A Semantic IoT System for Indoor Environment Control based on Sensor and Actuator Support Toolbox

Faiza Tila

2016

A Thesis

For the Degree of Master of Science

A Semantic IoT System for Indoor Environment Control based on Sensor and Actuator Support Toolbox

Faiza Tila

Department of Computer Engineering

Graduate School

Jeju National University

June 2016

A Semantic IoT System for Indoor Environment Control based on Sensor and Actuator Support Toolbox

Faiza Tila

(Supervised by professor Do-Hyeun Kim)

A thesis submitted in partial fulfillment of the requirements for the degree of Master of Science in Computer Engineering.

2016. 06.

This thesis has been examined and approved by

…….

Thesis Committee Chair
Sang-Joon Lee, Professor, Jeju National University

…….

Thesis Committee Member
Jeong-Won Jo, Professor, Jeju National University

…….

Thesis Supervisor,
Do-Hyeun Kim, Professor of Computer Engineering, Jeju National University

Department of Computer Engineering

Graduate School

Jeju National University

Dedicated to my father and his love for our education and to my husband

for being a constant source of support and encouragement!

Acknowledgements

Accomplishing this research has been an amazing journey with pit falls and plateaus. I have learned a lot of new things and met a lot of great people. The journey to completing this study includes the help and support of many people. First of all I would like to pay my gratitude to my supervisor DoHyeun Kim, for introducing me to a world of new technologies. Without his help and numerous meetings I could not have achieved what I did. Secondly I would like to thank my husband, my mentor, for always being on my side, and for his unconditional love and support. This would not have been possible without his constant motivation. I would like to thank my parents for their endless prayers, love and support. They have always encouraged me in pursuing my dreams, believed in me and taught me to overcome the difficulties in life. Finally I would like to thank my siblings, my lab members and my friends for being supportive and helpful throughout my time here.

To all of you, a heartfelt thanks for everything you did!!

Table of Contents

Contents

List of figures

Acronyms

API	Application Programming Interface
RDF	Resource Description Framework
Owl	Ontology Web Language
SPARQL	Simple Protocol and RDF Query Language
GIS	Geographic Information System
HTTP	Hyper Text Transfer Protocol
SQL	Structured Query Language
IoT	Internet of Things
SSN	Semantic Sensor Network
ASP	Actuator Service Provider
SSP	Sensor Service Provider
URI	Uniform Resource Identifier
DUL	Dulce Ultra Lite
OGC	Open Geospatial Consortium
SWE	Sensor Web Enablement
SOA	Service Oriented Architecture
WCF	Windows Communication Foundation
SOAP	Simple Object Access Protocol
PMV	Predicted Mean Vote
ET	Effective Temperature
IoT	Internet of Things
WSDL	Web Service Description Language

Abstract

The vision of the Internet of Things (IoT) is to connect devices all over the Internet to share and exchange data in order to provide useful services to people. According to Cisco's predictions there will be more than 50 billion devices connected to the Internet by 2020. These devices will generate huge amount of data to be acquired by many services and application in areas such as smart homes, smart grids, healthcare, and environmental monitoring. These devices are of heterogeneous nature, producing data in various different formats and requiring different protocols to communicate. This makes interoperability one of the most fundamental requirements to support various tasks such as object addressing, tracking and discovery as well as information representation, storage and exchange.

Recently, different semantic technologies such as ontologies, semantic annotations, linked data and semantic web services have gained popularity specifically in the connected devices domain. In this thesis we have used semantic technologies to overcome the issue of interoperability in an indoor IoT system for environment control. The proposed system is an indoor environment monitoring and controlling application that collects data from different service domains/ service modules it is built upon. To provide interoperability between these domains in order to use their services, we are using semantic technologies. Each service module collects data from its respective domain, and stores and handles the domain knowledge using ontologies. Semantic sensor service provider module is used to collect real time sensing data and sensor information, semantic actuator service provider module is used to exchange messages between the actuators and the top modules, and GIS service provider module provides the location information of these devices. The functionality of these modules is to process this data and provide it to the top module which is the application server. The main idea is to provide an integrated application server that handles the knowledge retrieved from the service modules using semantic technologies. Application server integrates device information with its respective

location information and stores it in RDF form in the application server ontology. This enables semantic interoperability between all the service modules in the system based on the integrated knowledge.

We have developed a toolbox for each module that handles the data creation and storage in the ontology. The data stored by each service module in its ontology can be manipulated using the toolbox associated with it. Each ontology model is a formal representation of the knowledge or context within a service domain. This facilitates effective data access, integration, resource discovery, semantic reasoning and knowledge extraction. Semantic description can be applied to various resources in IoT. An ontology for the Internet of Things provides all necessary semantics for the specification of IoT devices as well as the specifications of the IoT solution that is deployed using these devices. These semantics include terminology related to sensors and observations that is already defined by the SSN ontology. In this thesis we reused the SSN ontology for basic definition of sensors and its observations. System evaluation is performed by comparing the system performance with SQL technology and with hybrid approach including both SQL and SPARQL technologies.

1 Introduction

In simple words, IoT refers to objects ("things/resources") and the virtual representations of these objects on the Internet. The aim of IoT is to represent real world entities on the Internet and to interconnect them to provide useful real world services. These services are offered by various heterogeneous resources or objects. Current advancement in information & communications technologies (ICT) and material sciences have enabled these devices with enough communication and computation capabilities for connecting and interacting with their surrounding environment. These objects produce data/services that represent real world information in a virtualized environment and enables end-users to interact with and control the real world phenomenon through that virtualized environment.

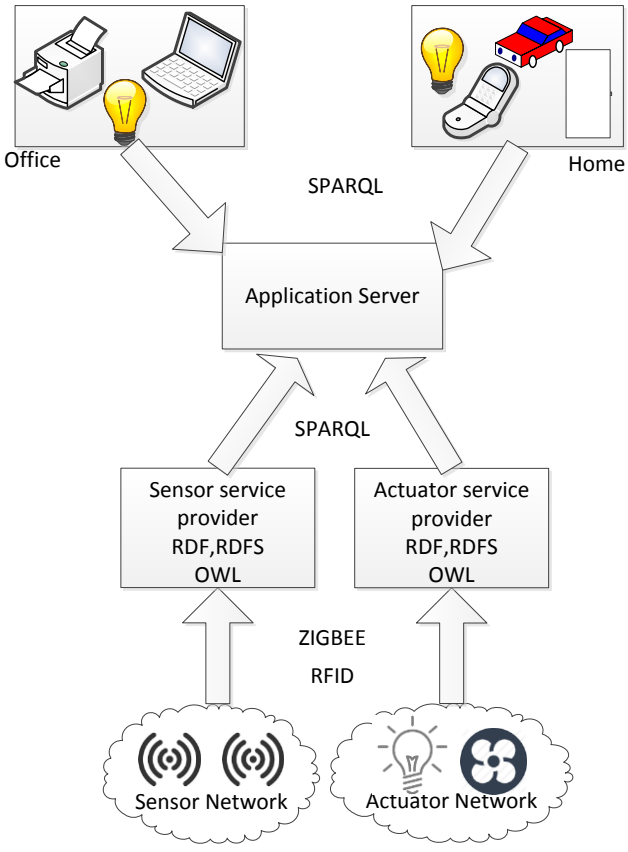

Figure 1 Semantic Interoperability

To enhance these real world data/services they must be able to integrate with data from different sources. Defining these data/services in a uniform way not only allows integration but also supports autonomous reasoning and decision making mechanisms [1]. The main support for realizing IoT comes from the progress in wireless sensor and actuator networks, and from constructing low cost and energy efficient hardware for sensor and device communications. Data collected from different devices and sensors is usually heterogeneous i.e. diverse in nature. This diversity, inconsistency and pervasiveness of the data make it a challenging task to process, integrate and interpret it. This makes interoperability among things on the Internet one of the most important challenges. The word semantic literally means meaning of something. The aim of the semantic web is to provide meaningful data rather than focusing on the structure or representation of data [2]. Its vision is to connect and to attach meaning to data on the Internet for the machines and humans to be able to understand what the data is and where is it coming from. Issues related to interoperability and ambiguity leads to semantic oriented solution towards IoT.

