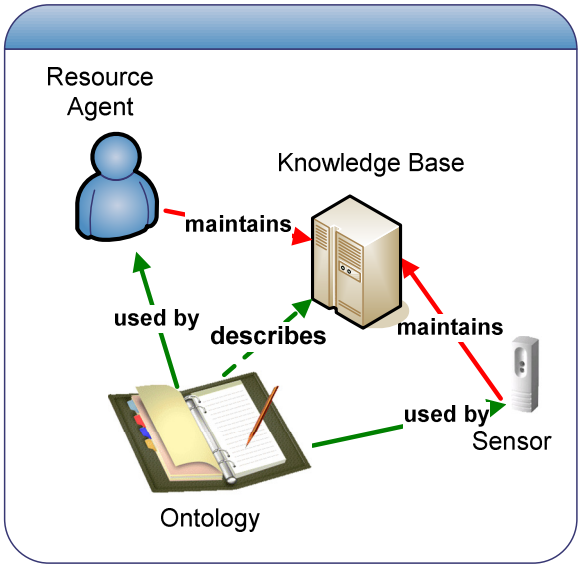


Figure 12: interaction between agent, sensor, application

As an example, in the IJkdijk scenario, a temperature sensor measures the temperature every minute and stores the temperature value, the unit and the time of measurement in the knowledge base. The resource agent recognizes a change to the knowledge base and use this to answer questions about sensor measurements or trigger subscription events about the new knowledge.

The knowledge in the knowledge module enables agents to have a mutual understanding of the data in the network. The knowledge module thus enables mutual understanding, which was requirement 5 of this framework. The next paragraph explains how this agent knowledge can be used for reasoning about the knowledge base.

Reasoning Module

As was explained in Sections 2.2.4 and 3.2, ontological metadata, rules and conversions can be used to increase the local knowledge of an agent. The reasoning module uses different types of reasoning methods to find new knowledge.

The OWL reasoner PELLET is used to infer new facts from the knowledge base by applying the OWL metadata to the knowledge base. The OWL reasoner fills in gaps in the knowledge base that were left empty by the sensor.

The JENA built in rule reasoner executes the semantic rules on the semantic knowledge base. It does not use ontological metadata for reasoning, but it uses existing semantic facts in the knowledge base and the knowledge in hierarchical ontologies to derive new facts.

The JavaScript engine Rhino executes the conversion functions on the knowledge base to add new converted triples to the result set of a SPARQL question.

These three reasoning methods in the reasoning module enable increasing knowledge by using knowledge. This addresses requirement 9 of this framework.

As an example. In the IJkdijk scenario, if the water level measurement of a sensor changes to 125 centimetres and there is a semantic rule that says that if the water level is over 125 centimetres that the risk of flooding increases to "high". This rule can be implemented like a JENA rule and the reasoner can infer the risk of flooding by executing the rule on this new annotated sensor measurement. The knowledge that the risk of flooding is high can now be added to the knowledge base.

Knowledge Distribution Module

When new knowledge is inferred or added to the agent from a user application, this knowledge can be shared with the network. The knowledge distribution module sends the new knowledge; either a changed ontology, a new or changed conversion function or a new or changed rule to the ontology agent. Other agents are subscribed to the ontology agent to receive knowledge updates. So by sending a message to the ontology agent such an update is triggered and the other agents are notified. The knowledge distribution module addresses the knowledge sharing requirement of this framework.

This module also handles new knowledge updates that reach the resource agent. These updates are then added to the knowledge module.

Although not shown as a separate component in Figure 10, as soon as a resource agent enters the network, it registers with the description facilitator agent. Other agents are thus notified of the addition of a resource agent and they can request the ontology of the local knowledge base of the new resource agent. Receiving these ontology requests and sending the local ontology to the requesting agents is also part of the knowledge distribution module.

As an example, in the IJkdijk scenario. As soon as a new sensor node is added to the network, all other nodes in the neighbourhood are notified of this event. They can request the ontology of this new sensor node through the knowledge distribution module components. Upon receiving the ontology of this new agent they can start sending questions to this agent, because the new ontology can be used in the question federation process.

The reason why an agent maintains a knowledge base is because it wants to share this knowledge with other agents. The next paragraph discusses how an agent handles a semantic question about its knowledge and how the answer is returned.

Question Handling Module

This module has a component, implemented as an agent behaviour that captures messages to this agent that contain a semantic question or a subscription. These questions and subscriptions are described as SPARQL queries (Section 2.2.5). A question or subscription is always sent to a question federation component that splits the SPARQL query (33.1). The query splitting is based on the ontologies of the ontology component in the knowledge module. An agent thus always uses both its local ontology and ontologies of neighbours to determine where to send a part of the question or the subscription. A part of the query can thus be executed on the local knowledge base. Other parts might have to be sent to other agents. Only when the local answer is found and all the answers from other agents were received, does the agent start to combine the answers into one final answer. The agent knows which answers to combine for each question because a question and all its sub-questions carry a unique question identifier. The sub-answers can thus be combined by using this identifier to group it with other sub-answers. The answer is then sent to the agent that posed the question. The question handling module, by using a question splitting method, enables the complex combining of distributed resources. With this it addresses requirement 7 on complex semantic data combinations and the question part of 3 on events and questions.

Another requirement of the framework is that inconsistencies in shared knowledge are detected (Requirement 6). A resource agent could receive a question about semantic concepts such as classes and properties that are annotated as being deprecated in the ontology. In this case, the resource agent sends the answer to the question, but also sends a warning to the requesting agent, to alert him that the answer to the question might not be correct.

Another way to check for inconsistencies is implemented using ontology versioning (Section 2.2.3). A requesting agent sends the expected version of the ontology describing the knowledge base

of the receiving agent together with the question itself. The receiving agent checks if the expected ontology version matches the local ontology version. In case of an inconsistency between version numbers the requesting agent is sent an error message. The question itself is dropped.

Subscription Module

Requirement 3 stated that the framework should make it possible to handle both user initiated questions and network generated events. The subscription module makes this possible for agents in the network to subscribe to events generated by sensors at a resource agent.

An agent in the network can subscribe to a resource agent to be notified about an event that was measured for instance by a sensor that is connected to the sensor node on which this resource agent is running. As soon as a sensor changes the knowledge base, this change (after performing reasoning on it) is used to trigger subscriptions from other agents. The conditions on which to trigger are also expressed as SPARQL queries and can thus be executed on the semantic update to the knowledge base. If the SPARQL condition query returns a result this means that the subscription applies to the update. The agent that sent this subscription condition is now sent a notification about the event. Such a notification can contain details about the event. This depends on the SPARQL query that was used to trigger the subscription.

To clarify this, consider the following example. In the IJkdijk scenario a user wants to be notified when there is a high risk of flooding somewhere along the embankment. The user does not only want to be notified about this, but also wants to receive the exact location of the sensor that triggered the flooding event. A subscription is thus a SPARQL query that triggers the event of changed data and a semantic question in the form of a SPARQL query that is executed to get the remaining properties of the event that triggered. The user receives the answer to the semantic question only when the event condition was triggered in the network.

Aggregation Module

One of the requirements of the framework was that in network data aggregation should be possible (Requirement 4). The way we have decided to implement the resource agent makes it possible to have data aggregation at each sensor node. Data aggregation works with a number of local subscriptions that can be sent into the network. The subscriptions are federated into the network by using the normal question federation procedure (Section 3). However, as soon as an answer is received from the other resource agents this answer is added to the local knowledge base instead of being sent to a requesting agent.

As a small example, in the IJkdijk scenario, it could be useful to add sensor nodes to the network on which a resource agent is running that collects measurements from a number of homogeneous sensor nodes. This aggregating resource agent could execute a number of semantic rules on the aggregated data and then publish this data to the rest of the network.

In the following three paragraphs we discuss the other agents in the network: the gateway agent, ontology agent and the description facilitator.

Gateway Agent

A gateway agent is almost identical to a resource agent. The main difference is that the gateway agent does not have a knowledge base. So, a gateway agent cannot be asked any questions about local knowledge. Such questions are forwarded to resource agents in the network.

An additional capability of the gateway agent is that it can exchange message objects with a java user application. It can thus be used to send questions into the network from a user application, and it can send notification to user application objects in case an answer was received from any of the agents in the network.

The gateway agent is also used to send the semantic knowledge and knowledge updates to the user application. This way, the user application is shielded from the network, but still can notified about subscription events and knowledge updates.

Ontology Agent

The ontology agent is nothing more than a repository of knowledge available in the network. It stores the ontologies, rules and conversion functions that were published by the resource agents. It can share this knowledge with other agents in the network.

It is possible to add multiple ontology agents to the network, to reduce the downsides of having a central agent on which the whole network is depending.

Since the ontology agent is just an agent, it can receive new knowledge through agent messaging protocols. This allows for flexible maintenance of knowledge since every user application, administrator or sensor node in the network that is allowed to send messages to this agent is in theory able to share knowledge with other nodes in the network.

Description Facilitator

Every agent in the sensor network has to register to the description facilitator. This agent allows other agents to search for agents in the network. The description facilitator can, just like the ontology agent, be distributed using multiple sub-description facilitator agents. This reduces the downsides of having this central bottleneck in the network.

Grouping agents in the sensor network

When describing the implementation of the sensor network we have mainly talked on the level of single agents. The main advantage of a sensor network is of course if there are multiple agents. In theory there can be an unlimited amount of any of the agents. We have already shown that multiple ontology agents and description facilitators can stop the relying on a single bottleneck in the network. For each application or user that needs access to the network a gateway agent can be started. For any sensor, or group of sensors a new resource agent can be added to the network. The number of agents to add to the network depends on the size, the amount of data that needs to be stored at each agent and other constraints such as the implementation environment.

One of the requirements of the sensor network was that any level of distribution should be supported (Requirement 2). In essence this is possible with the framework we have just described. A completely centralized solution would have only a single resource agent, one gateway agent, one ontology agent and a description facilitator. This choice to use this framework is then however irrational, since its strengths arise from the distribution. In a fully distributed system, a number of resource agents, an ontology agent, a description facilitator and one or more gateway agents are all fully connected to one another. Anything in between a centralized and a fully distributed architecture requires grouping of agents. Figure 13 shows an example agent deployment diagram of a clustered distributed network (See Section 2.1.2 for the clustered communication architecture).

A user group has access to a network of sensors through a gateway agent. This gateway agent is fully connected to some of the agents in the network, but not to others. One or more of the resource agents in each group (group on the left and group on the right in the figure) are connected. We labelled this connection with “1” in Figure 13. A question from a user whose gateway can only access the resource agents of a single group can thus still question information stored at the resource agents of another group through this connection. The agents that have connections to other groups could potentially function as an aggregator. Such an aggregator thus shields the information from its group and decides what knowledge to make available to another group. This agent has become an entity (Requirement 1 of this framework), since it can decide who to give access to his knowledge base and what knowledge to make available.

With this we finalize the discussion of the design of the framework for a distributed semantic sensor network. We have introduced a number of agents that could fulfil the requirements that we identified from existing literature and a number of scenarios. We have shown how these agents were implemented and how together they can give the sensor network the behaviour that we want: to be able to handle questions and events on sensor measurements in a dynamic sensor network environment in which knowledge, measurements and composition of the network topology is ever changing.

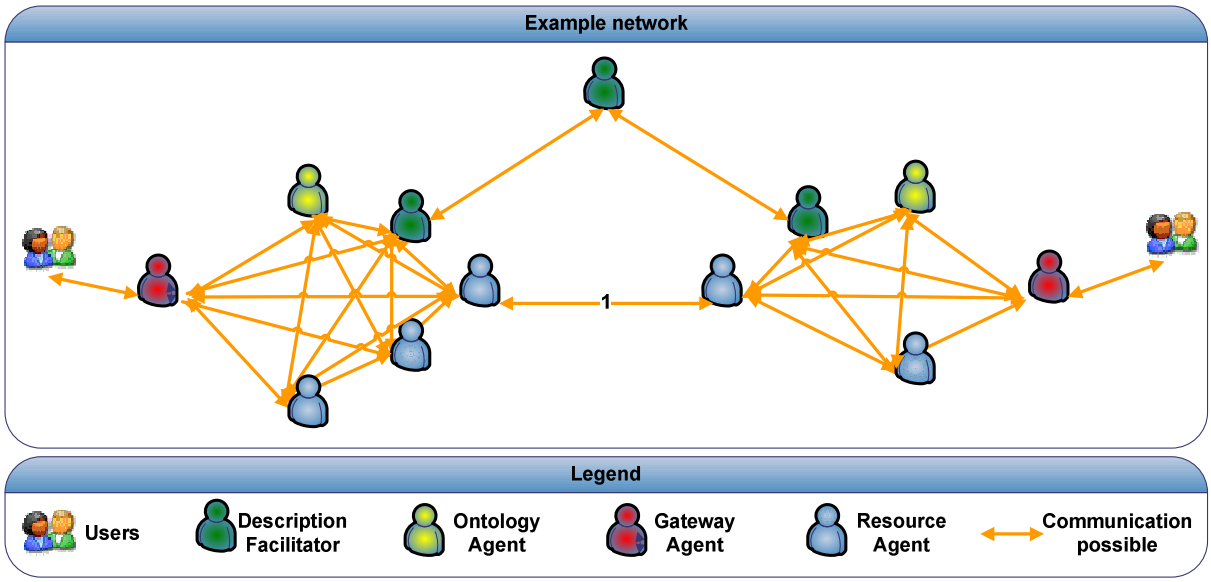


Figure 13: Agent Deployment Diagram

In the next section we evaluate the proposed framework by testing some example questions of the IJkdijk scenario in a demo of the IJkdijk implemented with this framework. With the implementation we will show how the example questions from the scenario are handled by the network. Moreover, we will show an example of knowledge distribution, knowledge deprecation and how these effect the behaviour and capabilities of the network as a whole.

These experiments will then provide us with input for the discussion section in which we evaluate the framework; its strengths and current weaknesses.

5 Case study

The evaluation of the distributed semantic sensor network framework is done in two ways. First, an experimental evaluation is performed by doing a case study with an IJkdijk simulation. Second, an analysis is performed, to see whether the challenges and requirements of a distributed semantic sensor network have been met. The IJkdijk scenario is implemented in a simulation based on our proposal framework. Many different settings of sensors and different sensor types are possible for the IJkdijk scenario. We have chosen to simulate only a small number of sensors of three different types: temperature, pressure and water level sensors. To handle the sensor measurements from these sensors we introduced six resource agents. Each agent maintains a knowledge base containing the sensor measurements of its sensors and is able to answer questions about these measurements. Figure 14 shows the physical architecture of the IJkdijk and how the different sensors are handled by each resource agent (1 through 6). Three aggregating sensor nodes were added to the simulation (7, 8 and 9). Each of these aggregates knowledge from other sensors, as is indicated by the dotted lines. Each sensor is located in a different area. The scenario distinguishes between locations in the Netherlands and Germany.

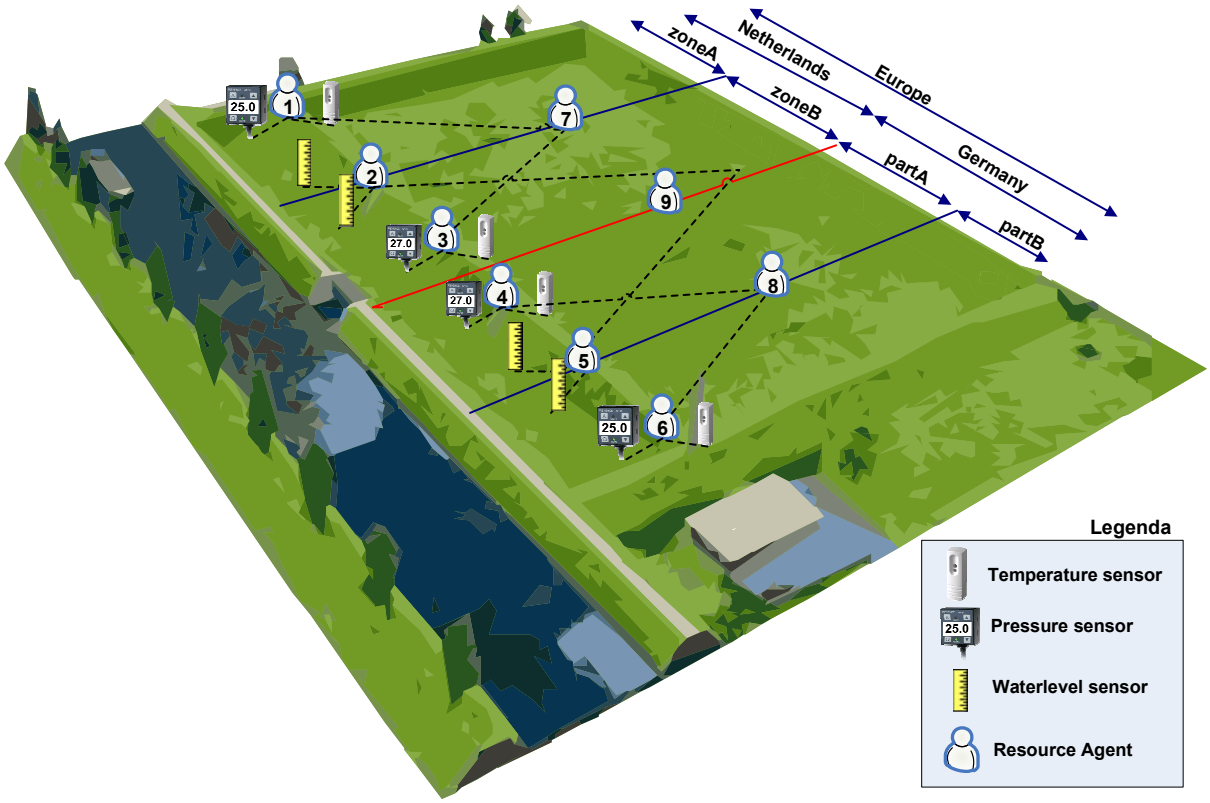


Figure 14: Case study of the IJkdijk, with our framework

Since some of the sensors are located in the Netherlands and others in Germany we also assume that these sensors are maintained by different companies. For ease and recognition we will call these Dutch company (agents 1, 3 and 7) and German company (agents 4, 6 and 8). Moreover, a specialized water management company (agents 2, 5 and 9) has decided to add water level sensors to the network, which they maintain themselves. It is assumed that all companies use different conventions for annotating sensor data. However, each of these uses RDF to model their data and an OWL ontology to describe the infrastructure of the RDF knowledge bases.

The knowledge bases and ontologies maintained by each agent in the network can become quite complex. Some of these are given in the Appendix B. The most important knowledge of each agent is given in the tables below. For each agent the type of sensor, the unit with which it measures and the location is given. For the resource agents that aggregate sensor data from other agents we show what type of sensor data is aggregated, what new knowledge is derived from that and what the unit of this new knowledge is.

Resource agent #	Sensors	Unit	Location
1	Temperature	Degrees Celsius	Europe, Netherlands, zone A
	Pressure	Newton Meter	Europe, Netherlands, zone A
2	Water level	Centimetres	Europe, Netherlands, zone A
	Water level	Centimetres	Europe, Netherlands, zone B
3	Temperature	Degrees Celsius	Europe, Netherlands, zone B
	Pressure	Newton Meter	Europe, Netherlands, zone B
4	Temperature	Degrees Fahrenheit	Europe, Germany, part A
	Pressure	Newton Meter	Europe, Germany, part A
5	Water level	Metres	Europe, Germany, part C
	Water level	Metres	Europe, Germany, part D
6	Temperature	Degrees Fahrenheit	Europe, Germany, part B
	Pressure	Newton Meter	Europe, Germany, part B

Resource agent #	Aggregation of	Derives	Unit	Location
7	Temperature, Pressure	Embankment Stability	StabilityFactor	Europe, Netherlands, zone A, zone B
8	Temperature, Pressure	Embankment Stability	StabilityFactor	Europe, Germany, part A, part B
9	Water level	Flooding Risk	FloodingRisk	Europe, part A, part B, part C, part D

The agents in the network for the IJkdijk scenario have been identified. An important design decision when implementing a distributed semantic sensor network with our proposed framework is how to group agents. Grouping agents results in a clustered sensor network architecture, which has some advantages over fully distributed network architectures when it comes to scalability and accessibility of sensors measurements. The communication architecture in Figure 15 shows how agents are grouped for this particular simulation of the IJkdijk scenario. Given the fact that three companies have added sensor nodes to the network, we decide to split the German company (agents 4, 6 and 8) and Dutch Company (agents 1, 3 and 7) sensor nodes, with temperature and pressure measurements, in two distinct groups. The water management company also groups its sensor nodes into a distinct group (agents 2, 5 and 9) and decides to make only the aggregating resource agent 9 accessible for other agents. This is why the gateway agent is in a separate group with resource agent 9. Each group has an ontology agent and a description facilitator agent to handle the addition and removal of new agents and knowledge to that group. In this scenario each group describes sensor data in a different language (different ontology). The German Company, Dutch Company and water level Company each describe sensor measurements with their own ontologies.

A user application allows a user to ask questions to the network. This user application adds a gateway agent to the network. For this particular simulation the gateway agent is grouped with the sensor nodes of both the Dutch and Germany company, and is also grouped with the aggregating resource agent 9 of the water management company.

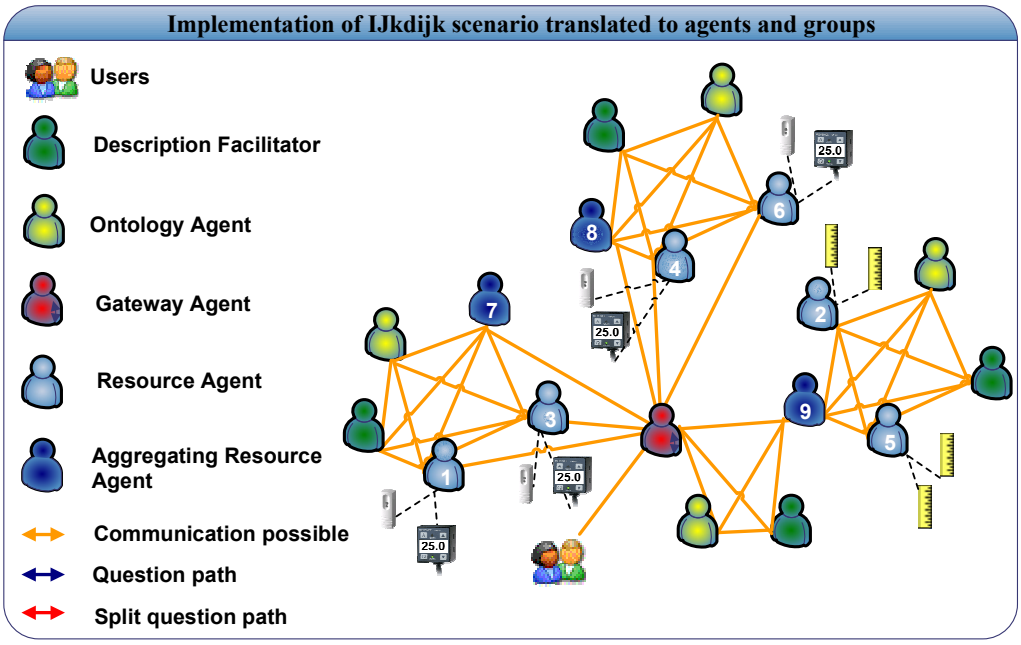


Figure 15: Grouping for IJkdijk. On the left, legend for each of the examples below

Given the translation of the scenario into the distributed sensor network framework we can start discussing question federation and knowledge sharing in the network in an experiment of the IJkdijk simulation.

Initializing the network

Each agent shown in Figure 14 is implemented on a sensor node, which has the processing capabilities to run the JADE agents and use the Semantic Web technologies such as the JENA framework for RDF, OWL, rule reasoning and SPARQL. The agents boot at network initialization. Each resource agent loads its knowledge base, which they share with the sensors connected to the sensor node. The resource agents subscribe to the ontology and description facilitator agents in their group to receive knowledge updates and to be notified when new resource agents enter the network. The description facilitator agent makes sure the resource and gateway agents in a group know of each others existence by sending notifications. Upon receiving such a notification, the resource and gateway agents request the ontology of the new resource agents. The resource and gateway agents thus have the ontologies of their neighbouring resource agents. These ontologies can be used to federate questions. When a resource agent has also received the shared ontologies and data conversion knowledge from the ontology agent, the resource agents can use this for reasoning on the local knowledge base. The agents are now ready to start receiving questions from users and other sensor nodes.

How questions are handled by the framework

In this section a number of questions that were mentioned in the introduction section about the IJkdijk are tested against an implementation of the scenario. We will show how a question, translated to a SPARQL query is handled by the network. The SPARQL queries are based on ontologies and data sets explained throughout this report. For some of the agents an example of the knowledge base and ontologies can be found in Appendix B. A SPARQL question is constructed given one or more ontologies. A question is thus posed in a specific “language” or a combination of languages. For each question it is explained how the question is split (if necessary) and where it is sent. The answer that is received from the network is given (if available).

For each question an evaluation is given of the behaviour of the framework. The example questions are presented in increasing complexity. Knowledge sharing examples are given in between questions to show how these effect the capabilities of the network.

The first and most simple question for the IJkdijk scenario is explained in Figure 16. A SPARQL query about temperature measurements is formulated by using the Dutch company ontology for sensors.

What are the temperatures measured at the embankment?

```
PREFIX sensors: <http://ijkdijk.nl/sensors.owl#>

SELECT
  ?sensor
  ?temperature
WHERE
{
  ?sensor  a sensors:TemperatureSensor ;
     sensors:measurement ?m .
  ?m sensors:value ?temperature .
}
```

Answer
Node1:sensor1 25^^degreesCelsius
Node3:sensor1 27^^degreesCelsius

The gateway agent forwards the question to each agent having knowledge about temperature measurements, given the Dutch company sensor ontology. These agents return all their measurements to the gateway agent. The gateway agent sends all these measurements to the user.

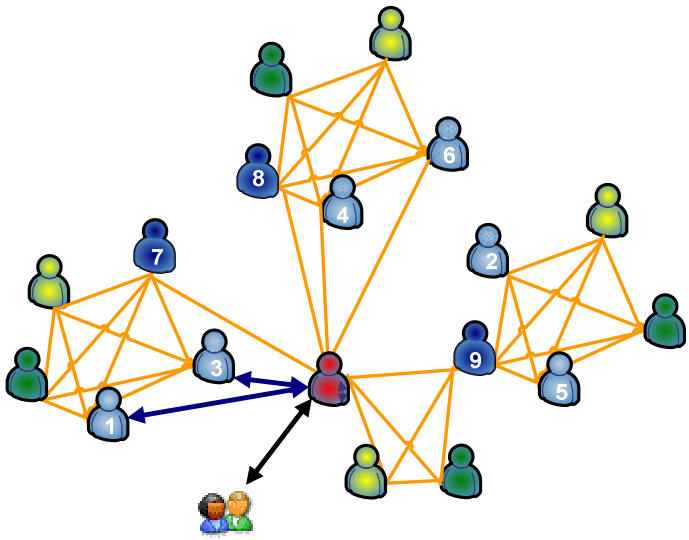
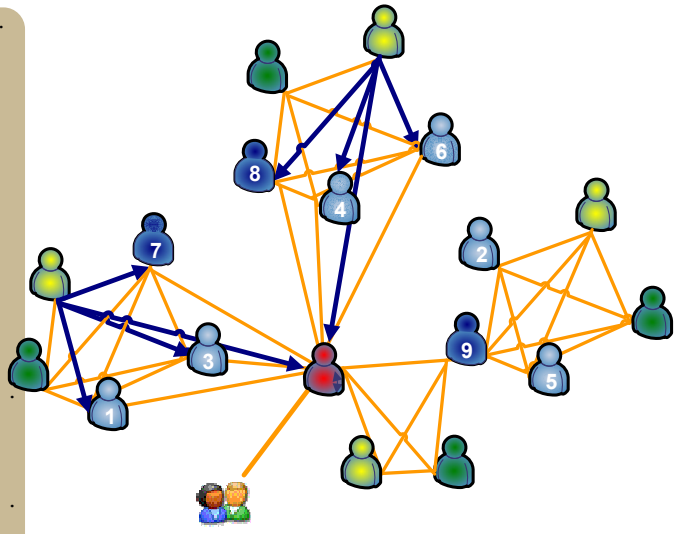


Figure 16: example question: what are the temperatures measured at the embankment?

Resource agents 1 and 3 describe their measurements with this ontology and are thus sent this question (These sensors speak the language in which the question was asked). The network in its current state does not allow the German company sensors to be addressed by this same question. However, we would like to know the temperature measurements along the entire embankment and thus need to address these sensors as well. Both the German and Dutch company decide to share a mapping (translation) of their sensor ontology with the sensor ontology of the other company with the network. Each company notifies the ontology agent in their group about the new ontology and these ontologies are now shared with the resource agents in that group and also with the gateway agent, which is in both groups. An example of ontology sharing is given in Figure 17.

Share a mapping ontology between German and Dutch company sensor ontologies

```
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix g: <http://ijkdijk.de/sensors.owl#> .
@prefix d: <http://ijkdijk.nl/sensors.owl#> .

<http://ijkdijk.eu/sensormapping.owl>
      a owl:Ontology .

d:Sensor owl:equivalentClass g:Sensor .

d:TemperatureSensor owl:equivalentClass
      g:TemperatureSensor .

d:PressureSensor owl:equivalentClass
      g:PressureSensor .

d:location owl:equivalentProperty g:location .

d:Measurement owl:equivalentProperty g:Reading .

d:measurement owl:equivalentProperty
      g:measurement .

d:timestamp owl:equivalentProperty g:timestamp .

d:value owl:equivalentProperty g:sensorvalue .
```



The Dutch and German company decide to create an mapping ontology to describe how their sensor ontologies relate. This mapping ontology is shared with the network through the ontology agents of both groups. The resource agents receive the mapping ontology and add this to their knowledge base. By importing this mapping in the private ontology of each resource agent, the mapping becomes possible through OWL reasoning at the resource agent.

Figure 17: example of knowledge sharing in the form of a mapping ontology

The mapping ontology has now been received by the resource agents. The mapping ontology is added to the collection of ontologies in the knowledge module of the agent (see Figure 10). The mapping ontology is used in the reasoning module of the agent, where the agent uses the new ontology to translate its knowledge base into the terms of the ontology of the other company's annotation. Now the OWL reasoning process in the reasoning module enables the resource agents to answer questions about sensor measurements described in both the local ontology and the ontology of the other company. We have evaluated the example question in Figure 16 again and this time resource agents 4 and 6 also received the question and returned their temperature measurements.

From these examples we can see that agents in the network are autonomous entities (Requirement 1) and that any degree of distribution is possible with the framework (Requirement 2). A mutual understanding is accomplished by sharing mapping ontologies between sensor nodes, enabling the sensor to integrate their measurements with sensor nodes described by different ontologies (Requirement 5).

In the question federation section we described how adding static resources as individuals to the ontology of a resource agent makes it possible to federate questions with these resources very specifically to this resource agent. Each of the resource agents has static knowledge about their location along the embankment added to their ontology. Take a look at Figure 18. A user wants to know the temperature(s) measured at zone A along the embankment. The gateway agent uses the ontologies of all resource agents and decides, based on individuals in the ontology, that resource agent 1 is the only agent with knowledge about temperature measurements in zone A.

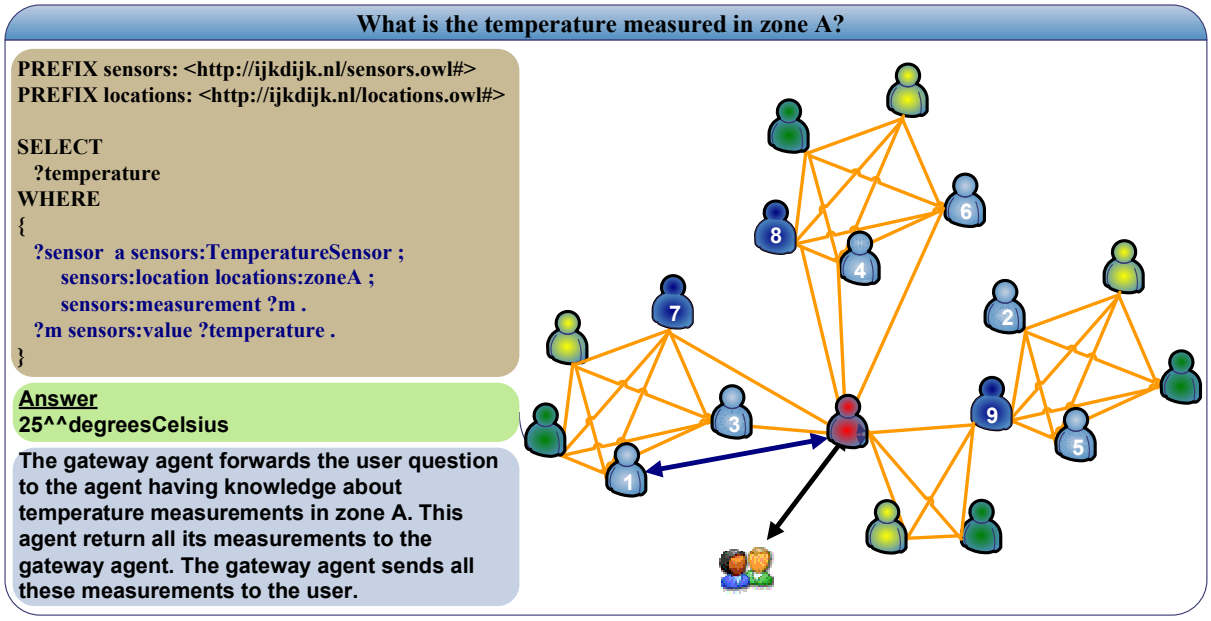


Figure 18: example question: what is the temperature measured in zone A?

Each resource agent maintains sensor measurements at a specific location. Assume that each of these agent knows about a locations ontology, and this ontology defines what locations are part of another location (such as Germany is part of Europe). By introducing a rule into the network that allows agents to derive the super locations of their current location, the agents become capable of answering questions that apply to these locations. Figure 19 shows how such a rule can be distributed in the network through the ontology agent. This rule specifically applies to locations for the Dutch side of the IJkdijk and is thus only added to the ontology agent of the Dutch company group.

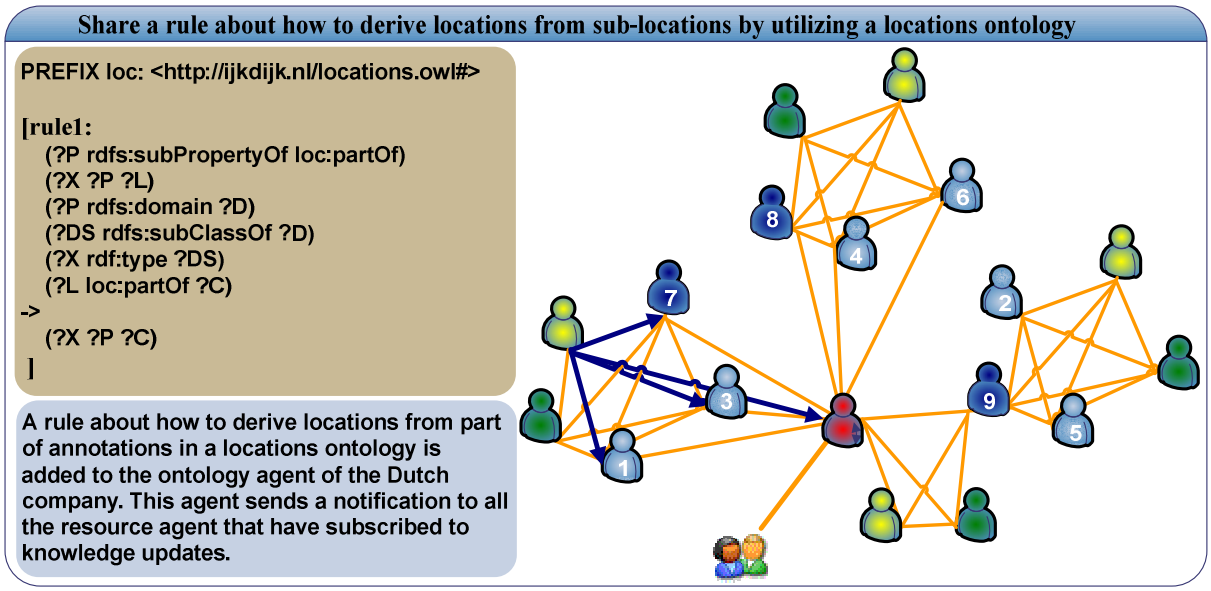


Figure 19: knowledge sharing example, sharing a rule to derive locations from sub-locations

Introducing this rule in the network allows a user to ask questions about sensors in the Netherlands or Europe instead of sensors in a specific zone. Whereas before it was only possible to send a question to a resource agent in a specific zone, such as the example in Figure 18. We have evaluated the effect of the above rule by sending the question of Figure 18 again, but with the locations:zoneA substituted with locations:Netherlands. This question about temperature measurements in the Netherlands was sent to both resource agent 1 and 3.