Applying semantic technologies to the things on IoT will make its data unambiguous and transparent for both the users and the applications using it. It also provides efficient data access and integration, resource discovery, reasoning and knowledge extraction. The different semantic web technologies such as ontologies, semantic annotations, semantic web services and linked data can be used for fulfilling the goals of IoT. In this thesis we have developed a semantic IoT system for indoor environment monitoring and control using dotNetRdf API. The proposed system is a hybrid system that utilizes both semantic and database technologies to collect, store and provide environmental context information. The system is an IoT structure that consists of several service modules. To provide interoperability between these domains in order to use their services, we are using semantic technologies. Each service module collects data/knowledge from its respective domain. Each service module stores and handles the domain knowledge using ontologies. The functionality of these modules is to process this data and provide it to the top module which is the application server. Figure 1 shows how the interoperability issue is handled in our system. The

main idea is to provide an integrated application server that manages the information collected from the service modules using semantic technologies.

We designed an ontology model for each of these modules that serve as semantic repositories for storing and representing the semantics and relations of the data collected by each module. The ontology models enables interoperability by annotating the information collected from the devices with semantics. These ontology models are considered as the key solution for the devices to convey their context as useful data. The semantic annotation offers reasoning over the data for knowledge extraction, device discovery and efficient querying on the application server layer despite the underlying technologies of the devices used. The devices in the system are of diverse nature producing data in different formats, but due to the semantics added to the device information in the service provider layer, message exchange between the devices is not an issue.

These semantic repositories in the system are considered as the key solution for the devices to convey their contexts as useful information. Data relations and data descriptions of this contextual data make it interoperable for external users and stakeholders using these ontologies. As we know, ontologies are lightweight and are required to be in memory during program execution in order to be manipulated by applications. In order to preserve the system performance, database techniques might be the best solution [3]. This study presents architecture and models to assess how both the technologies can be used to achieve the best results in a real time system. Databases are widely used. Their main aim is to store vast amount of data. Using the database for real time data storage overcomes the lack of flexibility and performance, when the size of the ontology becomes considerable. The sensing data collected in this system might grow with the addition of new devices which will affect the processing performance of ontology.

Using databases addresses this issue effectively, as they are structured to use a large quantity of data making it easily accessible and with a high performance. With database techniques the real time IoT device data can be accessed and updated safely with high efficiency. The real time sensing data is collected through the service provider and storing it in the database. Using ontology for

storing real time sensing data might decrease system performance especially with the growth of ontology size. We developed a toolbox for each module that handles the data creation and storage in the ontology. The toolbox uses the content service and dotNetRdf API provided by its respective service provide to access and manipulate the RDF data stored in the ontology. This allows the users to use SPARQL queries.

2 Related work

The term semantic web was introduced by Berners-Lee in 1998. It is defined as the web of data. The two main ideas of the semantic web are, 1. The idea to provide homogeneous formats for integrating and compiling data from heterogeneous sources and, 2. The idea to provide meaningful data of how the data relates to each other and to real world objects, thus allowing machines to process and interpret that data[4]. It is defined by the W3C Semantic Web Activity as an entity that is the extension of the World Wide Web in which the meaning or semantics of information on the web is annotated to it so that it is machine understandable [4]. Semantic Web is growing with each day passing. To relate the data or information to make it machine interpretable, the semantics or definitions for information is defined through ontologies.

Semantic web offers an array of languages for modeling and differentiated based on the level of their expressivity. They provide different tools that can be used by users to define or model different kinds of information. The semantic web standards are defined by organizing these languages in layers with each language built on the one before [5]. Figure 2 taken from [6] shows the architecture of the semantic web, it is also known as the semantic web stack. It illustrates the technologies or modelling languages standardized by W3C to enable semantic web [7]. The next layer is the extensible markup language (XML) layer with XML namespace and XML schema definitions. This layer makes sure that there is a common syntax used in the semantic web. XML is a general purpose language for documents containing structured information. Middle layer shows the semantic web technologies. The first one is RDF (Resource Description Framework) that is the building block of data representation format for the semantic web. It enables applications to represent resource information in the form of simple graphs. Its basic aim is to represent metadata about resources on the World Wide Web (WWW).

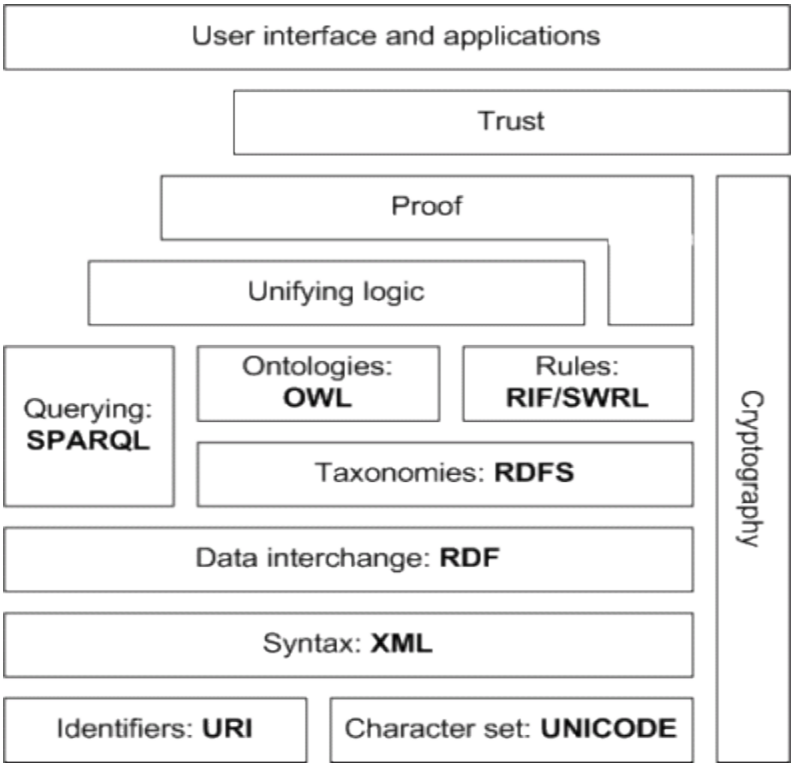

Figure 2 the Semantic Web Stack

All data in the semantic web uses RDF as the primary representation language. RDFS (RDF Schema) provides more expressivity to the RDF by letting applications to create hierarchies of the information. It can be used to define taxonomies of classes and properties and sue them to create light weight ontologies. OWL (Web ontology language) extends the expressivity level of the RDFS by adding more constructs to enhance the RDF statements [8]. It is derived from description logic and offer more constructs over RDFS. OWL comes in three forms i.e. OWL Lite for taxonomy definitions and simple constraints, OWL DL for full description logic support, and OWL Full for maximum expressiveness and syntactic freedom of RDF. OWL and RDFS have defined semantics that enables reasoning within ontologies and knowledge bases. Some relations cannot be defined using these languages. In such cases rule languages are being standardized for the semantic web. Two emerging rule languages are RIF and SWRL. The data defined using the above described semantic web technologies can be queried using an SQL like query language.

Simple protocol and RDF query language (SPARQL) uses RDF triples and resources for matching part and results part of the query. It is expected that all semantics and rules are implemented below the Proof layer and the results will be used to prove deductions. Formal proof together with trusted inputs for the proof will mean that the results can be trusted as shown in the top layer in Figure 2 [1]. For reliable inputs cryptography means are to be used, such as digital signatures for verification of the origin of the resources. On top of these layers, applications with user interfaces can be built. With the application of these Semantic Web technologies, it is possible to automate operations, say, from completing all that you need for a travel to updating of your personal records. Semantic Web then can be defined as a web of information on the Internet and Intranet that contains characteristics of annotation which enables accessing of precise information that you need. Various domains can get benefits from these technologies mainly with issues like heterogeneity, complexity, and volume[9]. These technologies are helpful in managing, querying and combining sensors and observation data. Semantic web technologies could be used in isolation or in augmenting SWE standards in the form of Semantic Sensor Web [10].