A downside that we experienced from introducing rules like these lies in the fact that these rules add new triples to the knowledge base of a resource agent. The rules work bottom up and thus generate a huge amount of new triples. The SPARQL language selects triples from a knowledge base. By adding new triples to this knowledge base as an effect of rule reasoning, the SPARQL query will return multiple results, from which some are 'derived results'. To give an example, we can ask the network for all the temperature measurements and their measuring location. Before introducing this rule we would have received the answers from resource agents 1 and 3, that their location is zone A. After introducing the rule and reasoning with the rule, the resource agents 1 and 3 return results for zone A, the Netherlands and Europe. An example of the network answer for such a question is given in Appendix C, Figure 27. Locations the Netherlands and Europe are derived as being super locations of zone A, but since the triples are now in the knowledge base, a question about locations will always result in three answers instead of a single answer. This effect has implications for the behaviour of the sensor network, for instance when ordering results, or limiting results to the highest value only.

With this example we showed that the framework addresses the knowledge sharing requirement (Requirement 8) and can use this knowledge gain new knowledge by reasoning with it (Requirement 9).

Thus far, each question has only travelled a path of length one, since the gateway agent could directly address all the involved agents. However, since the gateway agent has access to resource agent 9, but not directly to resource agents 2 and 5, we can ask a question about water level measurements that should travel a path of length two. Figure 20 explains the handling of such a question.

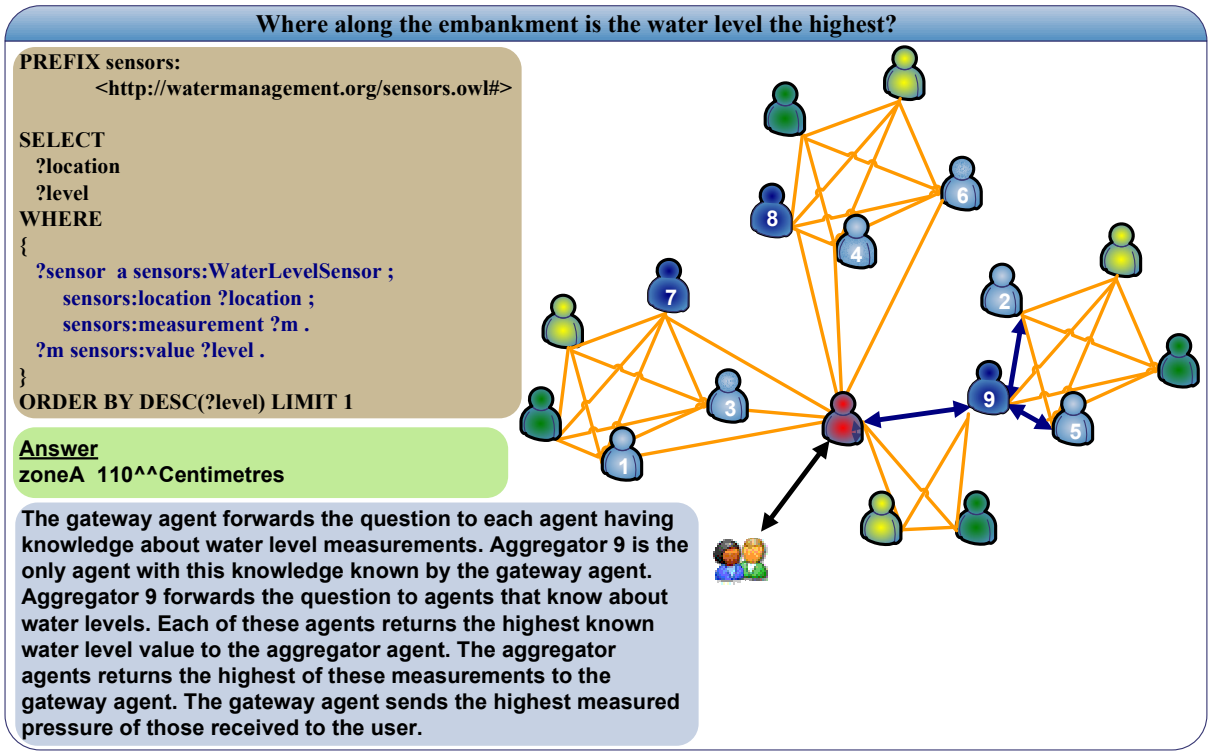


Figure 20: example question: where along the embankment is the water level the highest?

For this question to reach the water level sensor nodes, the aggregating resource agent 9 had to publish to the rest of the network that it has knowledge about water level sensor measurements. Although this agent does not directly manage water level sensors, by adding the water level sensor ontology classes and properties to its ontology it can make other sensor nodes think that it has knowledge about these things. Upon receiving the question, the aggregating resource agent forwards the question to agents 2 and 5, since these agents have not yet received this question. By sending the order by clause of this question to each agent, the amount of network traffic is limited, since only the highest results are sent between agents.

The rerouting of a question is an interesting capability of the framework. The strength of the framework however results from its capability to split questions and address sensors nodes with parts of the question. Figure 21 shows an example of a question that cannot be answered by a single agent, because none of the agents has knowledge about all the sensor types required to answer the question. The gateway agent splits the question into two sub-questions. The blue lines indicate the parallel sending of the first sub-question, indicated with blue in the SPARQL query. The red lines indicate the serial sending of the second sub-question, indicated with red in the query, when the answers to the first are being received.

We observed two main inefficiencies in the current framework that are closely related and can be well observed in this question. We have already explained how rule reasoning can result in many derived results in a SPARQL question. When the sub-question about temperature measurements in Figure 21 returns these derived results, then the serial federation of the next sub-question about pressure measurements will be sent ones for every derived result. For example, agent 1 returns three results: a temperature measurement at location zone A, the Netherlands and Europe. The question about pressure measurements will be sent three times to agent 9, ones for every location (serial federation). The question federation behaves this way, since the ?location variable in the pressure measurement sub-question is substituted with each location result of the temperature measurement sub-question. To get an idea of the explosion of serial sub-questions that can occur this way, take a look at the full results for this question in Appendix C, Figure 26. For each result row a question was sent to resource agent 9. Out of nine questions, eight of those are not necessary, assuming that we are not interested in the derived locations. A second inefficiency in the network is that all the sub-questions are sent serially (one after the other) to resource agent 9. It would be more efficient to

rewrite the query to ask about all the measurements for all the locations in a single question to resource agent 9. This however requires grouping of results, currently not possible with SPARQL. In the discussion section we will say more about limitations like these of the current Semantic Web technologies.

This example shows that the framework addresses the requirement of complex semantic data combinations (Requirement 7), by allowing for question federation.

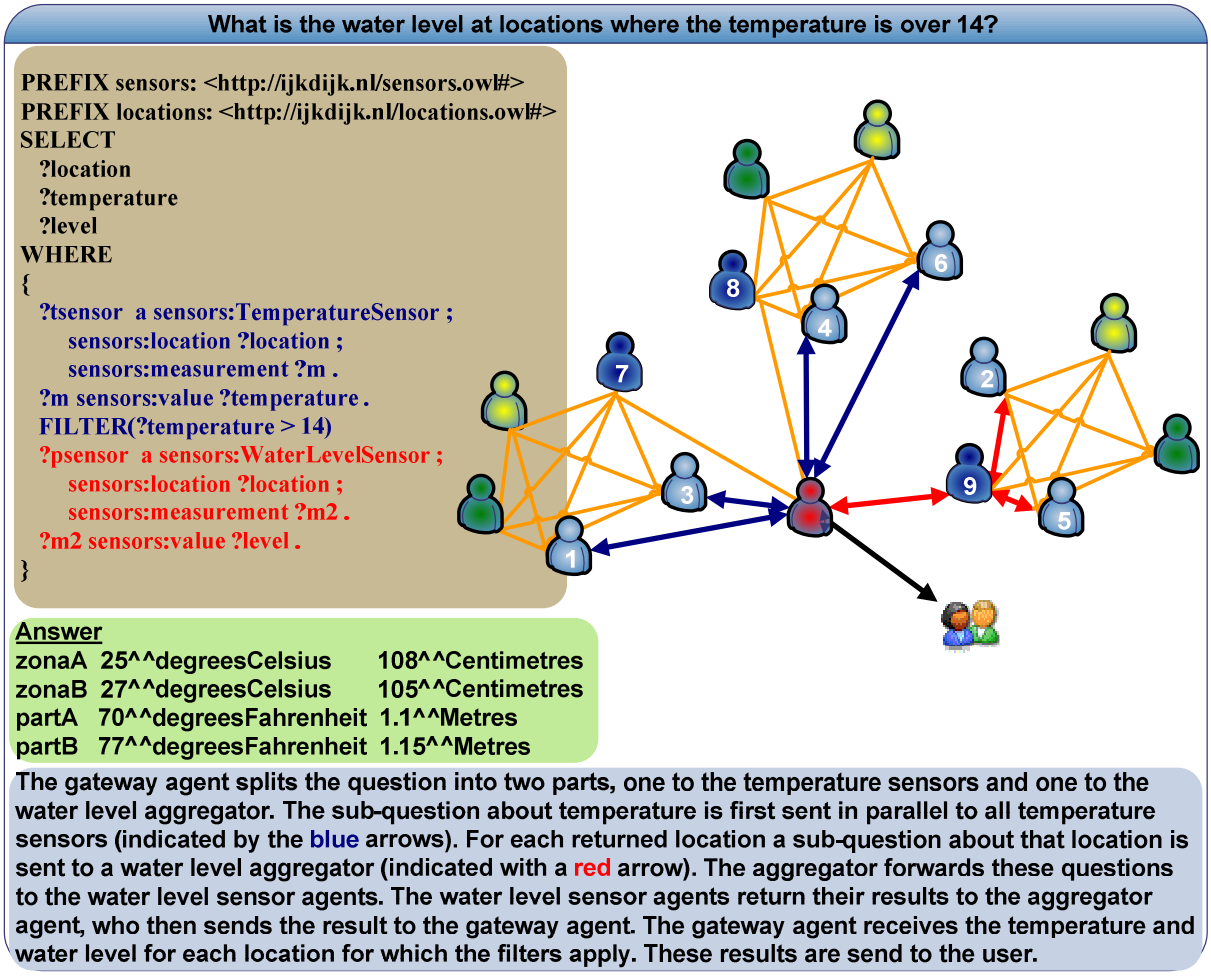


Figure 21: example question: what is the water level at locations where the temperature is over 14?

In the answer to the previous question there are differences in unit for the answers. The temperature values are in degrees Celsius or degrees Fahrenheit and the pressure values are in Centimetres or Meters. The filter method in this case does not work properly, since the form of the temperature measurement is not the same for every result. Also, would this question have contained an order by clause that ordered the results on the temperature value, then the results would not be ordered correctly in their semantic meaning, since a Fahrenheit measurement of 70 is lower then a degrees Celsius measurement of 25. However, the value 70 is higher then the value 25, so the ordering is the other way around.

To solve this problem we can distribute data conversion information in the network. In this case, conversion knowledge about how to convert from a degrees Fahrenheit measurement to a degrees Celsius measurement would solve this problem. Figure 22 gives an explanation of how data conversion knowledge can be distributed in the network. In the previous question, adding a conversion property function to convert the ?temperature measurement to degrees Celsius would be sufficient to solve this problem once the conversion knowledge is distributed in the network.

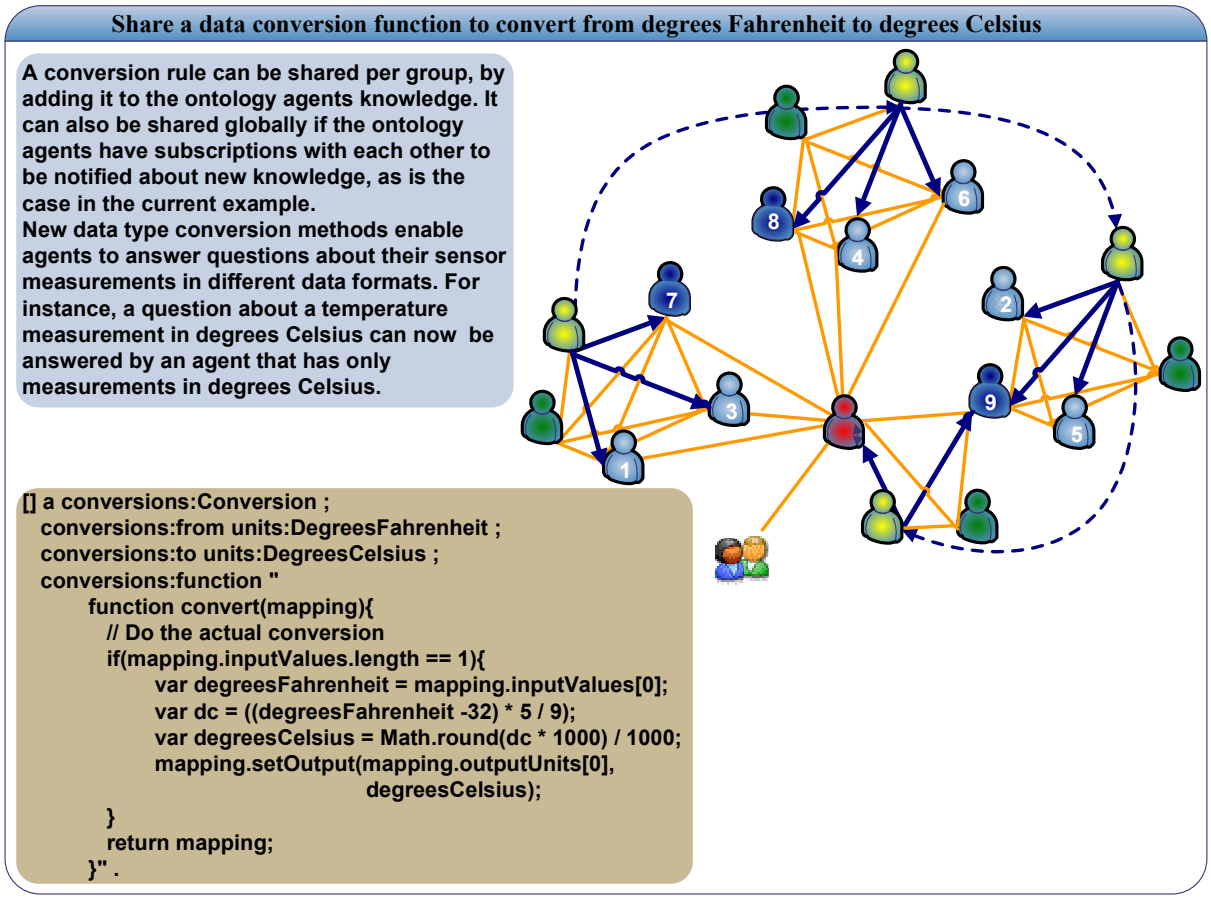


Figure 22: knowledge sharing: sharing a data conversion function

By sharing such conversion rules, the behaviour of the network changes when it comes to questions like the one in Figure 21. Appendix C, Figure 28 shows a result of a similar question, but this time with datatype conversion and an order by descending temperature. Each agent has converted its answer to the correct unit, allowing a better integration of the distributed data.

Ontologies can be shared in the same way as rules and conversion knowledge (See Figure 19 and Figure 22). Ontologies serve two purposes in our framework. Infrastructure ontologies are used to describe the knowledge of a sensor node and are used in the question federation process. Hierarchical ontologies contain domain knowledge that can be used by many sensor nodes in the network to increase their knowledge by reasoning with these ontologies and semantic rules. The infrastructure ontologies should not be shared among sensor nodes in the network. They describe the knowledge of a sensor node and should thus only be known by those agents which have this kind of knowledge. Hierarchical ontologies on the other hand can be shared all the time. These ontologies do not affect the question federation process.

An important aspect of a sensor network in a dynamic environment is that the sensor nodes themselves are aware of the environment around them. Sensor nodes have knowledge about the sensors connected to them. However, sensor nodes can also actively acquire knowledge from other sensor nodes, by questioning their knowledge. Resource agents that have subscriptions to other sensor nodes to be notified about changes in their measurements are called aggregating resource agents. Figure 23 gives an example of an event subscription sent by an aggregating resource agent. Event subscriptions are a requirement for the challenge on creating a distributed sensor network architecture (Section 4.1.1) and are a means to make knowledge sharing in a sensor network possible. Such subscriptions can be added to the resource agent by a human operator. The agent sends its subscriptions into the network at boot time. The trigger of this event subscription is first federated based on the trigger query (the select query). Upon notification, the aggregation agent sends the question about the event and adds the knowledge to its local knowledge base (the construct query).

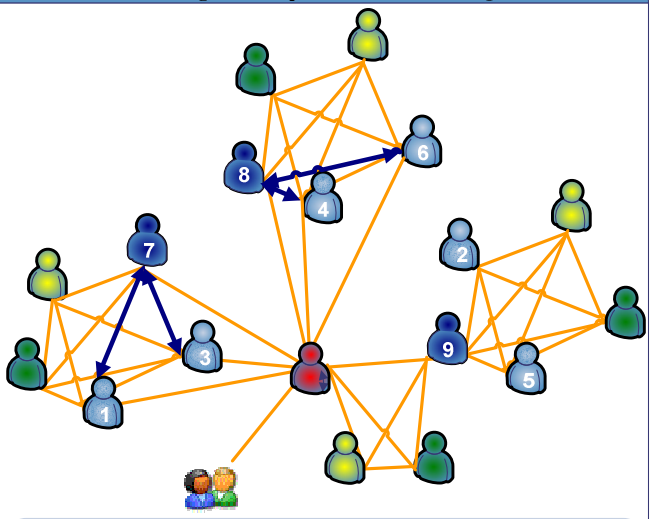
In this example, aggregating resource agents 7 and 8 always know about the highest temperature at the embankment. They can do the same for the highest pressure at the embankment. Based on this knowledge and a semantic rule these resource agents could infer the possibility of an embankment flooding or breaking. By sharing this knowledge with the network, by adjusting the ontology of these agents to include the status of the embankment, the sensors have added the ability for a user to question the status of the embankment.

Give me the latest temperature measurement when there is a change in temperature over 25 degrees Celsius

```
PREFIX sensors: <http://ijkdijk.nl/sensors.owl#>
PREFIX pf: <pfunction.library.>
PREFIX  units: <http://ijkdijk.nl/units.owl#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX : <http://ijkdijk.nl/resourceAgent7#>
SELECT
        ?sensor
WHERE {
        ?sensor a sensors:TemperatureSensor ;
                sensors:measurement ?m .
        ?m sensors:value ?value .
        ?value pf:convert (units:DegreesCelsius ?dc) .
        FILTER(?dc > 25) .
}

CONSTRUCT {
  :Embankment
        :highestTemperature ?dc .
}
WHERE {
        ?sensor a sensors:TemperatureSensor ;
                sensors:measurement ?m .
        ?m sensors:value ?value .
        ?value pf:convert (units:DegreesCelsius ?dc) .
}
ORDER BY DESC(?dc) LIMIT 1
```

Answer

```
@PREFIX : <http://ijkdijk.nl/resourceAgent7#>
 :Embankment
        :highestTemperature
                25^^degreesCelsius .
```



Both aggregator agents 7 and 8 send a subscriptions to agents in their neighborhood (red SPARQL question) to be notified about changes in temperature measurements when temperature reaches 25 degrees Celsius. Resource agents 1,3,4 and 6 execute the trigger question each time a temperature measurement changes. They first convert the changed measurement to a degrees Celsius format and check if the filter applies. If so, they notify the aggregator. The aggregator receives the notification and sends the second construct query (blue SPARQL question) to the resource agent that was triggered. The resource agent answers the question and the aggregator adds the new triples to its knowledge base.

Figure 23: event subscription: get the latest temperature, when there is a change in temperature over 25 degrees Celsius

Aggregating resource agents 7 and 8 can have knowledge about the latest temperature and pressure measurements of their part of the embankment with these subscriptions. For the user this makes it possible to directly question these sensor agents for the highest measurements, without having to address agents 1, 3 ,4 and 6. Moreover, the aggregating resource agents could use these acquired measurements to calculate embankment stability or flooding risk, by applying rules to the newly acquired data. A user could then have a subscription to the network, asking for a notification when there is a risk of flooding somewhere along the embankment. User subscriptions work in the same way as the subscription explained in Figure 23. Both the direct questions and the subscriptions show that our framework supports the sending and answering of subscriptions and questions about sensor measurements (Requirement 3) and that in network data aggregation becomes possible with subscriptions (Requirement 4).

We tested adding a new agent to the network. The registration with the description facilitator and the subscription with the ontology agents makes that the new agent can be questioned by other agents within a reasonable amount of time in the current implementation. Figure 24 shows how the agents communicated when the new agent entered the network. Being able to add and remove agents and sensors from the network is important to make it possible for the sensor network to operate in a dynamic environment in which sensors can be added or removed. This is a requirement to meet the challenge on knowledge sharing and reasoning as was discussed in Section 4.1.1.

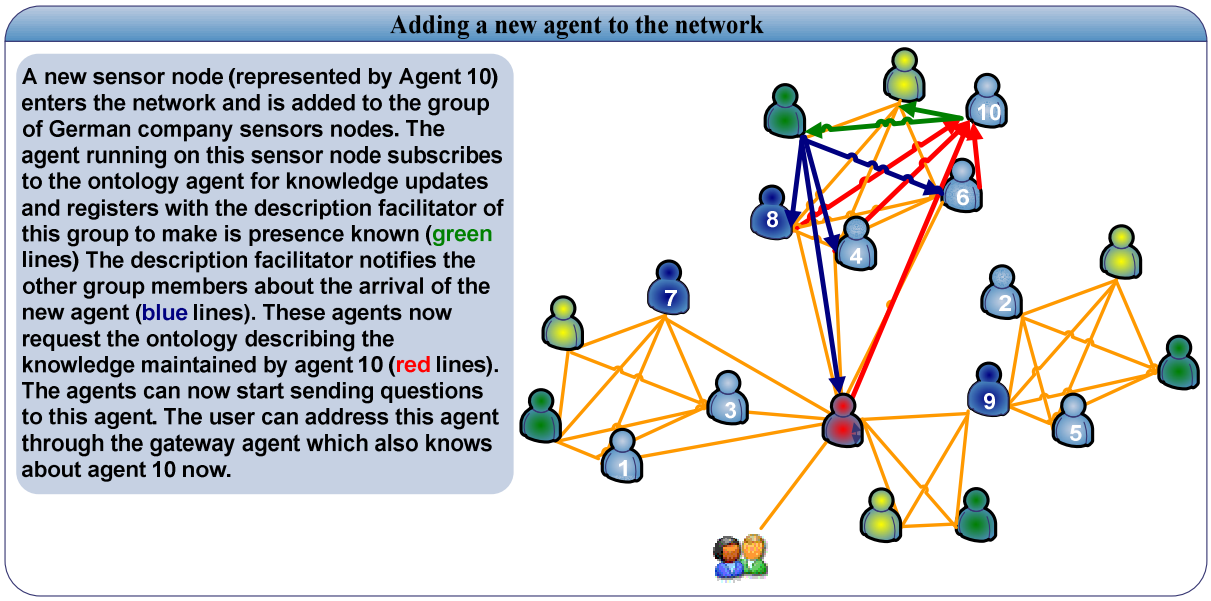


Figure 24: knowledge sharing: adding a new agent to the network

Our final experiment was to pose a question to the network which is based on an old ontology and thus contained deprecated resources in the query. In Appendix B in the ontology describing the knowledge base of resource agent 1, an example of annotating deprecation in an ontology is given. Both resource agents 1 and 3 have this same deprecation knowledge in their ontology. Figure 25 shows an example question which uses the deprecated URI sensors:PSensor to ask a question about pressure measurements.

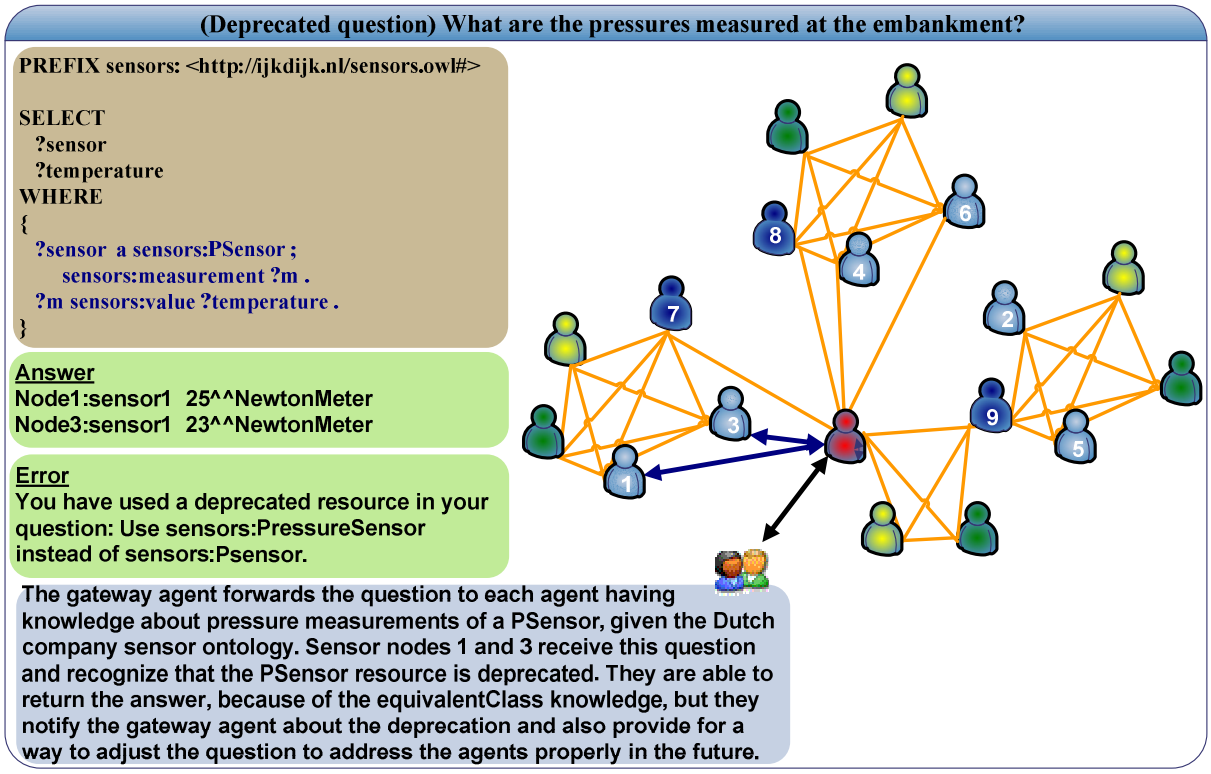


Figure 25: example of deprecated question: what are the pressures measured at the embankment?

Due to the fact that the resource agents know about an equivalent class of PSensor they are able to still answer the question. However, they alert the user about the deprecation with an error message. This error information can be used to change the question in case the resource agent does not support the deprecated annotation anymore in the future. Currently, the error message has to be interpreted by a human operator that maintains the resource agent. The operator can then change the private subscriptions of the resource agents in case they contain deprecated resources. Better would be to allow the agent to understand the error message and make the changes itself autonomously.

With the deprecation error messages we show that the framework supports the recognition of inconsistencies in shared knowledge (requirement 6).

5.1 Evaluation

In Section 4.1 a specification was given of the functional and non-functional requirements for a framework that could implement a distributed semantic sensor network that addresses all the challenges that were presented in the introduction of this report. In the discussion of the experiments with the framework was already mentioned how the functional requirements were met. The non-functional requirements however, can only be evaluated with an implementation of the framework, such as our implementation of the IJkdijk scenario. We will discuss in short how the framework addresses the non-functional requirements.

Usability requirements

A user that wants to question the network should be able to easily formulate a question based on the semantics, that describe the knowledge made available by the sensor nodes in the network. The returned answer should be displayed in an easy to understand format.

With the simulation it is possible to view the knowledge that is available in the network and create SPARQL queries about that knowledge. The network answer can be converted to a wide number of formats such as XML, RDF, N3, JSON and other more readable formats.

Efficiency requirements

Due to the possibility of adding resource poor nodes to the network, the chosen architecture should be lightweight in processor usage, memory, disk space and bandwidth. Moreover, it should be possible to dynamically assign functionality to sensor nodes.

In theory the agents are capable of running on any sensor node that supports Java. However, given the fact that OWL reasoning and rule reasoning are resource expensive processes, this might be an issue in practice with the current implementation of the framework. Dynamic assignment of sensor node functionality is possible by sharing knowledge between agents and to assign aggregation functionality to an agent.

A question asked to the network should return an answer within an appropriate amount of time. This reply-time depends on the complexity of the question but should depend as little as possible on the size of the network.

The semantic routing in the framework is very efficient since questions are directly routed to only those sensor nodes that are capable of answering the question. However, the bottom-up reasoning with ontologies, rules and knowledge base is still a slow process. This degrades the question answering time of the network, when new knowledge is added to the knowledge base through the sensors. Moreover, the explosion of questions in serial question federation, due to the generation of new triples by rule reasoning, is currently reducing the performance of the question answering in the network. The simulation however is running a single machine, currently deployed with 20 agents. It should be noted that in a distributed environment these agents are also distributed over multiple machines, reducing that resource requirements for each.

Reliability requirements

Whenever a question is asked to the network a reply should be given. Questions asked to the network should be replied to with the correct answer or at least an approximation of the correct answer. If a question cannot be answered the user should be notified about this as well.

With our implementation of the framework a question is always answered, either with an empty result set, a result from one or more sensors, or a message about deprecation in the knowledge the question is referring to.

Portability requirements

Since different parties add custom sensor nodes to the network, the chosen architecture has to be platform independent.

The current implementation of the framework is in Java, so any platform running Java should be able to run the framework.

External requirements

Interoperability requirements

Due to the nature of sensor networks, different sources of information must be queried. These resources could be supplied by a number of parties and presented using different formats. It is required that the chosen architecture is able to retrieve information in a wide number of formats and still be able to interpret the sensor data at different sensor nodes.

The current implementation allows the exchange of questions in SPARQL format only. The answers to questions can be sent in either Notation 3 or RDF/XML format.

It should be possible to extend the sensor network by adding new sensor nodes. This could implicate that information available in the network also changes. All parties and nodes have to be notified in case of changes to the knowledge available in the network.

Adding a sensor node, represented by an agent, results in a notification about this event to the other agents in that group. The knowledge of this new agent can be used as soon as the other agents have requested the ontology that describes the knowledge of this agent.

Privacy and Security requirements

Sensors can measure data which should not be made public. The architecture should make it possible to specifically state who should have access to specific knowledge. Furthermore, encryption of transmitted data should be possible in the chosen architecture.

Currently, grouping agents in a clustered network architecture is the only option to limit accessibility of agent knowledge. There is no access restriction on a knowledge level. Encryption in the framework is possible by adding the JADE security plug-in, which allows sending encrypted messages. This is currently not a part of the framework.

The proposed framework has been evaluated in an case study with a simulation of the IJkdijk. The challenges that were defined in the introduction of this report are being addressed by this framework and we have also explained how the non-functional requirements are met in the current implementation of the framework. During the design of the framework and the evaluation of the implementation some interesting findings were done, which will be discussed in the next section.

6 Discussion

The design of a distributed sensor network framework has been presented and we showed how the framework addresses the challenges introduced in the introduction section, by evaluating it against the IJkdijk scenario. During the design and evaluation process we have come across a number of issues that are worth going into a little deeper. These challenges are discussed in Section 6.1 on possibilities and limitations of the proposed framework. In Section 6.2 we discuss how our framework relates to other frameworks that where proposed by related research.

6.1 Possibilities and limitations

The challenges that we addressed are used as a guideline to present the possibilities and limitation of our framework in this section. As a reminder, the challenges were:

1. Construction of a distributed network architecture; how to facilitate communication among sensor nodes in a distributed environment
2. Facilitate the integration of sensor data; how to enable data integration by adding semantic web technologies to the sensor network domain and improve upon existing integration solutions
3. Question federation; how to split and federate a semantic question to enable the integration of distributed sensor data
4. Sharing of and reasoning with semantic knowledge; how can knowledge sharing allow a sensor network to operate in a dynamic environment and how reasoning with shared knowledge can increase knowledge at sensor nodes

Construction of a distributed network architecture

Our framework is specifically well suited for a distributed sensor network architecture. A distributed architecture suits the IJkdijk scenario well. For each sensor network scenario it is up to the designer to decide the level of distribution for the network. The following findings can help to support this decision:

- Are all the sensor measurements in the network relevant for a user or application? If so, this pleas for a more centralized network architecture. Since all the sensor data would have to be transported trough the network anyway, it is cheaper to send the data to a centralized location and execute all the questions on that centralized repository. In the evaluation we have seen that splitting a semantic question and reasoning with knowledge to support distribution can be a burden for speed in question handling in the network, especially in the case of serial federation. There is thus a trade-off between the need to keep data close to the source (the sensor) and the speed and ease with which this data can be accessed.
- The requirement to enable multiple parties to maintain sensors in the network and add sensor nodes to the network is a reason to use a distributed architecture, to allow each party to annotate sensor measurements with their own ontology. We have experienced that describing ontology mappings is possible, but describing a higher cardinality mapping is still incomplete with current OWL reasoning techniques (Section 2.2.3). Annotating data with different ontologies thus restricts the integration of sensor measurements.
- A sensor network designer should decide whether to use repeating requests to sensor nodes, or event subscriptions. Repeated requests result in more network traffic, since the same question is continuously sent over the network. Event subscriptions require more processing capacity of sensor nodes. These nodes have to monitor changes and trigger existing event subscriptions, which is an expensive process. The trade-off between repeated requests and event subscriptions mainly depends on the number of changes in sensor measurements at the sensor nodes. If the rate of changes in sensor measurements is very high, then using an event

subscription approach is very expensive; the processing at the sensor nodes is demanding and the number of event notifications is still very high. If the rate of changes is low, then there is considerably less network traffic when using an event subscription approach.

Shared knowledge through data integration

With our proposed framework we have introduced a number of ways to improve upon existing data integration solutions for integrating distributed data. Adding semantics to sensor data has proven to be worthwhile when it comes to the possibility of federating a semantic question to sensor nodes that can actually answer the question (semantic routing). Using data conversion in question handling further improves data integration of distributed data. How and when to start annotating sensor data with semantic metadata is a design decision, which depends on the requirements of the network. The following findings can help support these decisions:

- Data integration would not be so difficult if all the nodes in the network annotate their sensor measurements with the same ontology. Reusing existing ontologies in a sensor network is a good idea and improves data integration capabilities of the network. A network designer should thus consider using already existing ontologies. Sharing existing ontologies with the network enables other parties in the network to build mappings to these ontologies, or annotate their own sensor data with these ontologies.
- Annotating at a very low level, meaning the smallest sensor measurements are given a semantic meaning, has some consequences. By annotating at the lowest level of sensor measurements it becomes possible to ask questions about these measurements, making the answering capabilities of the network very precise. However, in a situation where questions about the smallest measurements are not being asked, annotating at the lowest level results in a too expressive knowledge base, slow reasoning and worse data integration. When to start annotating semantic data thus depends on the questions that are to be answered by the network. It can be worthwhile to consider adding semantics to sensor measurements only after a processing phase in which the sensor data is changed and combined at the sensor or sensor node itself.