The "Internet of Things" (IoT) also referred to as the web of things or the Internet of everything is the emerging idea of connecting more devices to smart networks and to be able to interact with humans and each other through the Internet. According to the research in [16], there were approximately 6.3 billion people on the planet in 2003, and 500 million devices connected to the Internet. This means there was less than one (0.08) device for every person. But IoT didn't exist around that time. According to Cisco IBSG IoT was born between 2008 and 2009. Around 2010 the growth of the smartphones and tablet Pc's was immense which increased the number of connected devices to 12.5 billion. While the world's population increased to 6.8 billion making more than one device per person. The research in [16] predicts the number of connected devices to be 50 billion by 2020. However the number of devices connected per person might get low as most of the world population is not connected to the Internet. These devices will need to connect and communicate to process data and to make it available to applications using it. Depending on the

nature of these devices, several different technologies are used for connecting them to IoT. These technologies include: a) sensing and actuating devices, b) identity devices (e.g. RFID, barcodes), and c) embedded electronics. A primary goal of interconnecting these devices and collecting/processing data from them is to create situation awareness and enable applications, machines, and human users to better understand their environments. The data collected from these devices is diverse in nature, which makes it difficult to process integrate and interpret the data [17].

Internet of things (IoT) enables a global connectivity between the real world and a virtual world of entities or objects. It requires interoperability due to its vision of the millions of devices connected to each other [11].

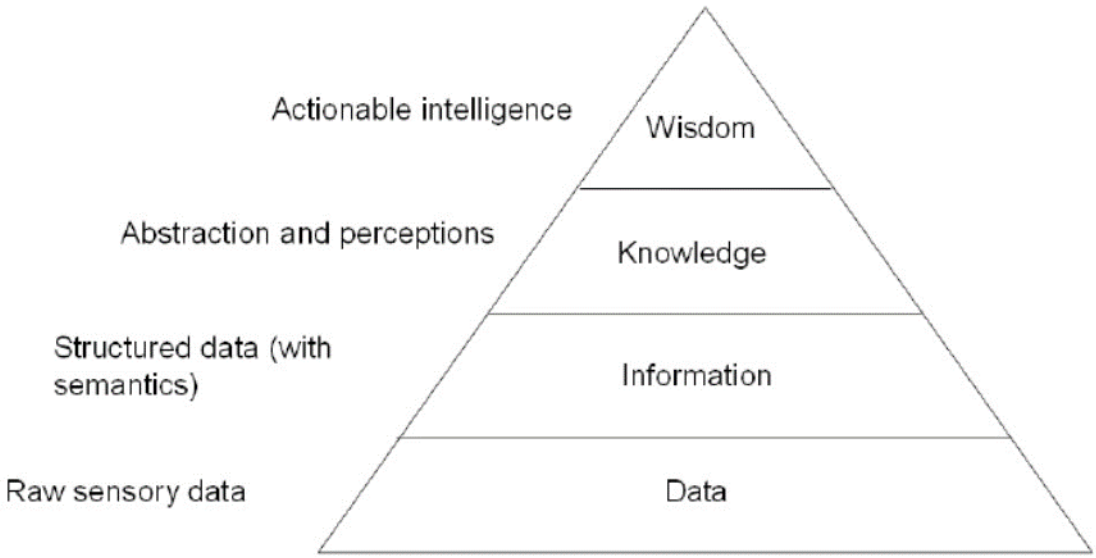

Figure 3 IoT domain knowledge

The applications using these heterogeneous devices require an interface that provides meaningful information about each device in the environment. By getting the right information about the situation of each device in a network, the applications can perform its tasks in an intelligent manner. It makes the communication between the devices and the applications easier. The data collected by IoT devices from a vast domain is processed and analyzed to turn into actionable knowledge to provide a better perception of the physical world to its users. IoT uses an array of steps for gathering this data and turning it to knowledge understandable by users and

machines [12] as shown in Figure 3. As the figure depicts in the first step, raw data is collected from different sources i.e. sensors and other devices available in the surroundings. The data gathered comes from disparate sources having multimodal and diverse nature, which makes processing and integrating it a challenging task. To enhance interoperability the next level's job is to structure the data meaningfully in order to make it machine readable. The next layer takes this structured information and present high level abstractions that provide insights to the underlying data.

The top layer takes this knowledge combines domain and origin knowledge with it and transforms it to wisdom which the IoT applications can use to solve decision making problems. Since many of the data generating IoT devices and resources are highly distributed, heterogeneous, and resource constrained. Issues related to interoperability, automation, and data analytics lead to semantic oriented perspective towards IoT [13]. In an IoT system the IoT devices should be managed and represented through suitable technologies i.e. semantic technologies. Semantic annotation of data can provide machine interpretable descriptions on what the data represents, where it originates from, how it can be related to its surroundings and who is providing it.

2.1 Existing Semantic IoT Systems

In this section we discuss the recent studies that worked on the application of semantic technologies to the field of the Internet of things, which are widely to solve interoperability issues among the IoT resources. It facilitates effective data access and integration, resource discovery, semantic reasoning, and knowledge extraction [16]. One of the main ideas in IoT semantic modelling is to represent the data observation and measurement from sensors. The W3C's incubator group has worked on modelling sensor description and observation on the semantic sensor networks and the OGC sensor web enablement suite [14] of XML based standards. The aim of the OGC standards suite is to use standard protocols and APIs for discovering and accessing

sensor networks and archived sensor data. These standards comprises of modelling schemas i.e. observation and measurement and sensorML [15].

The idea of using ontology driven information system for sensor networks was introduced in [16]. The authors have presented a two phased solution that can be employed to enable a real world wireless sensor network to adapt itself to variations in environmental conditions. The first phase executes an efficient algorithm to dynamically calibrate sensed data, and the second phase executes an efficient ontology driven algorithm to determine the future state of the network under existing conditions. The ontology captures the most important features of a sensor node that describe its functionality and its current state.

Use of sensing devices for collecting data is increasing due to its applications in various areas. This increase is causing an upsurge of data with different data formats from different devices, which requires advanced analytical processing and interpretation by machines. This sensor data is becoming the focus for many researchers these days. The Sensor Web Enablement (SWE) [4] initiative of the Open Geospatial Consortium (OGC) defined data encodings and web services to store and access sensor-related data. The models, encodings, and services of the SWE architecture enables implementation of the interoperable and scalable service oriented networks of heterogeneous sensor systems and client applications. In this regard, SemSOS has proposed ontology models for sensor domain and sensor observations, with semantics annotated to the sensor data and using these models to reason sensor observations. This enables SemSOS to provide the ability to query high level knowledge of the environment as well as low level raw sensor data [17].

The approach in this study is similar to the sensor service provider module of our system, i.e. to leverage semantic technologies in order to provider and apply more meaningful representation of sensor data. 52North's SOS implementation is designed to provide interfaces to sensor observation data stored in a database, with the sensor descriptions stored in XML files as shown in Figure 4. The visualization layer provides an interface to the external clients to interact with SOS,

in our system the application client layer does the same job. An ontology based prototype sensor repository referred to as OntoSensor [18] has also been developed.