The SPARQL query language is currently very limited. There are no possibilities to do any computations on collected results with built in methods, such as calculating the mean or running average of a series of values. For now, the unprocessed sensor measurements have to be collected locally and an application should then do the processing to calculate these values. An alternative is possible until the SPARQL language is extended: by allowing pre processing at the sensor nodes it would become possible to generate knowledge in the network, making it unnecessary to collect all the raw measurements before processing on the data. A sensor could always keep a record of its latest measurement, but also a list of previous measurements. A sensor could supply the network with pre processed values, such as the highest or lowest measurement, or maybe a running average of measurements. Such pre processing at sensor nodes is partially possible with a rule system. However, a more extensive processing system should be introduced to make this flexible.

We have introduced ontology versioning as a method for sensor nodes to know whether questions about local knowledge are still valid. The version number of the expected ontology is sent with a question and is checked against the latest ontology. If the versions are not equivalent, then the sensor node or user that posed the question is notified and can request the latest ontology from this resource agent. OWL makes it possible to annotate deprecation in an ontology, but it is not possible to annotate the changes per version. Even more, it is not possible to annotate additions or removals of concepts in the ontology. Currently, changes in ontologies have to be manually incorporated into private questions and subscriptions of resource agents by a human operator. Wider support for deprecation and versioning is necessary to enable agents to automatically change questions and subscriptions given a new ontology.

Finally, a remark about using shared ontologies. Currently, OWL ontologies are only capable of fully importing shared ontologies. This means that all the classes and properties of a shared ontology are included in the private ontology of a sensor node. This is not always the desired behaviour. An ontology could describe some property or class that we would like to reuse, but others

that are not used or are conflicting with the private ontology should be disregarded. A proposal for a new OWL 1.1 specification includes E-connected ontologies, in which it is possible to link structures of one ontology to structures within another. This would help reducing the size of ontologies and decrease the time needed for OWL reasoning, which is an expensive operation. We have not evaluated the use of E-connections in ontologies, since OWL reasoners such as Pellet do not yet fully support this feature.

Question federation

Question federation enables the querying of multiple distributed sensor nodes, in which a question can be split into sub-questions that are answered by different nodes. Currently, the question splitting algorithm as we have presented it is capable of splitting semantic SPARQL questions. This is quite sufficient for the way we have modelled our sensor data and the type of questions that were asked to the network. However, a knowledge base storing sensor data in RDF can be very complex. There are structures such as lists and bags that we have not used. Moreover, knowledge can be described with different degrees of cardinality. As a simple example, a speed sensor could annotate a movement of an object in 3D space with a single value, whereas another speed sensor could annotate this movement with three different values, one for the X, Y and Z direction of the object. Integrating these measurements with a SPARQL question is currently not possible with the proposed question splitting method. The reason why is because currently SPARQL questions are formulated using a single ontology and ontology mappings enable this same question to be sent to sensor nodes that annotate measurements with a different ontology. These sensor nodes use the ontology mapping to generate new triples bottom-up that describe their knowledge base in terms of another ontology. Combining sensor measurements of distributed sources that describe their knowledge with a different cardinality requires the rewriting of a SPARQL query in terms of the other ontology. Question rewriting has another advantage in that it removes the necessity in our framework for sensor nodes to do bottom-up OWL reasoning to add triples to their knowledge base to describe the knowledge in terms of other ontologies.

This brings us to the point where we suggest that for fully supporting the integration of distributed data it is required to use a combination of both question splitting and question rewriting techniques. The question splitting algorithm as we have proposed it is then used to split the question in sub-questions and determine the sensor nodes where to send these sub-questions. Each sub-question should then be rewritten (translated) in terms of the ontologies of the target sensor node, by using a question rewriting technique such as the MiniCon Algorithm [56]. This enables combining knowledge that relates in a many-to-many cardinality and also allows for data conversion of many-to-many relations, such as in the speed sensor example earlier.

We have seen that questions can be federated. In practice, subscription triggers can also be federated in the same way. This would allow to be alerted about events that are triggered by multiple sensors. In the current framework this is however not possible, due to synchronisation issues. Decisions have to be made to decide whether two sensor measurements occur simultaneously and thus constitute the same event, or that they belong to different events. An important aspect of simultaneous measurements is the time interval within which the measurements are recorded. This was out of the scope of the current research project and could be investigated in future work.

Another weakness in the framework is that it is assumed that each sensor node that is sent a question or sub-question will answer this question. The execution of the split question waits for each sensor node to send a reply, which could be an empty answer as well. However, in a real world sensor network it could occur that sensor nodes are unreachable or that communication between sensor nodes fails. Failing communication can partially be handled with interaction protocols such as the FIPA protocols. The FIPA protocols include a time to live parameter for a conversation. We have not utilized this, due to prioritizing other framework requirements, but doing so in future work is a good idea. When the time to live has expired the question can be sent again, or an empty answer from the sensor node can be assumed.

We have mentioned security as one of the non-functional requirements for a sensor network. By introducing semantics into the sensor network domain a number of interesting opportunities arise when it comes to security issues. For example, it becomes possible to analyze a semantic question and based on its content determine whether the sensor node that sent the question is allowed to pose this

kind of question. This introduces the possibility of filtering both questions and answers in a sensor network based on content. We have not done any research on this topic and it is thus open for future research.

Finally, the current question splitting algorithm does not support optional statements in a SPARQL query, which is part of the SPARQL specification.

Sharing of and reasoning with semantic knowledge

Knowledge sharing is a fundamental aspect of our framework. It allows a sensor network to be usable in a dynamic environment. For sharing knowledge we have used existing FIPA communication protocols. These protocols have proven to be helpful. A protocol constrains the interaction between sensor nodes and it enables handling multiple conversations in parallel. We have been able to use the protocols without the necessity of changing the standards that are proposed by FIPA. We have however used the possibility to add user defined parameters to FIPA ACL messages, to exchange grouping, question type and question path information among agents in a conversation, all within the scope of the FIPA specification.

In Section 2.2.3 we have made the distinction between infrastructure ontologies and hierarchical ontologies. We have used infrastructure ontologies to describe the knowledge of each sensor node and hierarchical ontologies to describe the relations of other types of knowledge, such as location along the IJkdijk embankment. We have experienced that when it comes to sharing these types of knowledge there is a difference (See Section 5, page 68). Hierarchical ontologies can be freely shared by all the sensor nodes. They introduce additional knowledge for sensor nodes, but do not effect the question federation process particularly. Infrastructure ontologies on the other hand can not be as freely shared. An infrastructure ontology is very specific for a sensor node, since it describes the knowledge of that node. Would this knowledge be shared in the network, then the question federation algorithm would think every sensor node is capable of answering a question. We can learn two things from this. First, infrastructure ontologies, describing the sensor measurements and other knowledge of sensor nodes, should be very specific and contain no additional knowledge that is not needed to describe the way in which the knowledge base of a sensor node is annotated. Second, when adding new knowledge to the network that is not saying anything about the knowledge base of a sensor, then it should be added to a separate ontology, which can be imported by the infrastructure ontology. Such hierarchical ontologies can make question federation more specific, as we have seen in the IJkdijk scenario by introducing rules and an ontology describing locations in the network, but the question federation itself is not effected by hierarchical ontologies.

When it comes to reasoning with semantic knowledge there are a number of issues that should be considered. Our framework uses a bottom-up reasoning approach that pre generates new semantic knowledge before handling questions. We will compare this to a top-down approach by addressing these issues bullet wise.

- Bottom-up reasoning requires a sensor node to perform OWL and rule reasoning every time the knowledge base changes. This implicates that when a sensor adds a measurement to the knowledge base, new knowledge is inferred and added to the knowledge base as new semantic triples. This approach should be used only when the number of changes to the knowledge base is low.
- Top-down reasoning is the process where reasoning is performed on the knowledge base when a question is received by the sensor node. Reasoning could then be performed on the complete knowledge base before executing the question, or the question could first be interpreted and reasoning is performed on only that part of the knowledge base that applies to the question. Although difficult to implement, such a process would be very efficient compared to bottom-up reasoning in a sensor network where there is a large number of updates to the knowledge of sensor nodes and a relatively small number of questions about this knowledge.
- The trade-off between bottom-up and top-down reasoning thus depends on the number of changes to the sensor nodes knowledge and the amount of questions about this knowledge. In between solutions can be thought of that perform bottom-up reasoning for knowledge that is very static and top-down reasoning for knowledge that changes a lot.

Knowledge sharing combined with knowledge reasoning is a very powerful addition to a sensor network that should operate in a dynamic environment. By distributing new knowledge in the network it becomes possible to increase the knowledge of existing data, by reasoning with new rules, or by being able to convert data with data type conversion. Currently, it is up to a user to introduce new knowledge to the network. This knowledge is then sent to a group, a number of groups or to the whole network.

Bottom-up reasoning adds triples to a knowledge base of a sensor node. This has implications on questions that are posed about this knowledge base. We have seen that 'derived results' are also selected by SPARQL queries, even though sometimes these results are just a duplication of other results. This affects the efficiency of question handling, especially in cased where questions are split. A question rewriting approach could be a possible solution to this problem, as it makes it unnecessary to generate all knowledge bottom-up. However, sometimes it could be useful to be able to access these derived results. The SPARQL language does not incorporate such decision making in a query. A possible solution would be to extend the SPARQL query language to add reasoning capabilities to the query language.

The introduction of a new rule that adds triples to a knowledge base currently requires the ontology of the sensor node that receives this new knowledge to be changed so that it describes this new knowledge. This is required for other agents to be able to know that the agent has this new knowledge. Changing the ontology is an extra operation for a human maintainer of a sensor node. It would be better if a rule would also describe its effect on the knowledge base and thus should provide for a way to change the ontology describing this knowledge base.

The speed of question handling in the network could be increased considerably if the distribution of new knowledge in the network would be targeted to those sensor nodes only, that have use for the new knowledge. Each additional rule with which to reason on a knowledge base is an extra processing step. Thus, the less rules the better. The same holds for importing shared ontologies and reasoning over those with OWL reasoning. The less triples to reason with, the faster the process. Thus, distribution based on the content of new knowledge could reduce the number of rules that should not be known by each sensor node. This could be achieved by comparing the content of a knowledge update with the ontologies describing sensor nodes. An overlap in resource URIs means that that sensor node should know about the knowledge update.

An other interesting idea is that the network itself could infer new knowledge and rules by applying data mining techniques on distributed knowledge, or analyzing user questions. Such a capability would make a sensor network much more autonomous, reducing the maintenance efforts for the network.

6.2 Relation to related research

Different research groups have proposed data integration solutions for sensor networks, like ours. In Section 1.4 we introduced the different frameworks for sensor networks. We have grouped these solutions by: database oriented sensor networks, agent based sensor networks, and semantic sensor networks. We will discuss these different approaches and how they address the challenges for sensor networks that we have identified (See Section 1.3). We compare our framework to other frameworks and mention both advantages and disadvantages of our approach.

Database inspired sensor networks

Database inspired frameworks, such as TAG and TinyDB [13, 20] and Cougar [30] use a flat homogenous sensor network architecture to meet the distributed architecture challenge. It is different from our approach in that these frameworks require the construction of a routing tree to send questions to sensor nodes. In our agent-based distributed network architecture, the agents can be fully connected to one another and questions are dynamically federated through the network, based on local knowledge. This makes our approach more flexible. However, predefined routing trees in database inspired frameworks do make the sending and retrieval process of a question faster, since less time is spent in analyzing a question and where to send it. These type of frameworks typically support both direct questions to sensor nodes and event notification, as does our framework.

Database inspired frameworks address the data integration challenge by simply making the assumption that every sensor node in the network annotates sensor data identically. All sensor

measurements are thus described by the same properties and all sensor measurements of a type of sensor have the same unit. Such an assumption makes that there is no requirement for adding semantic web technologies to sensor data and sensor data can simply be stored in a shared database table format. This sensor identity assumption is useful in situations where the complete network is designed by a single party and few changes are made to the way in which sensor data is stored and annotated. Our more dynamic approach to annotation, by using semantic web technologies shows its strength in situations where different parties annotate semantic data differently and mappings between sensor annotations are required.

Database inspired frameworks partially address the question federation challenge by using a query language. However, questions are not split into multiple sub-questions. The complete question is federated to each node in the network, which makes sense, since all sensor nodes are assumed to be identical. If we compare this to our question federation approach we see that our framework is capable of answering questions for which the knowledge is distributed over different sensor nodes that each have different types of sensors connected to it. This makes our framework more useable in heterogeneous sensor networks.

None of the database inspired frameworks have incorporated knowledge sharing in the framework. The fact that sensor nodes in the network are all identical reduces the use of knowledge sharing in a sensor network. However, in large networks which are difficult to maintain, due to the huge amount of sensor nodes, it can prove worthwhile to use knowledge sharing to distribute rules and other types of knowledge in the network, to change the behaviour and answering capabilities of the network after deployment. In short, the advantages and disadvantages of database inspired frameworks compared to our framework:

Advantages

- Fast question federation by utilizing routing trees
- No need for semantics and expensive reasoning processes because the network is homogeneous

Disadvantages

- Can only be used for homogeneous sensor networks
- Uses flooding and routing trees due to the lack of semantics for question federation.
- No knowledge sharing to change the behaviour of the sensor network

Agent based sensor networks

Agent based frameworks such as IrisNet [19], AIGA [14], the Biswas and Phoha architecture [18] and the SWAP framework [15], use a multi-agent systems approach to address the distributed sensor network architecture challenge. The strength of this approach is that knowledge exchange in the network is flexible. Agents can have conversations with one another, to exchange sensor data and ask questions about sensor data to one another. We have fully utilized the advantage of these framework, by using the multi-agent paradigm as a bases for our distributed sensor network architecture.

When it comes to data integrations, existing agent based frameworks turn out not to be very flexible. The agent frameworks lack semantics in the knowledge base containing the sensor data. This is why the federation of questions about sensor measurements of other sensor agents and the interpretation and integration of the answers have to be pre-programmed into the agents. These agent based frameworks thus generally lack flexible data integration, making it impossible to ask agents in the network a question about sensor measurements which is not pre-programmed into the agent. Pre-programming question handling into the agents does make federation and interpretation of question cheaper than the more costly question federation process we have incorporated into our framework. The SWAP framework uses ontologies to describe sensor knowledge of different agents in the sensor network. The integration of sensor data in SWAP is more flexible than the other agent based frameworks, however, each question still has to be pre-programmed into the SWAP framework agents.

In principle, agent based frameworks are heterogeneous. Each agent in the network has different sensors. Questions about these sensor measurements can thus be federated by the agents. Question federation is possible in these agent-based frameworks, but due to the lack of semantics the federation process has to be pre-programmed into the agent as well. Each agent thus has to know in

advance the knowledge of its neighbours and it knows how to collect and interpret this knowledge for each possible user question. The use of semantics in our framework removes the necessity to pre-program all possible questions into the agents. This makes our framework more flexible.

Knowledge sharing is not incorporated into the design of these agent based frameworks, mainly because the agents are pre-programmed. In short, the advantages and disadvantages of agent based frameworks compared to our framework:

Advantages

- No costly question federation process, due to pre-programming questions and answer interpretation

Disadvantages

- Questions have to be pre-programmed due to the lack of semantics
- Pre-programming the agents makes these frameworks unsuitable for dynamic sensor network environments
- No knowledge sharing to change the behaviour of the sensor network

Semantic sensor networks

Semantic based sensor networks, such as ISENSE [17] incorporate semantic web technologies to annotate sensor data and enable integration of distributed sensor data.

ISENSE is a database inspired framework. But differs from other database inspired frameworks in that semantics form the basis of the complete framework. The ISENSE framework uses semantic web technologies to annotate sensor data semantically. ISENSE uses the SPARQL query language to address distributed semantic data sources and integrate sensor measurements. The use of semantics with RDF and OWL and the SPARQL query language for data integration is comparable to how our framework uses these techniques. The main difference with our approach is that our framework makes it possible to do additional in network conversions of data from one unit to another. This makes it possible to compare sensor measurements in the network, whereas ISENSE leaves conversion up to the end user.

When it comes to question federation there are some differences between ISENSE and our framework. ISENSE uses a question rewriting algorithm to rewrite SPARQL queries into the correct semantics that can be understood by different distributed sensor nodes. This framework makes the assumption that all the sensor nodes are homogenous. The sensor nodes thus describe the same types of sensor measurements, but each sensor can annotate its sensor data given a different ontology. Our framework does not assume homogeneity. Distributed sensor data of heterogeneous sensor nodes can be integrated by using a question splitting algorithm which uses ontologies of sensor nodes to split a SPARQL question into sub-questions.

ISENSE does not use knowledge sharing, because it is deployed only in sensor network environments in which the topology, the type of sensors and the amount of sensors is static.

Advantages

- The use of semantics and semantic web technologies enables these frameworks to be flexible when it comes to addressing sources in the network which use different semantics to annotate their sensor data
- Question federation by rewriting questions

Disadvantages

- The ISENSE framework does not allow data integration of heterogeneous sensor data
- Does not allow online changes, such as the addition or removal of sensors, or changes in the sensor data annotation
- No knowledge sharing to change the behaviour of the sensor network

In general, none of the existing frameworks provide for a way to flexibly integrate sensor data of heterogeneous sensor nodes in the network. We believe the use of semantics in the sensor network domain is useful whenever the sensor nodes in the network are heterogeneous. Sensor nodes can use different ontologies to annotate the infrastructure of the sensor data or they can have different types of sensors connected to them. Both make sensor nodes different from one another, making it difficult to integrate their measurements with measurements of other sensor nodes by fully pre-programming the interaction between sensor nodes.

Our work in a wider context

For readability this report has mainly discussed the application of our proposed framework in a sensor network environment. By taking some distance to the presented work we can see that it is applicable to a much wider range of problems. As there is a trend to combine more and more information in organization networks, the world wide web, healthcare and customer services, there is also an increasing need to be able to integrate different types of data. Our findings of data integration are applicable to any situation in which data should be combined. Our work on ontology mapping, question federation, knowledge sharing and data conversion make it possible to increase data integration in these domains as well.