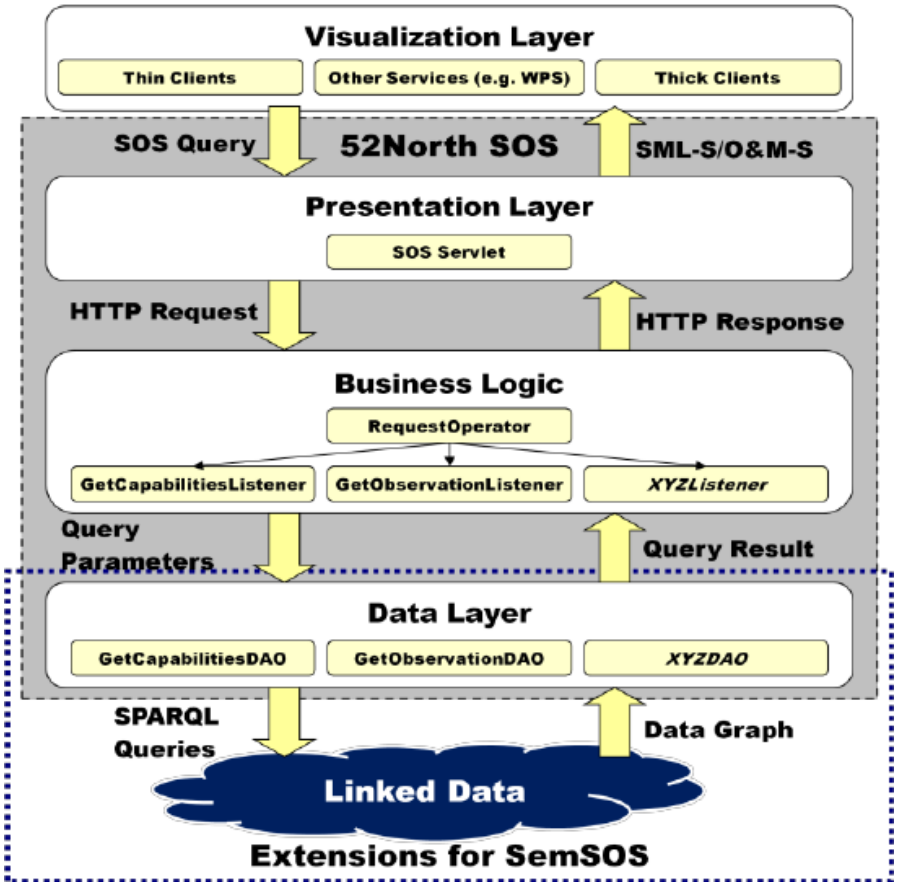

Figure 4 52North SOS Architecture

OntoSensor is a repository containing concepts and relations definitions from SensorML [14]. It extends concepts from the IEEE SUMO ontology, and reference terms from ISO 19115. The authors approach is to use upper level ontologies to deploy a framework in which translation among different domain ontologies can be more readily accomplished. The definitions of high level concepts pertaining to sensors can be used as background knowledge for the integration of data from heterogeneous sensors. Barnaghi, Payam, et al [12] have presented a semantic model for heterogeneous sensor data representation. A sensor data description model is created by using the common standards and logical description frameworks proposed by the semantic web community. The work describes a sensor data ontology which is created based on the Sensor Web Enablement

(SWE) and SensorML data component models. W3C's SSN ontology [9] have been developed to describe the capabilities and properties of sensors, the act of sensing and the resulting observations. Similar studies for representing sensor observations include Sensor Data Ontology (SDO) [19], Sensei O&M [20], and SensorML [14]. Figure 5 describes a M2M architecture that merges data from heterogeneous sensors, add semantics to the measured data making it interoperable with external applications [21]. It consists of gateways that add semantics to sensor measurements enabling application to avail these services regardless the heterogeneity of the devices used.

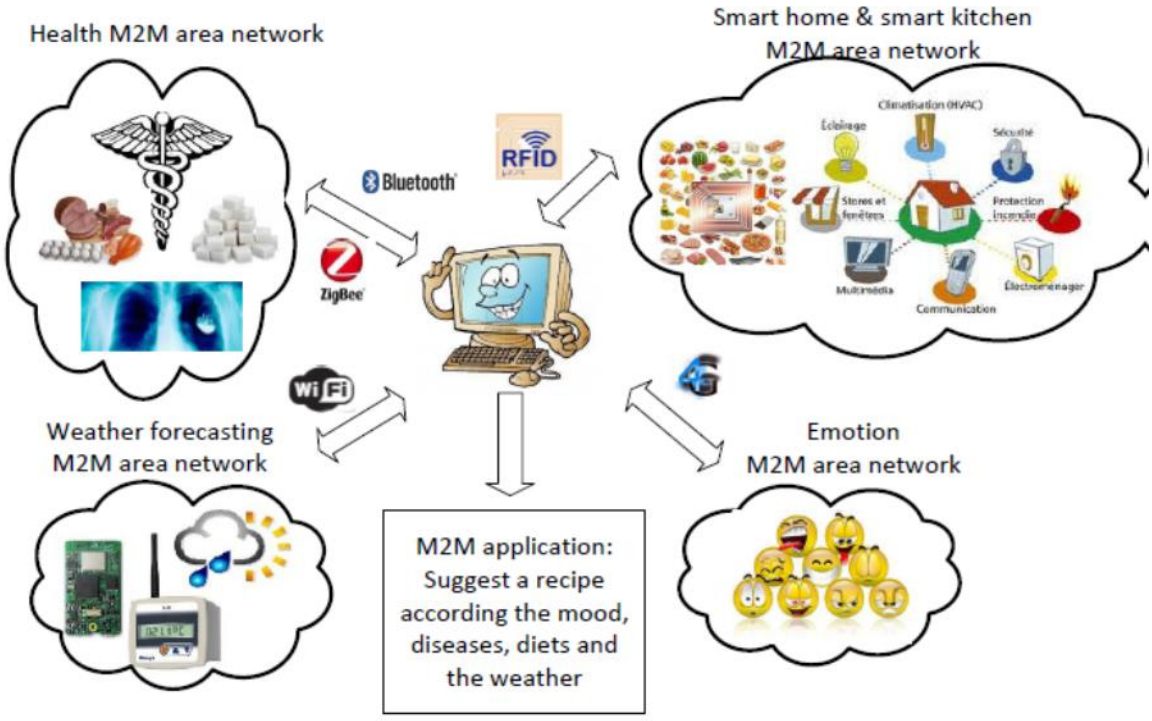

Figure 5 Merging heterogeneous IoT domain

The study in [22] is focused on building an ontology for home energy management domain while reusing the concepts and relationship from Suggested Upper Merged Ontology (SUMO) [23]. The main objective of this study is to provide an ontology that encompasses knowledge of home appliances, their context, causality and relationships. A hierarchical model of the ontology is shown in Figure 6. The ontology developed in this study has the potential to be adopted by vendors and

manufacturers of home appliances that allows automated reasoning over energy efficiency related characteristics of the appliances.

The work in [24] has combined the context of IoT with semantic technologies to build an integrated semantic service platform (ISSP) to support ontological models in various IoT based service domains of a smart city. ISSP is used to develop a prototype service for a smart office, which can provide personalized office environment by interpreting user text input via a smartphone. The service domain knowledge is handled using ontologies which provides interoperability between different service domains based on the integrated knowledge. The authors in [25] have proposed a framework for Linked Open Data (LOD) about sensor data gathered from the physical environment. The work is based on introducing a method to publish LOD. RDF and SPARQL queries are used for semantic discovery between the LOD clouds. This framework enables interoperability between platforms using a sensor dataset description for LOD. The studies that are described in this section focus on using semantic modelling to model sensor data and observations in an IoT environment or some researches model the IoT services exposed by IoT devices whereas other studies focus on modelling sensor data as well as location of the sensors available in a physical environment. In this thesis we proposed to control the comfort index of an indoor environment using a semantic IoT system. The indoor environment we control is the engineering building of Jeju National University. It consists of sensors and actuators to adjust the temperature of the labs in the building.

We have used semantic technologies to provide interoperability between the different sensor and actuator data generated by these IoT devices in the system. We have used semantic modelling to not only capture the sensor data and observations but we have developed ontology models for representing actuators, actuator data, actuator location, and map information of the indoor environment. The semantic indoor IoT system consists of separate modules for representing the data generated by the devices, i.e. for capturing sensor data and information the semantic sensor service provider module is used which uses the sensor ontology to register the sensing devices

available in the environment. For representing the sensor data we have reused the SSN ontology [9].