We have discussed sensors and sensor nodes as the only data sources. However, we could use the same principles to link other data sources, such as databases, web services, web feeds or user applications. Each data source should then be represented by an agent that can answer questions about its knowledge. This eliminates the requirement to rewrite all the existing software and applications in the real world to fit this particular need. An agent can operate on a shared knowledge base with an application, sensor or database.

The knowledge infrastructures which we have seen in the sensor network domain of the IJkdijk were kept simple. With RDF it is actually possible to describe very complex knowledge structures. An example of this is a description of a traveller movement process on an airport. Such a process describes how the current state of a traveller translates into the remaining steps to take for such a traveller. By describing complex knowledge this way it becomes possible to distribute knowledge on traveller whereabouts and airport processes, to create a network that can help guidance of travellers on an airport. Possible questions of a traveller could then be: where am I, what are my next steps to take? And for an airline company: which of my travellers are going to be late and how long does it take them to get to the departure gate?

When it comes to the use of agents, we could translate the concept of agents to a web service oriented architecture which has the same characteristics as distributed agent systems. However, there is a requirement for web services to work with subscription protocols and to facilitate knowledge sharing, which is not a common capability of existing web service implementations.

7 Conclusion

This project was inspired by the latest research on integration of distributed sensor measurements and the practical problems that have arisen in the IJkdijk sensor network project. The goal was to see whether semantic web technologies could be used in a sensor network domain in order to overcome the data integration problems of a network of distributed sensors. We have seen that the use of semantics in sensor networks is advantageous if sensor nodes in the network are heterogeneous, meaning that they have different types of sensors connected to it, that measure different phenomena. Moreover, if different parties add sensors and sensor nodes to the network and store sensor measurements differently, then the use of semantics allows the integration of these differently annotated sensor data.

A number of challenges were identified as a result of open research questions in the field of sensor networks and practical issues from a real world sensor network scenario. The main challenges for integrating distributed sensor measurements that we have addressed are:

- Construction of a distributed network architecture; how to facilitate communication among sensor nodes in a distributed environment
- Facilitate the integration of sensor data; how to enable data integration by adding semantic web technologies to the sensor network domain and improve upon existing integration solutions
- Question federation; how to split and federate a semantic question to enable the integration of distributed sensor data
- Sharing of and reasoning with semantic knowledge; how can knowledge sharing allow a sensor network to operate in a dynamic environment and how reasoning with shared knowledge can increase knowledge at sensor nodes

Based on literature and our own findings in the domain of sensor networks we constructed a framework that addresses these challenges for a distributed semantic sensor network. In a design-science research process a framework was constructed based on a multi-agent network architecture, semantic web technologies for data integration and an innovative proposal for question federation. The framework facilitates knowledge sharing, allowing the sensors in the network to incorporate changes in the environment and human induced changes, such as the addition or removal of sensor nodes, or changes in knowledge structures and datatypes.

A proof of concept was given by implementing the IJkdijk scenario with the proposed framework. We have conducted an experimental evaluation of the framework in a case study and identified strong and weak points of the current design. Currently, the question federation of semantic questions works well and can be used to ask questions about sensor measurements of a large number of distributed heterogeneous sensors. Together with the data conversion methods, this has improved data integration capabilities of sensor networks, compared to existing solutions. It is now possible to address multiple distributed heterogeneous sensors with a single question, without having to worry about how the question is split into sub-questions and which sensors should be sent these questions. Moreover, by introducing conversion of sensor data at the sensor node it is now possible to integrate sensor data in the network, whereas before it was only possible to integrate the data as a post-processing step after collection of all the measurements. Environmental changes, deprecation of knowledge and change of knowledge representation structures and data types are the main causes of reduced usability of other sensor network frameworks. The introduction of knowledge sharing in our sensor network framework has allowed us to tackle these problems. Distribution of rules, shared ontologies, mapping ontologies and data conversion knowledge allows the sensor network to be used in such a dynamic environment.

We already mentioned that the current semantic web technologies such as SPARQL and OWL are limiting the possibilities for using these techniques in a sensor network domain. A more expressive SPARQL specification should include aggregate functions to calculate things as running average over

multiple results from a query, which is currently not possible in the network. Moreover SPARQL should include grouping to enable the user handle query outcomes more diversely, for instance, to handle derived results in the question outcome. The OWL specification is currently not sufficient in a number of ways. First, it lacks proper versioning capabilities, making it impossible for a sensor node to know how exactly an ontology changed since the previous version. Second, OWL only allows to import other ontologies completely. E-connection in OWL would make it possible to link ontologies on instance level, greatly increasing the speed of OWL reasoning, which currently makes our framework slower than desired. Finally, describing a high cardinality mapping is not fully supported with OWL. This makes it impossible to fully map two ontologies that have complex and distinct infrastructures in their knowledge base.

We have identified a number of issues upon which future research could focus. When it comes to the splitting and distribution of semantic questions we have to conclude, that for dealing with higher cardinality relations between knowledge infrastructures of sensor nodes, a question federation method has to be created that combines both question splitting and question rewriting techniques (See Section 6.1, question federation). Future research might include looking into the possibilities of combining our question splitting algorithm with the MiniCon question rewriting algorithm used by Dimitrov et. al. [17].

The reasoning processes of OWL reasoning and rule reasoning are very expensive. We have a number of ideas to reduce the costs of these reasoning processes in a distributed environment. Content based distribution of knowledge reduces the number of 'non applicable' rules at sensor nodes, reducing the required processing resources for rule reasoning. Rewriting semantic questions with ontology mappings eliminates the need to generate all possible mappings bottom up at sensor nodes. This reduces the OWL reasoning requirements of sensor nodes. Future research might look into the possibilities of content based knowledge distribution and question rewriting and with this make using a distributed sensor network architecture feasible for many sensor network scenarios in which distributed storage of data (as opposed to centralized storage) is a requirement.

Future research could also look into the possibilities of making the sensor network more autonomous. Extended versioning capabilities in OWL are necessary to allow sensor nodes to automatically change questions and subscriptions given a changed ontology. Further more, executing a rule on the knowledge base of a sensor node might effect the knowledge of a sensor node. This change has to be incorporated into the infrastructure ontology that describes the knowledge of this sensor node. Rule knowledge should thus include both the rule and the change that it implies on an infrastructure ontology, so that the sensor node can change its own ontology, without human intervention, when adding a rule to its knowledge. Also, the sensor network could analyze question distribution to optimize the behaviour of question handling in the network. The analysis of repeated questions in the network could result in the autonomous creation of aggregating resource nodes in the network to cache results. A future extension to the current framework could include pre processing knowledge and capabilities to sensor nodes, to allow more flexible in-network computations of aggregated knowledge, increasing the knowledge at sensor nodes. Making a sensor network more autonomous implicates that fewer human intervention is needed to keep the network running and usable and also makes it easier to get answers that require less post-processing efforts, from the network.

Finally, measuring simultaneity of sensor measurements is currently an open research topic. Neither us, nor earlier research on this topic has investigated how a question about simultaneous events measured by different sensors can be answered. We have not focussed on timing aspects in our work. It is interesting to see whether adding a timestamp to sensor measurements and the use of time ranges in semantic SPARQL questions is sufficient to tackle this issue.

7.1 Contributions

This research project has made the following contributions to the sensor network and semantic web domain:

- The use of semantic web technologies in the sensor network domain was evaluated by doing a case study on a real world sensor network scenario

- We have used an multi-agent architecture to implement an architecture for a distributed sensor network and we evaluated the existing FIPA-protocols
- A question splitting algorithm was proposed with which becomes possible to integrate heterogeneous sensor data of distributed sensors, by splitting a semantic question and fusing the answers from distributed sources to form the answer to the question
- We have shown how data conversion at the sensor nodes makes it possible to integrate sensor data at the sensor node itself, instead of centralized by the user
- Knowledge sharing was introduced to sensor networks. We showed how sharing different types of knowledge among sensor nodes can change the answering capabilities of the sensor network

Appendix A Query splitting algorithm

```
Algorithm SplitQuery
1  Input: The list of triples contained in the query, (triples)
2  Input: A list of nodes and the ontology describing each node, (inode)
3  Output: A list of queries and the services where they are sent to, (queries)
4
5  nodes  ← inode, assigned ← [ ]
6
7  for each triple in the list triples, do
8    subject ← subject of the triple
9    predicate ← predicate of the triple
10    object ← object of the triple
11    if object is a class, do
12       nodes ← subset of inode in which this object is described as a class
13     bnodes ←set of nodes to which a triple was added with an equivalent variable
14              in object or subject position
15       if bnodes is not empty, do
16                nodes  ← nodes ∪ bnodes
17       fi
19    else the subject is not a class, do
20       nodes ← subset of inode whose ontologies describe the predicate of this
21                triple as a property
22       if the subject of the triple is bound, do
23        bnodes ← subset of inode whose ontology contains subject as an individual
24         if bnodes is not empty, do
25          rnodes ← nodes in assigned that also contain this subject in one of the
26                   assigned triples
27         if rnodes is not empty, do
28                     nodes  ← nodes ∪ bnodes
29         fi
30       fi
31       else the subject is not bound, do
32         for each service in the list nodes, do
33           domain ← domain of predicate in the ontology of service
34           dnodes ← nodes in assigned which where assigned a triple whose
35                    object is the same as the domain of this triple predicate
36           rnodes ← nodes in assigned which where assigned a triple whose
37                    range is the same as the domain of this triple predicate
38         bnodes ← dnodes ∩ rnodes
39         if bnodes is not empty, do
40            gnodes ← set of nodes in assigned to which a triple was added with an
41                     equivalent variable in object or subject position
42                    if gnodes is not empty, do
43                       nodes  ← nodes ∪ gnodes
44              fi
45         fi
46          for end
47       fi
48       assigned ← assigned ∩(triple, nodes)
49    for end
50
51    groups ← construct groups of triples that are sent together to service(s)
52    groups ← remove services from groups whose ontologies do not support
53           individuals in one of the triples in a group.
54
55   return queries constructed from groups + the remaining services to which they
56         are sent
```

- "←" is a loose shorthand for "changes to". For instance, "largest ← item" means that the value of largest changes to the value of item.
- **for each** is a loop which is ended by **for end**
- **if …, do else** is an if-else statement which is ended by **fi**.
- "**return**" terminates the algorithm and outputs the value that follows.

Example query for question federation algorithm

Ontology of sensor node 1

```
    @prefix : <http://ijkdijk.nl/service1#> .
@prefix sensors: <http://ijkdijk.nl/sensors.owl#> .
@prefix locations: <http://ijkdijk.nl/locations.owl#> .
@prefix owl: < http://www.w3.org/2002/07/owl#> .
@prefix rdfs: < http://www.w3.org/2000/01/rdf-schema#> .

sensors:TemperatureSensor a owl:Class .
sensors:Measurement a owl:Class .

 sensors:measurement a owl:ObjectProperty ;
    rdfs:domain sensors:TemperatureSensor ;
    rdfs:range sensors:Measurement .

sensors:value a owl:DatatypeProperty ;
    rdfs:domain sensors:Measurement .

sensors:location a owl:ObjectProperty ;
    rdfs:domain sensors:TemperatureSensor ;
    rdfs:range locations:Location .

:Sensor_1 a sensors:TemperatureSensor ;
    sensors:location locations:zoneA .

:Sensor_2 a sensors:TemperatureSensor ;
    sensors:location locations:zoneA .
```

Ontology of sensor node 2

```
@prefix : <http://ijkdijk.nl/service3#> .
@prefix sensors: <http://ijkdijk.nl/sensors.owl#> .
@prefix locations: <http://ijkdijk.nl/locations.owl#> .
@prefix owl: < http://www.w3.org/2002/07/owl#> .
@prefix rdfs: < http://www.w3.org/2000/01/rdf-schema#> .

sensors:PressureSensor a owl:Class .
sensors:Measurement a owl:Class .

 sensors:measurement a owl:ObjectProperty ;
    rdfs:domain sensors:PressureSensor ;
    rdfs:range sensors:Measurement .

sensors:value a owl:DatatypeProperty ;
    rdfs:domain sensors:Measurement .

sensors:location a owl:ObjectProperty ;
    rdfs:domain sensors:PressureSensor ;
    rdfs:range locations:Location .

:Sensor_1 a sensors:PressureSensor ;
    sensors:location locations:zoneA .

:Sensor_2 a sensors:PressureSensor ;
    sensors:location locations:zoneB .
```

Example SPARQL question

```
REFIX sensors: <http://ijkdijk.nl/sensors.owl#>
PREFIX locations: <http://ijkdijk.nl/locations.owl#>

SELECT
  ?location
  ?temperature
  ?pressure
WHERE
{
  ?tsensor  a sensors:TemperatureSensor ;
      sensors:location locations:zoneA ;
      sensors:measurement ?m .
  ?m sensors:value ?temperature .
}
```

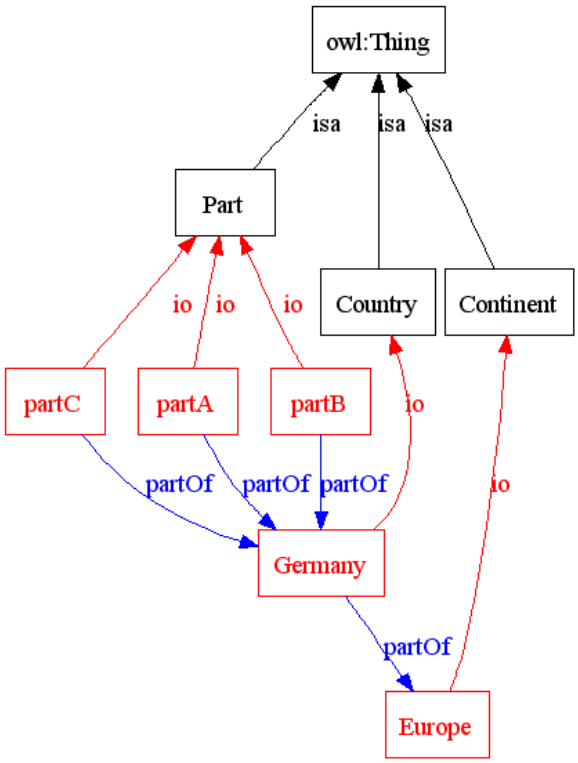
Algorithm applied to query, per triple

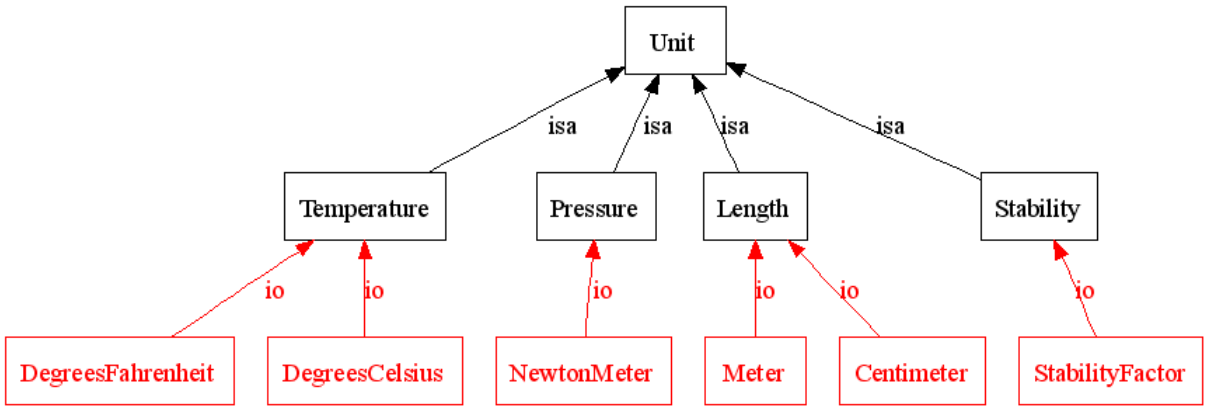
1. ?tsensor a sensors:TemperatureSensor ;
   L11: Object is a class
   L12: Sensor node 1 have class
   L48: Assign triple to Sensor 1
2. ?tsensor sensors:location ?location ;
   L20: Sensor 1 & 2 have property
   L31: subject is not bound (variable)
   L33: domain is sensors:TemperatureSensor or sensors:PressureSensor
   L34: sensor1 was assigned the triple 1, where the domain matches the object of triple 1.
   L39: bnodes contains Sensor 1
   L40: triples 1 and 2 share variable ?tsensor
   L43: possible nodes is restricted to Sensor 1
   L48: Assign triple to Sensor 1
3. ?tsensor sensors:measurement ?m .
   --- same as triple 2 ---
4. ?m sensors:value ?temperature .
   L20: Sensor 1 & 2 have property
   L31: subject is not bound (variable)
   L33: domain is sensors:Measurement
   L34: Sensor1 was assigned triple 3, where the domain matches the object of triple 4.
   L40: triples 3 and 4 share variable ?m
   L43: possible nodes is restricted to Sensor 1
   L48: Assign triple to Sensor 1

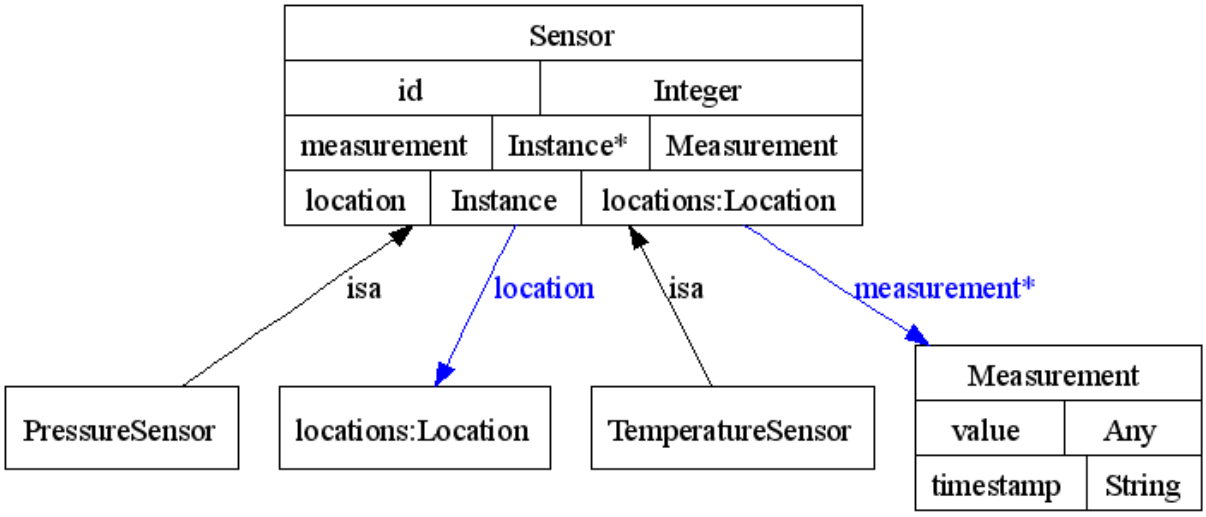
   L51: group all triples per sensor. All triples in this example go to Sensor 1 and are thus added to 1 SPARQL query.
   L52: Sensor 1 has a static sensor:location of locations:zoneA in its ontology. This query can thus be sent to Sensor 1.
   L55: return query + target (= Sensor 1)

Table 21: example of question federation algorithm applied to a question. L = line nr. in algorithm

Appendix B Example knowledge base and ontologies







Knowledge base of Resource Agent 1

```
@prefix sensors:  <http://ijkdijk.nl/sensors.owl#> .
@prefix xsd:     <http://www.w3.org/2001/XMLSchema#> .
@prefix units:   <http://ijkdijk.eu/units.owl#> .
@prefix rdfs:    <http://www.w3.org/2000/01/rdf-schema#> .
@prefix rdf:     <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix locations:  <http://ijkdijk.nl/locations.owl#> .
@prefix :        <http://ijkdijk.nl/resourceAgent1#> .

:Sensor_2
     a       sensors:PressureSensor ;
     sensors:measurement
           [ a       sensors:Measurement ;
             sensors:timestamp "2006-12-30T15:30:28"^^xsd:dateTime ;
             sensors:value "25"^^units:NewtonMeter
           ] .

:Sensor_1
     a       sensors:TemperatureSensor ;
     sensors:measurement
           [ a       sensors:Measurement ;
             sensors:timestamp "2006-12-30T15:30:28"^^xsd:dateTime ;
             sensors:value "25"^^units:DegreesCelsius
           ] .
```

Ontology describing the knowledge base of resource agent 1

```
@prefix sensors:  <http://ijkdijk.nl/sensors.owl#> .
@prefix units:   <http://ijkdijk.eu/units.owl#> .
@prefix rdfs:    <http://www.w3.org/2000/01/rdf-schema#> .
@prefix :        <http://ijkdijk.nl/resourceAgent1#> .
@prefix owl:     <http://www.w3.org/2002/07/owl#> .
@prefix locations:  <http://ijkdijk.nl/locations.owl#> .

<http://example.org/resourceAgent1>    a       owl:Ontology ;
     rdfs:label "resourceAgent1" ;
     owl:imports <http://ijkdijk.nl/sensors.owl> , <http://ijkdijk.de/locations.owl> ,
                 <http://ijkdijk.nl/locations.owl> , <http://ijkdijk.eu/sensormapping_dw.owl> ,
                 <http://ijkdijk.eu/units.owl> , <http://ijkdijk.nl/locationsmapping_dw.owl> ,
                 <http://ijkdijk.eu/sensormapping.owl> ;
     owl:priorVersion "20071921075757" ;
     owl:versionInfo "20072021085232" .

:Sensor_2    a       sensors:PressureSensor ;
     sensors:location locations:zoneA .

:Sensor_1    a       sensors:TemperatureSensor ;
     sensors:location locations:zoneA .

sensors:TemperatureSensor    a       owl:Class ;
     rdfs:subClassOf sensors:Sensor .

sensors:PressureSensor   a       owl:Class ;
     rdfs:subClassOf sensors:Sensor .

sensors:Psensor a owl:DeprecatedClass ;
    owl:equivalentClass sensors:PressureSensor .
```

Deprecation knowledge

Location ontology mapping of locations in the dutch company ontology and the water management ontology

```
@prefix d:      <http://ijkdijk.nl/locations.owl#> .
@prefix w:      <http://watermanagement.org/locations.owl#> .
@prefix xsd:    <http://www.w3.org/2001/XMLSchema#> .
@prefix rdfs:   <http://www.w3.org/2000/01/rdf-schema#> .
@prefix owl:    <http://www.w3.org/2002/07/owl#> .

<http://ijkdijk.nl/locationsmapping_dw.owl>  a      owl:Ontology ;
    rdfs:label "locationsmapping_dw" ;
    owl:priorVersion "20061230162400"^^xsd:string ;
    owl:versionInfo "20061231162400"^^xsd:string .

d:Continent  owl:equivalentClass w:Continent .

d:zoneB   =      w:partB .
d:zoneA   =      w:partA .
d:partOf  =      w:partOf .
d:Europe  =      w:Europe .
```

Appendix C Full results of questions

What is the water level at locations where the temperature is over 14, ordered by descending temperature?

temperaturesensor	location	level	temperature
resourceAgent6:Sensor_1	dlocations:Europe	"1.1"^^units:Meter	"77"^^units:DegreesFahrenheit
resourceAgent6:Sensor_1	dlocations:Germany	"1.1"^^units:Meter	"77"^^units:DegreesFahrenheit
resourceAgent6:Sensor_1	wlocations:Europe	"1.1"^^units:Meter	"77"^^units:DegreesFahrenheit
resourceAgent6:Sensor_1	dlocations:Europe	"1.15"^^units:Meter	"77"^^units:DegreesFahrenheit
resourceAgent6:Sensor_1	dlocations:Germany	"1.15"^^units:Meter	"77"^^units:DegreesFahrenheit
resourceAgent6:Sensor_1	dlocations:partB	"1.15"^^units:Meter	"77"^^units:DegreesFahrenheit
resourceAgent6:Sensor_1	wlocations:Europe	"1.15"^^units:Meter	"77"^^units:DegreesFahrenheit
resourceAgent6:Sensor_1	wlocations:partD	"1.15"^^units:Meter	"77"^^units:DegreesFahrenheit
resourceAgent6:Sensor_1	dlocations:Europe	"105"^^units:Centimeter	"77"^^units:DegreesFahrenheit
resourceAgent6:Sensor_1	wlocations:Europe	"105"^^units:Centimeter	"77"^^units:DegreesFahrenheit
resourceAgent6:Sensor_1	dlocations:Europe	"108"^^units:Centimeter	"77"^^units:DegreesFahrenheit
resourceAgent6:Sensor_1	wlocations:Europe	"108"^^units:Centimeter	"77"^^units:DegreesFahrenheit
resourceAgent4:Sensor_1	dlocations:Europe	"1.1"^^units:Meter	"70"^^units:DegreesFahrenheit
resourceAgent4:Sensor_1	dlocations:Germany	"1.1"^^units:Meter	"70"^^units:DegreesFahrenheit
resourceAgent4:Sensor_1	dlocations:partA	"1.1"^^units:Meter	"70"^^units:DegreesFahrenheit
resourceAgent4:Sensor_1	wlocations:Europe	"1.1"^^units:Meter	"70"^^units:DegreesFahrenheit
resourceAgent4:Sensor_1	wlocations:partC	"1.1"^^units:Meter	"70"^^units:DegreesFahrenheit
resourceAgent4:Sensor_1	dlocations:Europe	"1.15"^^units:Meter	"70"^^units:DegreesFahrenheit
resourceAgent4:Sensor_1	dlocations:Germany	"1.15"^^units:Meter	"70"^^units:DegreesFahrenheit
resourceAgent4:Sensor_1	wlocations:Europe	"1.15"^^units:Meter	"70"^^units:DegreesFahrenheit
resourceAgent4:Sensor_1	dlocations:Europe	"105"^^units:Centimeter	"70"^^units:DegreesFahrenheit
resourceAgent4:Sensor_1	wlocations:Europe	"105"^^units:Centimeter	"70"^^units:DegreesFahrenheit
resourceAgent4:Sensor_1	dlocations:Europe	"108"^^units:Centimeter	"70"^^units:DegreesFahrenheit
resourceAgent4:Sensor_1	wlocations:Europe	"108"^^units:Centimeter	"70"^^units:DegreesFahrenheit
resourceAgent3:Sensor_1	locations:Europe	"1.1"^^units:Meter	"27"^^units:DegreesCelsius
resourceAgent3:Sensor_1	wlocations:Europe	"1.1"^^units:Meter	"27"^^units:DegreesCelsius
resourceAgent3:Sensor_1	locations:Europe	"1.15"^^units:Meter	"27"^^units:DegreesCelsius
resourceAgent3:Sensor_1	wlocations:Europe	"1.15"^^units:Meter	"27"^^units:DegreesCelsius
resourceAgent3:Sensor_1	locations:Europe	"105"^^units:Centimeter	"27"^^units:DegreesCelsius
resourceAgent3:Sensor_1	locations:Netherlands	"105"^^units:Centimeter	"27"^^units:DegreesCelsius
resourceAgent3:Sensor_1	locations:zoneB	"105"^^units:Centimeter	"27"^^units:DegreesCelsius
resourceAgent3:Sensor_1	wlocations:Europe	"105"^^units:Centimeter	"27"^^units:DegreesCelsius
resourceAgent3:Sensor_1	wlocations:partB	"105"^^units:Centimeter	"27"^^units:DegreesCelsius
resourceAgent3:Sensor_1	locations:Europe	"108"^^units:Centimeter	"27"^^units:DegreesCelsius
resourceAgent3:Sensor_1	locations:Netherlands	"108"^^units:Centimeter	"27"^^units:DegreesCelsius
resourceAgent3:Sensor_1	wlocations:Europe	"108"^^units:Centimeter	"27"^^units:DegreesCelsius
resourceAgent1:Sensor_1	locations:Europe	"1.1"^^units:Meter	"25"^^units:DegreesCelsius
resourceAgent1:Sensor_1	wlocations:Europe	"1.1"^^units:Meter	"25"^^units:DegreesCelsius
resourceAgent1:Sensor_1	locations:Europe	"1.15"^^units:Meter	"25"^^units:DegreesCelsius
resourceAgent1:Sensor_1	wlocations:Europe	"1.15"^^units:Meter	"25"^^units:DegreesCelsius
resourceAgent1:Sensor_1	locations:Europe	"105"^^units:Centimeter	"25"^^units:DegreesCelsius
resourceAgent1:Sensor_1	locations:Netherlands	"105"^^units:Centimeter	"25"^^units:DegreesCelsius
resourceAgent1:Sensor_1	wlocations:Europe	"105"^^units:Centimeter	"25"^^units:DegreesCelsius
resourceAgent1:Sensor_1	locations:Europe	"108"^^units:Centimeter	"25"^^units:DegreesCelsius
resourceAgent1:Sensor_1	locations:Netherlands	"108"^^units:Centimeter	"25"^^units:DegreesCelsius
resourceAgent1:Sensor_1	locations:zoneA	"108"^^units:Centimeter	"25"^^units:DegreesCelsius
resourceAgent1:Sensor_1	wlocations:Europe	"108"^^units:Centimeter	"25"^^units:DegreesCelsius
resourceAgent1:Sensor_1	wlocations:partA	"108"^^units:Centimeter	"25"^^units:DegreesCelsius

Figure 26

What are the temperature measurements and their location?

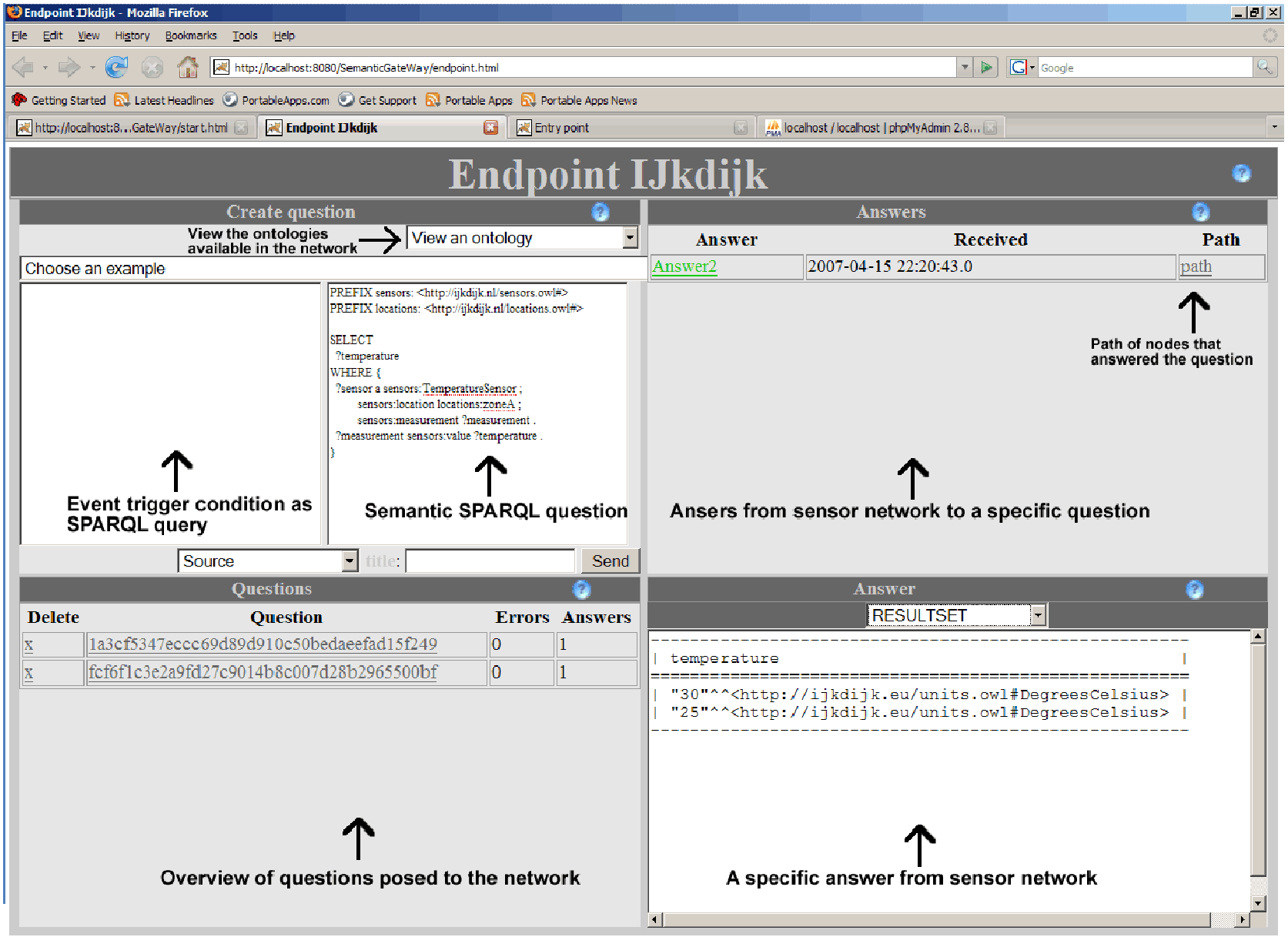
sensor	temperature	loc
resourceAgent6:Sensor_1	"77"^^units:DegreesFahrenheit	dlocations:Europe
resourceAgent6:Sensor_1	"77"^^units:DegreesFahrenheit	dlocations:Germany
resourceAgent6:Sensor_1	"77"^^units:DegreesFahrenheit	dlocations:partB
resourceAgent6:Sensor_1	"77"^^units:DegreesFahrenheit	wlocations:Europe
resourceAgent6:Sensor_1	"77"^^units:DegreesFahrenheit	wlocations:partD
resourceAgent4:Sensor_1	"70"^^units:DegreesFahrenheit	dlocations:Europe
resourceAgent4:Sensor_1	"70"^^units:DegreesFahrenheit	dlocations:Germany
resourceAgent4:Sensor_1	"70"^^units:DegreesFahrenheit	dlocations:partA
resourceAgent4:Sensor_1	"70"^^units:DegreesFahrenheit	wlocations:Europe
resourceAgent4:Sensor_1	"70"^^units:DegreesFahrenheit	wlocations:partC
resourceAgent3:Sensor_1	"27"^^units:DegreesCelsius	locations:Europe
resourceAgent3:Sensor_1	"27"^^units:DegreesCelsius	locations:Netherlands
resourceAgent3:Sensor_1	"27"^^units:DegreesCelsius	locations:zoneB
resourceAgent3:Sensor_1	"27"^^units:DegreesCelsius	wlocations:Europe
resourceAgent3:Sensor_1	"27"^^units:DegreesCelsius	wlocations:partB
resourceAgent1:Sensor_1	"25"^^units:DegreesCelsius	locations:Europe
resourceAgent1:Sensor_1	"25"^^units:DegreesCelsius	locations:Netherlands
resourceAgent1:Sensor_1	"25"^^units:DegreesCelsius	locations:zoneA
resourceAgent1:Sensor_1	"25"^^units:DegreesCelsius	wlocations:Europe
resourceAgent1:Sensor_1	"25"^^units:DegreesCelsius	wlocations:partA

Figure 27

What is the water level at locations where the temperature is over 14, ordered by descending temperature, after converting all the unities to the same format?