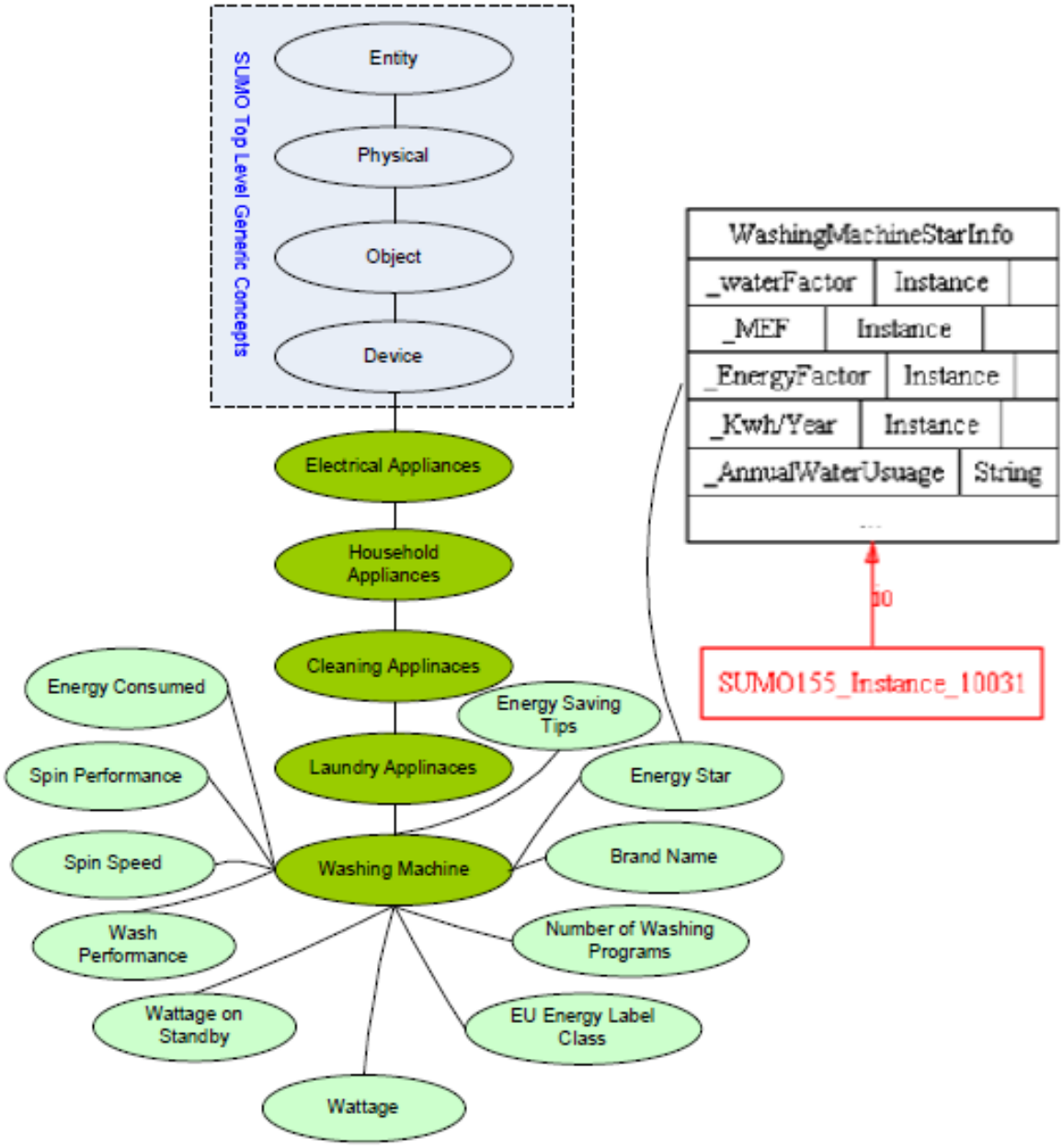

Figure 6 Home Energy Management Ontology [21]

Semantic actuator service provider is used to capture actuator data which uses the actuator ontology to represent actuators and actuator data. Semantic GIS service provider module uses the GIS ontology to represent the location information of the sensors and actuators registered in the

sensor and actuator ontology respectively. Ontology schemas of the system modules enable interoperability by annotating context information with semantics, offering efficient querying and reasoning over the context information. The ontology model presented in this study is considered as the key solution for devices to convey their context as useful data. Data relations and data descriptions of the contextual data make it interoperable for external users and stakeholders using the same ontology. As we know, ontologies are required to be in memory during program execution in order to be manipulated. This might decrease system performance, specially querying, with the growth of the ontology size. In order to preserve the performance, database techniques might be the best solution [12].

This study presents architecture and models to assess how both the technologies can be used to achieve the best results in a real time system. Databases are widely used. Their main aim is to store vast amount of data. Using the database for real time data storage overcomes the lack of flexibility and performance, when the size of the ontology becomes considerable. The sensing data collected in this system might grow with the addition of new devices which will affect the processing performance of ontology. Using databases addresses this issue effectively, as they are structured to use a large quantity of data making it easily accessible and with a high performance. With database techniques the real time IoT device data can be accessed and updated safely with high efficiency.

2.2 The SSN Ontology

With the rapid growth in sensing devices and systems, semantic technologies are used in various studies to manage the enormous amount of data generated as well as the sensors themselves. A huge number of applications are using sensors nowadays ranging from meteorology to medical care to environmental monitoring to security and surveillance. With this the volume of data and the heterogeneity of devices and data formats also grow massively. By using semantics users can manage and query sensors and data. Indeed as the scale and complexity of sensing networks increases, machine interpretable semantics may allow autonomous or semi-autonomous agents to

assist in collecting, processing, reasoning about and acting on sensors and data. For their own part, users generally want to operate at levels above the technical details of format and integration, and rather work with domain concepts and restrictions on quality, allowing technology to handle the details.

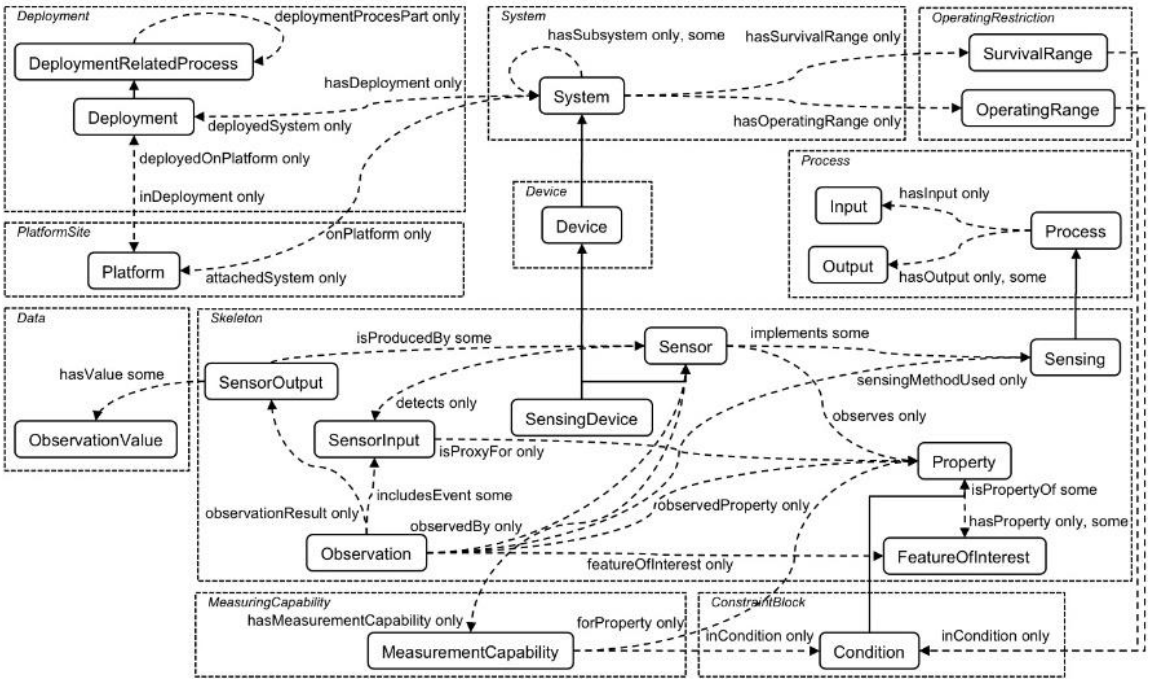

Figure 7 the SSN Ontology Key Concepts source: [9]

The SSN-XG is a W3C incubator group initiated by the CSIRO and Wright State University as a forum for the development of an OWL ontology for sensors and to further investigate annotation of and links to existing standards. The ten conceptual modules and key concepts and relations of the SSN ontology are shown in Figure 7. The full ontology consists of 41 concepts and 39 object properties: that is, 117 concepts and 142 object properties in total, including those from DUL. The development of the design of SSN is the result of two iterations.

Iteration 1: ontology module development and examples,

Iteration 2: alignment to the DOLCE Ultra Lite upper ontology.

Figure 7 shows different concepts and domain connected to each other through object properties. The SSN ontology developed originally has been aligned with the SSO and DUL

ontologies. The SSO (Stimulus Sensor Observation) Ontology Design Pattern represents all kinds of sensors or observation based ontologies and vocabularies for the Semantic Sensor Web and Linked Data. The classes and relations in the SSO are focused on sensor, stimuli, and observations. Each class of SSN is defined as a subclass of an existing DUL class and related to other SSN and DUL classes. The measurement capability and the condition part describes that a sensor may have a number of measurement capabilities (ssn: MeasurementCapability), describing the capability of the sensor in various conditions (ssn:Condition). The system perspective consists of a system (ssn:System) concept representing parts of the sensing infrastructure. A system has components (ssn:hasSubSystem) which are systems. Systems, of which devices and sensing devices are sub concepts, have operating and survival ranges, may be deployed and maybe mounted on platforms. The SSN ontology is reused by a number of research projects for representing sensors and its observations.

In [26] it is used to annotate semantics to streaming sensor data which helps in analyzing the data. It is used as a basic building block in [27] to build large semantic sensor network applications for environmental management. In this study we have reused the SSN ontology for defining sensors and their observations. It is added as a part of the sensor service provider ontology to describe sensor, sensor observations and sensor measurements. Figure 8 illustrates the concepts and properties this study has reused from SSN and DUL ontologies. We have used SSN's sensor definition to represent the sensors that are registered in sensor service provider ontology as shown in Figure 8. The MeasurementCapability class is connected to the sensor by the hasMeasurementCapability object property.

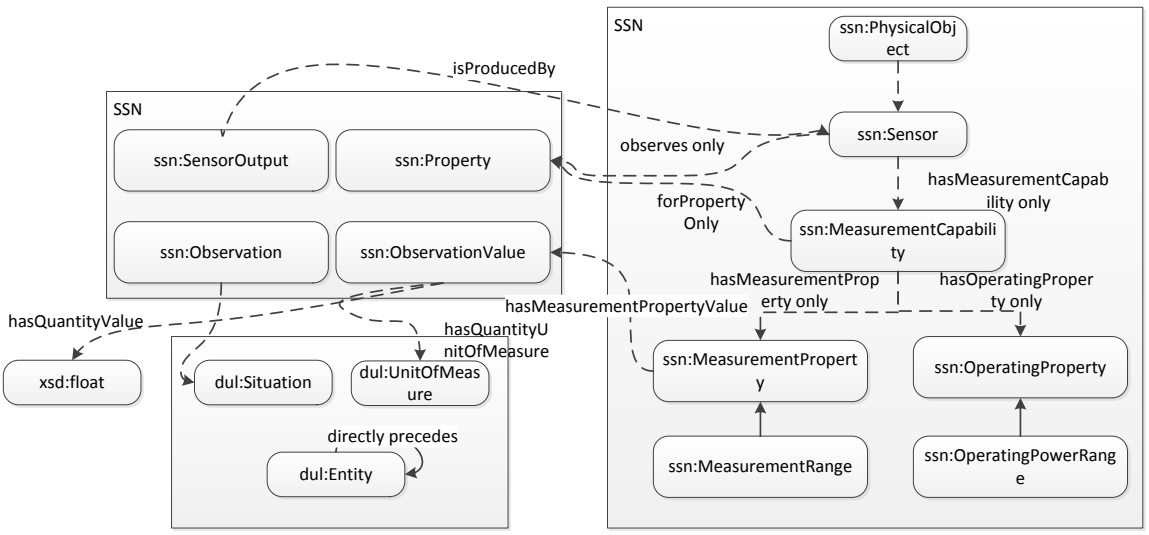

Figure 8 Reused SSN Concepts

The MeasurementCapability class consists of a number of measurement properties, of those we have used the measurement range defined by the MeasurementRange class. MeasurementRange class consists of a set of values that the sensor can return as the result of an observation under the defined conditions with the defined measurement properties. OperatingRange is another property we have used to define the power range in which system/sensor is expected to operate. This is represented using the OperatingPowerRange class. observesOnly defines the relation between a sensor and the property it can observe. SensorOutput class represents an observed value produced by a sensor. The value itself is represented by the ObservationValue class. Each observed value has a unit of measurement and a quantity value as represented in the figure. An Observation is a Situation in which a Sensing method has been used to estimate or calculate a value of a Property of a FeatureOfInterest. Observation is a subclass of DUL:Situation, which represents things that have a ssn:observedProperty property who must be a ssn:Property. The SSN classes and properties represented above describe the characteristics and observations of sensors. There is no information about actuators or the location of sensors and actuators. In this study we developed ontologies to represent sensor and actuator information as well as their location information. It will allow the

users to not only query sensor and actuator data but also locate them. The following sections describe the semantic IoT system architecture and its semantic modelling design phase.

3 Semantic IoT System Architecture based on Sensor and Actuator Network

Semantic IoT indoor environment system is based on a mash up of database and semantic technologies. It is based on a layered architecture as shown in the Figure 9. Each layer consists of a semantic repository that annotates meaningful information with the data that arrives at the layer. This helps in transparency and interoperability within the modules of the system. The bottom most layer in the architecture is the hardware layer. It consists of any kind of sensors and actuators that are available in an indoor area.

The data from these physical devices is collected by the next layer which is the control and acquisition layer. This layer consists of middleware's that helps in monitoring the ports on which the devices are available, collecting sensing data from the sensors, managing the server connection, and exchanging command and control messages between the server and the devices. There are two categories of the middleware's in this layer, the first is sensor middleware which collects and parses the real time sensing data, whereas the second one is the actuator middleware which communicates messages between server and client devices.

The next layer is called the processing layer. Processing layer consists of service providers, which offers services for storing and managing the data received from the middleware layer. These services are used by other modules in the system to manipulate the data stored in the database and ontology. For this purpose dotNetRdf API has been used. It is a powerful and easy to use API for working with RDF, SPARQL, and semantic web. The services offered by the providers use semantic functions, they are called semantic functions because they use SPARQL queries and dotNetRdf API to manipulate the data stored in the ontology model. The three categories of service providers in this layer are semantic sensor service provider, semantic actuator service provider, and semantic GIS service provider. Each service provider offers three services i.e. content service,

provider service and sensing or control service in case of sensor, actuator and GIS provider respectively. It also consists of a database and an ontology model. Semantic sensor service provider stores real time sensor data in the database, whereas the information related to the sensors such as sensor properties, categories and the relationships between them is stored in the ontology.