temperaturesensor	location	temperature	level
resourceAgent3:Sensor_1	locations:Europe	"27"^^units:DegreesCelsius	"105"^^units:Centimeter
resourceAgent3:Sensor_1	locations:Netherlands	"27"^^units:DegreesCelsius	"105"^^units:Centimeter
resourceAgent3:Sensor_1	locations:zoneB	"27"^^units:DegreesCelsius	"105"^^units:Centimeter
resourceAgent3:Sensor_1	wlocations:Europe	"27"^^units:DegreesCelsius	"105"^^units:Centimeter
resourceAgent3:Sensor_1	wlocations:partB	"27"^^units:DegreesCelsius	"105"^^units:Centimeter
resourceAgent3:Sensor_1	locations:Europe	"27"^^units:DegreesCelsius	"108"^^units:Centimeter
resourceAgent3:Sensor_1	locations:Netherlands	"27"^^units:DegreesCelsius	"108"^^units:Centimeter
resourceAgent3:Sensor_1	wlocations:Europe	"27"^^units:DegreesCelsius	"108"^^units:Centimeter
resourceAgent3:Sensor_1	locations:Europe	"27"^^units:DegreesCelsius	"110"^^units:Centimeter
resourceAgent3:Sensor_1	wlocations:Europe	"27"^^units:DegreesCelsius	"110"^^units:Centimeter
resourceAgent3:Sensor_1	locations:Europe	"27"^^units:DegreesCelsius	"115"^^units:Centimeter
resourceAgent3:Sensor_1	wlocations:Europe	"27"^^units:DegreesCelsius	"115"^^units:Centimeter
resourceAgent6:Sensor_1	dlocations:Europe	"25"^^units:DegreesCelsius	"105"^^units:Centimeter
resourceAgent6:Sensor_1	wlocations:Europe	"25"^^units:DegreesCelsius	"105"^^units:Centimeter
resourceAgent6:Sensor_1	dlocations:Europe	"25"^^units:DegreesCelsius	"108"^^units:Centimeter
resourceAgent6:Sensor_1	wlocations:Europe	"25"^^units:DegreesCelsius	"108"^^units:Centimeter
resourceAgent6:Sensor_1	dlocations:Europe	"25"^^units:DegreesCelsius	"110"^^units:Centimeter
resourceAgent6:Sensor_1	dlocations:Germany	"25"^^units:DegreesCelsius	"110"^^units:Centimeter
resourceAgent6:Sensor_1	wlocations:Europe	"25"^^units:DegreesCelsius	"110"^^units:Centimeter
resourceAgent6:Sensor_1	dlocations:Europe	"25"^^units:DegreesCelsius	"115"^^units:Centimeter
resourceAgent6:Sensor_1	dlocations:Germany	"25"^^units:DegreesCelsius	"115"^^units:Centimeter
resourceAgent6:Sensor_1	dlocations:partB	"25"^^units:DegreesCelsius	"115"^^units:Centimeter
resourceAgent6:Sensor_1	wlocations:Europe	"25"^^units:DegreesCelsius	"115"^^units:Centimeter
resourceAgent6:Sensor_1	wlocations:partD	"25"^^units:DegreesCelsius	"115"^^units:Centimeter
resourceAgent1:Sensor_1	locations:Europe	"25"^^units:DegreesCelsius	"105"^^units:Centimeter
resourceAgent1:Sensor_1	locations:Netherlands	"25"^^units:DegreesCelsius	"105"^^units:Centimeter
resourceAgent1:Sensor_1	wlocations:Europe	"25"^^units:DegreesCelsius	"105"^^units:Centimeter
resourceAgent1:Sensor_1	locations:Europe	"25"^^units:DegreesCelsius	"108"^^units:Centimeter
resourceAgent1:Sensor_1	locations:Netherlands	"25"^^units:DegreesCelsius	"108"^^units:Centimeter
resourceAgent1:Sensor_1	locations:zoneA	"25"^^units:DegreesCelsius	"108"^^units:Centimeter
resourceAgent1:Sensor_1	wlocations:Europe	"25"^^units:DegreesCelsius	"108"^^units:Centimeter
resourceAgent1:Sensor_1	wlocations:partA	"25"^^units:DegreesCelsius	"108"^^units:Centimeter
resourceAgent1:Sensor_1	locations:Europe	"25"^^units:DegreesCelsius	"110"^^units:Centimeter
resourceAgent1:Sensor_1	wlocations:Europe	"25"^^units:DegreesCelsius	"110"^^units:Centimeter
resourceAgent1:Sensor_1	locations:Europe	"25"^^units:DegreesCelsius	"115"^^units:Centimeter
resourceAgent1:Sensor_1	wlocations:Europe	"25"^^units:DegreesCelsius	"115"^^units:Centimeter
resourceAgent4:Sensor_1	dlocations:Europe	"21.111"^^units:DegreesCelsius	"105"^^units:Centimeter
resourceAgent4:Sensor_1	wlocations:Europe	"21.111"^^units:DegreesCelsius	"105"^^units:Centimeter
resourceAgent4:Sensor_1	dlocations:Europe	"21.111"^^units:DegreesCelsius	"108"^^units:Centimeter
resourceAgent4:Sensor_1	wlocations:Europe	"21.111"^^units:DegreesCelsius	"108"^^units:Centimeter
resourceAgent4:Sensor_1	dlocations:Europe	"21.111"^^units:DegreesCelsius	"110"^^units:Centimeter
resourceAgent4:Sensor_1	dlocations:Germany	"21.111"^^units:DegreesCelsius	"110"^^units:Centimeter
resourceAgent4:Sensor_1	dlocations:partA	"21.111"^^units:DegreesCelsius	"110"^^units:Centimeter
resourceAgent4:Sensor_1	wlocations:Europe	"21.111"^^units:DegreesCelsius	"110"^^units:Centimeter
resourceAgent4:Sensor_1	wlocations:partC	"21.111"^^units:DegreesCelsius	"110"^^units:Centimeter
resourceAgent4:Sensor_1	dlocations:Europe	"21.111"^^units:DegreesCelsius	"115"^^units:Centimeter
resourceAgent4:Sensor_1	dlocations:Germany	"21.111"^^units:DegreesCelsius	"115"^^units:Centimeter
resourceAgent4:Sensor_1	wlocations:Europe	"21.111"^^units:DegreesCelsius	"115"^^units:Centimeter

Figure 28


Endpoint IJkdijk - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost:8080/SemanticGateWay/endpoint.html
Getting Started Latest Headlines PortableApps.com Get Support Portable Apps Portable Apps News
http://localhost:8...GateWay/start.html
Endpoint IJkdijk
Entry point
localhost / localhost | phpMyAdmin 2.8...
Endpoint IJkdijk
Create question
View the ontologies available in the network
View an ontology
Choose an example
PREFIX sensors: <http://ijkdijk.nl/sensors.owl#>
PREFIX locations: <http://ijkdijk.nl/locations.owl#>
SELECT
?temperature
WHERE {
?sensor a sensors:TemperatureSensor ;
sensors:location locations:zoneA ;
sensors:measurement ?measurement .
?measurement sensors:value ?temperature .
}
Event trigger condition as SPARQL query
Semantic SPARQL question
Source
title:
Send
Answers
Answer
Received
Path
Answer2
2007-04-15 22:20:43.0
path
Path of nodes that answered the question
Ansers from sensor network to a specific question
Questions
Delete
Question
Errors
Answers
x
1a3cf5347eccc69d89d910c50bedaeefad15f249
0
1
x
fcf6f1c3e2a9fd27c9014b8c007d28b2965500bf
0
1
Overview of questions posed to the network
Answer
RESULTSET
temperature
"30"^^<http://ijkdijk.eu/units.owl#DegreesCelsius>
"25"^^<http://ijkdijk.eu/units.owl#DegreesCelsius>
A specific answer from sensor network

8 Bibliography

[1] T. Berners-Lee, R. Cailliau, A. Luotonen, H. F. Nielsen, and A. Secret, "The World-Wide Web," *Commun. ACM,* vol. 37, pp. 76--82, 1994.

[2] T. Berners-Lee, J. Hendler, and O. Lassila, "The Semantic Web," in *Scientific American*. vol. 284, 2001, pp. 34--43.

[3] S. Castano, A. Ferrara, S. Montanelli, E. Pagani, and G. Rossi, "Ontology-addressable contents in p2p networks," in *1st Workshop on Semantics in Peer-to-Peer and Grid Computing*, 2003.

[4] C. Tempich, M. Ehrig, S. Staab, F. van Harmelen, H. Stuckenschmidt, M. Sabou, R. Siebes, and J. Broekstra, "SWAP: ontology-based knowledge management with peer-to-peer," in *Workshop ontologiebasiertes Wissensmanagement, WM 2003, Lucern*, Bonn, 2003, pp. 17-20.

[5] J. van Diggelen, "Achieving Semantic Interoperability in Multi-agent Systems. A Dialogue-based Approach (PHD thesis)," SIKS, Dutch Research School for Information and Knowledge Systems., 2007.

[6] B. Krishnamachari, D. Estrin, and S. B. Wicker, "The Impact of Data Aggregation in Wireless Sensor Networks," pp. 575--578, 2002.

[7] Y. Yu, R. Govindan, and D. Estrin, "Geographical and Energy Aware Routing: A Recursive Data Dissemination Protocol for Wireless Sensor Networks," UCLA Computer Science Department 2001.

[8] A. Hac, *Wireless Sensor Network Design*: John Wiley & Sons Ltd, 2003.

[9] D. Ganesan, R. Govindan, S. Shenker, and D. Estrin, "Highly-resilient, energy-efficient multipath routing in wireless sensor networks," *SIGMOBILE Mob. Comput. Commun. Rev.,* vol. 5, pp. 11--25, 2001.

[10] N. Jain and D. P. Agrawal, "Current Trends in Wireless Sensor Networks," *International Journal of Distributed Networks,* vol. 1, pp. 101-122, 2005.

[11] C. Tempich, S. Staab, and A. Wranik, "REMINDIN': Semantic Query Routing in Peer-to-Peer Networks Based on Social Metaphors," in *Proc. of the 13th World Wide Web Conference*, New York, USA, 2004, pp. 640--649.

[12] W. Nejdl, M. Wolpers, W. Siberski, C. Schmitz, M. Schlosser, and I. Brunkhorst, "Super-peer-based routing and clustering strategies for RDF-based peer-to-peer networks," in *WWW '03: Proceedings of the 12th international conference on World Wide Web*, New York, NY, USA, 2003, pp. 536--543.

[13] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong, "TAG: a Tiny AGgregation service for ad-hoc sensor networks," *SIGOPS Oper. Syst. Rev.,* vol. 36, pp. 131--146, 2002.

[14] J. J. Nolan, A. K. Sood, and R. Simon, "An agent-based architecture for distributed imagery and geospatial computing," in *Applied Imagery Pattern Recognition Workshop, 2000. Proceedings. 29th*, 2000, pp. 252--258.

[15] I. S. Deshendran Moodley, "A New Architecture for the Sensor Web: The SWAP Framework," Report November 2006.

[16] Mike Botts, George Percivall, Carl Reed, and J. Davidson, "OpenGis Sensor Web Enablement Architecture Document," Open Geospatial Consortium, OGC 06-021, Report 2006.

[17] Dimitre A. Dimitrov, Jeff Heflin , Abir Qasem , and N. Wang, "Information Integration via an End-to-End Distributed Semantic Web System," in *The 5th international semantic web conference* Athens, USA, 2006.

[18] P. K. Biswas and S. Phoha, "A middleware-driven architecture for information dissemination in distributed sensor networks," in *Intelligent Sensors, Sensor Networks and Information Processing Conference, 2004.*, 2004, pp. 605--610.

[19] P. B. Gibbons, B. Karp, Y. Ke, S. Nath, and S. Seshan, "IrisNet: an architecture for a worldwide sensor Web," *Pervasive Computing, IEEE,* vol. 2, pp. 22--33, Oct.-Dec. 2003.

[20] S. R. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong, "TinyDB: an acquisitional query processing system for sensor networks," *ACM Trans. Database Syst.,* vol. 30, pp. 122--173, March 2005.

[21] Y. Yao and J. E. Gehrke, "The cougar approach to in-network query processing in sensor networks," *ACM SIGMOD Record,* vol. 31, pp. 9--18, September 2002.
[22] A. R. Hevner, S. T. March, J. Park, and S. Ram, "Design Science in Information Systems Research," *MIS Quarterly,* vol. 28, pp. 75--105, 2004.
[23] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, "A survey on sensor networks," in *Communications Magazine, IEEE*. vol. 40, 2002, pp. 102--114.
[24] M. Lenzerini, "Data integration: a theoretical perspective," in *Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems* Madison, Wisconsin: ACM Press, 2002, pp. 233--246.
[25] D. Calvanese, G. D. Giacomo, M. Lenzerini, D. Nardi, and R. Rosati, "Data Integration in Data Warehousing," *Int. J. Cooperative Inf. Syst.,* vol. 10, pp. 237-271, 2001.
[26] D. L. Hall and J. Llinas, "An introduction to multisensor data fusion," in *Proceedings of the IEEE*, 1997, pp. 6--23.
[27] H. Qi, X. Wang, S. Iyengar, and K. Chakrabarty, "Multisensor Data Fusion in Distributed Sensor Networks Using Mobile Agents," in *International Conference on Information Fusion*, 2001, pp. pp. 11--16.
[28] D. L. Hall, *Mathematical Techniques in Multisensor Data Fusion*. Norwood, MA, USA: Artech House, Inc., 1992.
[29] S. Cluet, C. Delobel, J. Simeon, and K. Smaga, "Your Mediators Need Data Conversion!," in *Proceedings ACM SIGMOD International Conference on Management of Data*, Seattle, Washington, USA, 1998, pp. 177-188.
[30] J. Gehrke and S. Madden, "Query processing in sensor networks," *Pervasive Computing, IEEE,* vol. 3, pp. 46--55, Jan.-March 2004.
[31] M. Ilyas, I. Mahgoub, and L. Kelly, *Handbook of Sensor Networks: Compact Wireless and Wired Sensing Systems*. Boca Raton, FL, USA: CRC Press, Inc., 2004.
[32] F. Heine, "Scalable p2p based RDF querying," in *Proceedings of the 1st international conference on Scalable information systems*, Hong Kong 2006, p. 17.
[33] S. Handschuh and S. Staab, *Annotation for the Semantic Web* vol. Volume 96 Frontiers in Artificial Intelligence and Applications: IOS Press, 2003.
[34] T. B. Passin, *Explorer's Guide to the Semantic Web*. Greenwich, CT, USA: Manning Publications Co., 2004.
[35] T. R. Gruber, "A translation approach to portable ontology specifications," *Knowledge Acquisition,* vol. 5, pp. 199--220, 1993.
[36] O. Lassila and R. Swick, "Resource Description Framework (RDF) model and syntax specification," 1998.
[37] S. Decker, P. Mitra, and S. Melnik, "Framework for the semantic Web: an RDF tutorial," *Internet Computing, IEEE,* vol. 4, pp. 68--73, Nov.-Dec. 2000.
[38] N. Shadbolt, T. Berners-Lee, and W. Hall, "The Semantic Web Revisited," in *IEEE Intelligent Systems*, 2006, pp. 96--101.
[39] R. J. Bayardo, W. Bohrer, R. S. Brice, A. Cichocki, J. Fowler, A. Helal, V. Kashyap, T. Ksiezyk, G. Martin, M. H. Nodine, M. Rashid, M. Rusinkiewicz, R. Shea, C. Unnikrishnan, A. Unruh, and D. Woelk, "InfoSleuth: Semantic Integration of Information in Open and Dynamic Environments," in *SIGMOD Conference*, 1997, pp. 195-206.
[40] D. L. McGuinness and F. van Harmelen, "OWL Web Ontology Language Overview," February 2004.
[41] M. J. van der Veen, "Using context information to reduce cardinality in a schema mapping task (Unpublished)," Groningen: Department of Bioinformatics, Rijksunversiteit Groningen, 2006, p. 13.
[42] R. Dhamankar, Y. Lee, A. Doan, A. Halevy, and P. Domingos, "imap: Discovering complex semantic matches between database schemas," in *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, Paris, France 2004, pp. 383-394.
[43] A. Doan, P. Domingos, and A. Y. Halevy, "Reconciling Schemas of Disparate Data Sources: A Machine-Learning Approach," in *Proceedings of the 2001 ACM SIGMOD international conference on Management of data*, Santa Barbara, California, United States 2001, pp. 509-520

[44] E. Rahm and P. A. Bernstein, "A survey of approaches to automatic schema matching," *The VLDB Journal,* vol. 10, pp. 334--350, 2001.
[45] B. Parsia and E. Sirin, "Pellet: An OWL DL Reasoner," in *3rd International Semantic Web Conference (ISWC2004)*, 2004.
[46] V. Haarslev and R. Moller, "Racer: A Core Inference Engine for the Semantic Web," in *Proceedings of the International Workshop on Evaluation of Ontology-based Tools (EON '03)*, 2003, pp. 27-36.
[47] J. J. Carroll, I. Dickinson, C. Dollin, D. Reynolds, A. Seaborne, and K. Wilkinson, "Jena: implementing the semantic web recommendations," in *WWW Alt. '04: Proceedings of the 13th international World Wide Web conference on Alternate track papers \& posters*, New York, NY, USA, 2004, pp. 74--83.
[48] S. J. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach (2nd Edition)*: Prentice Hall, 2002.
[49] C. Gutierrez, C. A. Hurtado, and A. O. Mendelzon, "Formal aspects of querying RDF databases," in *Proceedings of SWDB'03, The first International Workshop on Semantic Web and Databases*, 2003, pp. 293-307.
[50] E. Prud'hommeaux and A. Seaborne, "SPARQL Query Language for RDF," W3C April 2006.
[51] C. Bizer and R. Cyganiak, "D2R Server, Publishing Relational Databases on the Semantic Web," Internet, 2006.
[52] M. Woolridge and M. J. Wooldridge, *Introduction to Multiagent Systems*. New York, NY, USA: John Wiley & Sons, Inc., 2001.
[53] J. R. Searle, *Speech Acts: An Essay in the Philosophy of Language*. Cambridge: Cambridge University Press, 1969.
[54] "Fipa ACL Message Structure Specification, SC00061G, Standard, 2002/12/03," IEEE Computer Society, 2002.
[55] F. Bellifemine, G. Rimassa, and A. Poggi, "JADE A FIPA-compliant agent framework," 1999.
[56] R. Pottinger and A. Y. Levy, "A Scalable Algorithm for Answering Queries Using Views," in *Proceedings of the 26th International Conference on Very Large Data Bases* 2000, pp. 484--495.
[57] B. Quilitz, "DARQ - Federated Queries with SPARQL ": http://darq.sourceforge.net/, 2006, p. Implementation based on the JENA architecture.
[58] I. Sommerville, *Software Engineering (6th Edition)*: Pearson Addison Wesley, 2001.
[59] Magid Nikraz, G. Caire, and P. A. Bahri, "A Methodology for the Analysis and Design of Multi-Agent Systems using JADE," *International Journal of Computer Systems Science and Engineering,* 2005.