The reason for using database for storing real time sensor data is the possible decrease in system performance with the growth of ontology size. The functions of the content service are used to manipulate the sensor information in the ontology. Its functionalities include adding a new sensor, deleting a sensor, and updating sensor information, as well as managing the relationships between the sensors and middleware. The provider service is used for provision of sensor data and information to the server whereas the sensing service is used to collect and store sensing data in the database. The relationships between sensors and different modules in the system are modeled in the ontology as well. This will help in queries such as: Which sensors are connected to Middleware1? How many middleware are connected to service provider 2? The semantic actuator service provider also consists of an actuator ontology and a database. It uses the database for real time control of the actuators connected to the system. For example, it can ask for the state of an actuator from the ontology, and change the state by sending a control message through the database. It provides a content service, a provider service, and a control service.

The semantic functions in the content service are used to manipulate information in the ontology. The provider service uses SQL and SPARQL queries to provide the stored data from the database and ontology to the application server. The control service takes care of the control and command messages between the server and the actuators. Semantic GIS service provider stores map images in the database whereas building information in the ontology. Building information includes the number of buildings, the number of floors and rooms in a building. This information can help answer question likes: “How many sensors are there on fourth floor of building 1?” and “Where is sensor 2 located?” Content service of the GIS service provider uses dotNetRdf API to manage the building information in the ontology.

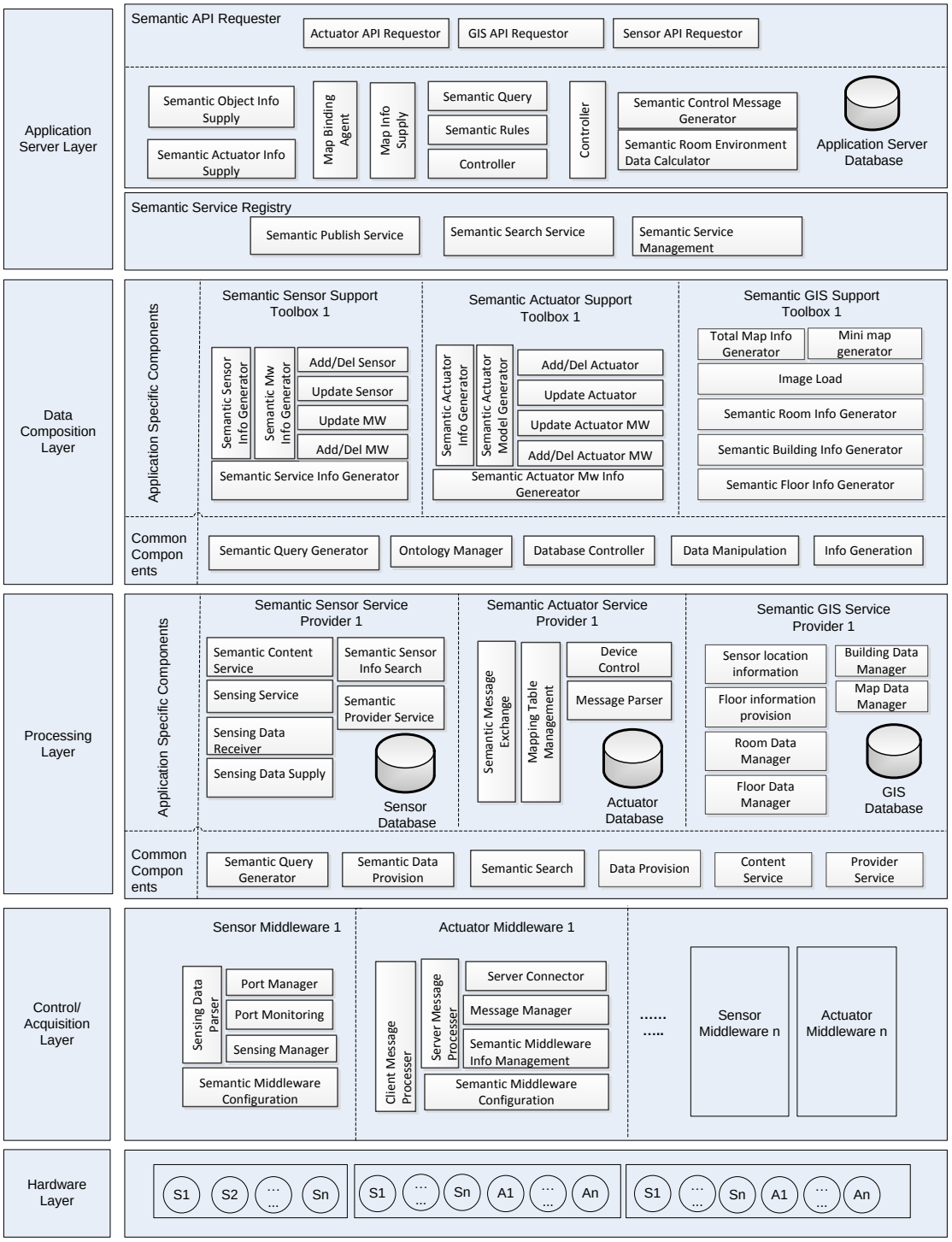

Figure 9 System Architecture

Provider service offers graphic images of building, floor and rooms map from the database whereas the relationship information between these entities and their attributes are provided from the ontology. The next layer in the architecture is the data composition layer. It consists of support

toolbox modules, which uses the service provider's functions to create information related to each module. Sensor support toolbox uses the sensor service provider's content service to perform the following functions

- Get sensors list from ontology,
- Add new sensors,
- Get/save sensing categories,
- Delete / update existing sensors,
- Add sensor middleware, Del/ update sensor middleware, and
- Add relationships between the modules.

Actuator, and GIS toolbox performs similar functions by using content service of actuator service provider and GIS service provider respectively. Semantic service registry is another small module that is used by the application server layer. The ontology model of this module stores the services that are offered by the service provider's available in the system. Semantic Sensor service provider, semantic GIS service provider, and semantic actuator service provider each creates its service information and registers it to the semantic service registry module. The service registry ontology model stores service attributes like service name, type, URI, and service explain. It also offers functions like updating service information and deleting service information. This module offers two main services i.e. to publish and to search a service, publish service is used by the bottom layers whereas the search service is used by the topmost layer which is the semantic application server layer. The semantic application server provides collective viewing and data processing interface. It uses the application server ontology to store the information it retrieves from semantic sensor service provider, semantic actuator service provider, and semantic GIS service provider. It processes the sensed data to find the room comfort index based on fuzzy rules and fuzzy variables. On any request from the client, the semantic application server performs the following series of steps;

- First it searches for the intended semantic sensor provider/semantic actuator provider service information in the semantic service registry module and binds this information to a map based on client's choice.
- Next it creates binding information for the map coordinates and sensor/actuator locations based on client's choice.
- The semantic application server finally saves the binding information to the application server ontology, creates application server provider service information and registers it to the service registry module.

4 Design of Semantic IoT System based on Sensor Network

As mentioned earlier semantic IoT indoor system is a layered architecture. It consists of five modules i.e. semantic sensor module, semantic actuator module, semantic GIS module, semantic service registry module, and application server module. Each of these modules has several components which provide services based on the data it collects. In this section we have explained the functionality and the data flow within each module and their components using configuration and sequence diagrams.

4.1 Semantic Sensor Platform

4.1.1 Semantic Sensor Service Provider

The following Figure 10 shows the detailed architecture of the semantic sensor service provider module. It shows the functionalities offered by each service provided by this module. The Figure 10 also gives an illustration of the sensor service provider ontology and database. The sensor service provider ontology reuses the Semantic Sensor Network (SSN) ontology. The semantic service interface provides access to the modules that uses the services of the semantic sensor service provider. Semantic queries are run and processed using dotNetRdf API. The three services provided by this module are semantic content service, semantic provider service and sensing service. Semantic content service is used for middleware information and sensor information management. Middleware information management involves storing middleware id, IP and service access information. Sensor information management involves creation and management of sensor information (such as sensor id, code, name and explain). Semantic provider service is utilized for searching sensors, sensor information provision, and sensing data provision. Sensor search requires a search keyword to retrieve a list of sensor ids.

Sensor information provision supplies sensor information stored in the ontology based on the sensor id. Sensing data provision supplies the real time sensing data stored in the service provider database to the client. The last service is the sensing service that performs two functions sensor state management, and sensing data receiving. Sensor state management keeps the record of the sensing state of sensors, and sensing data receiver provides the functionality of receiving sensing data collected from sensor middleware and storing it to the database.

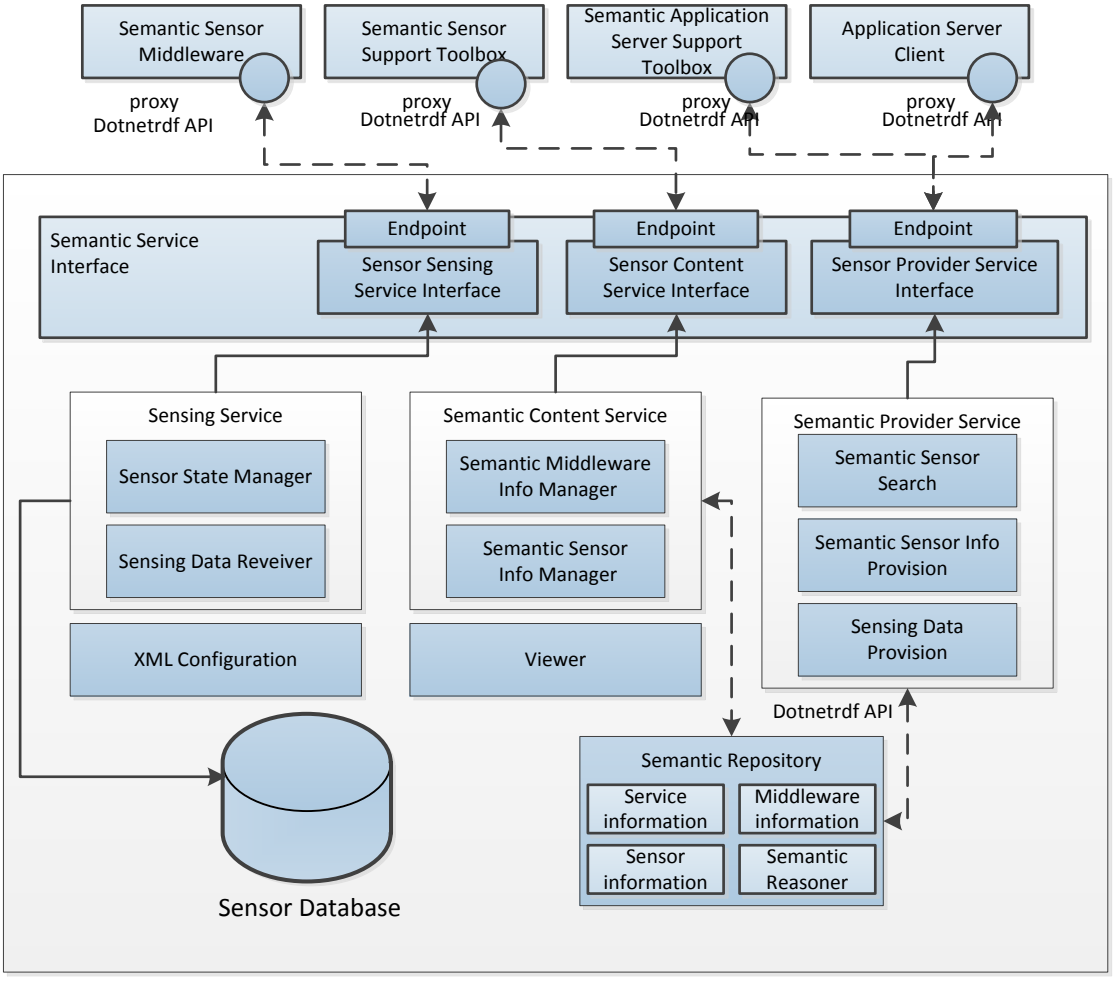

Figure 10 Semantic Sensor Service Provider Configuration Diagram

4.1.2 Semantic Sensor Middleware

Semantic sensor middleware is the intermediary module that collects real time sensing data from sensors connected to it. As shown in Figure 11 it consists of five modules. Semantic

middleware configuration manager verifies the semantic sensor service provider's access privileges by requesting middleware IP address and access rights. Sensing driver takes various sensors' sensing data format information and parses the received sensing data. Port monitoring monitors the state of the port connected to the middleware. Sensing data receiver accesses the sensing data sent from the sensor nodes and saves it in memory through the sensing data parser. Sensing data transporter reads the real time sensing data stored to the Memory and sends it to the semantic sensor service provider to be stored in database.

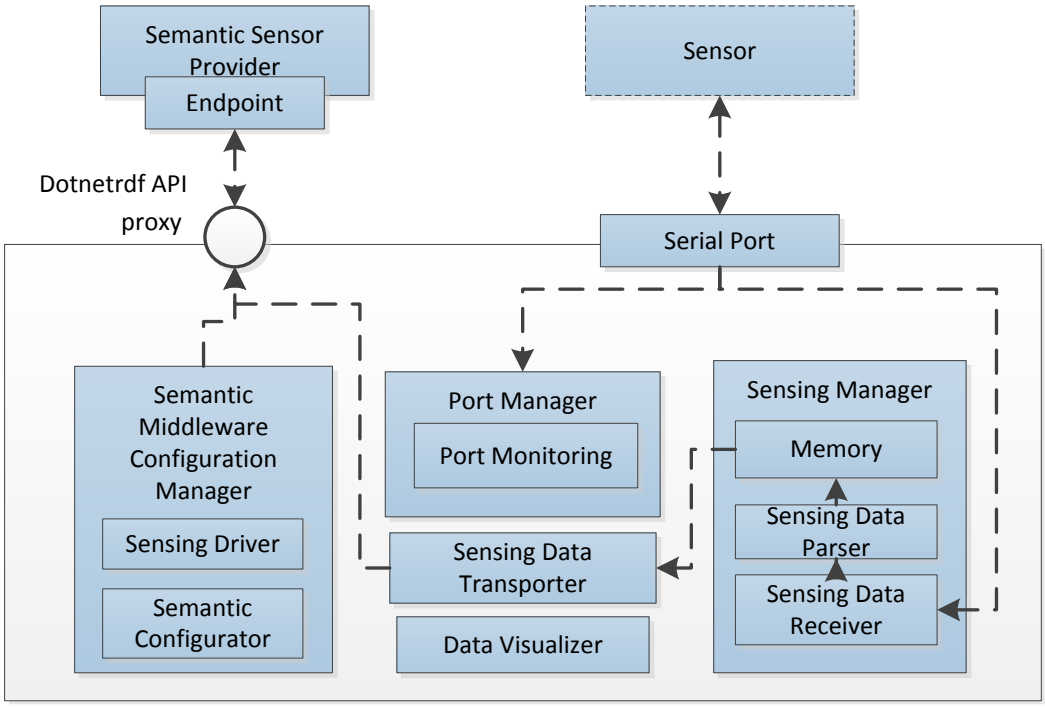

Figure 11 Semantic Sensor Middleware

4.1.3 Semantic Sensor Support Toolbox

Figure 12 shows the detailed architecture of the semantic sensor support toolbox. Its functionality is to create sensor information, provide middleware access management, and service information publishing support. The semantic sensor information generator uses the semantic content service provided by the semantic sensor service provider to create sensor information and stores it to the ontology. The semantic middleware manager creates middleware information and

manages the service provider's access privileges for the middleware. Semantic service information publisher creates service information (service name, service Uri, and a search keyword) through semantic service info generator and stores it to local XML file as well as publish it to the semantic service registry.

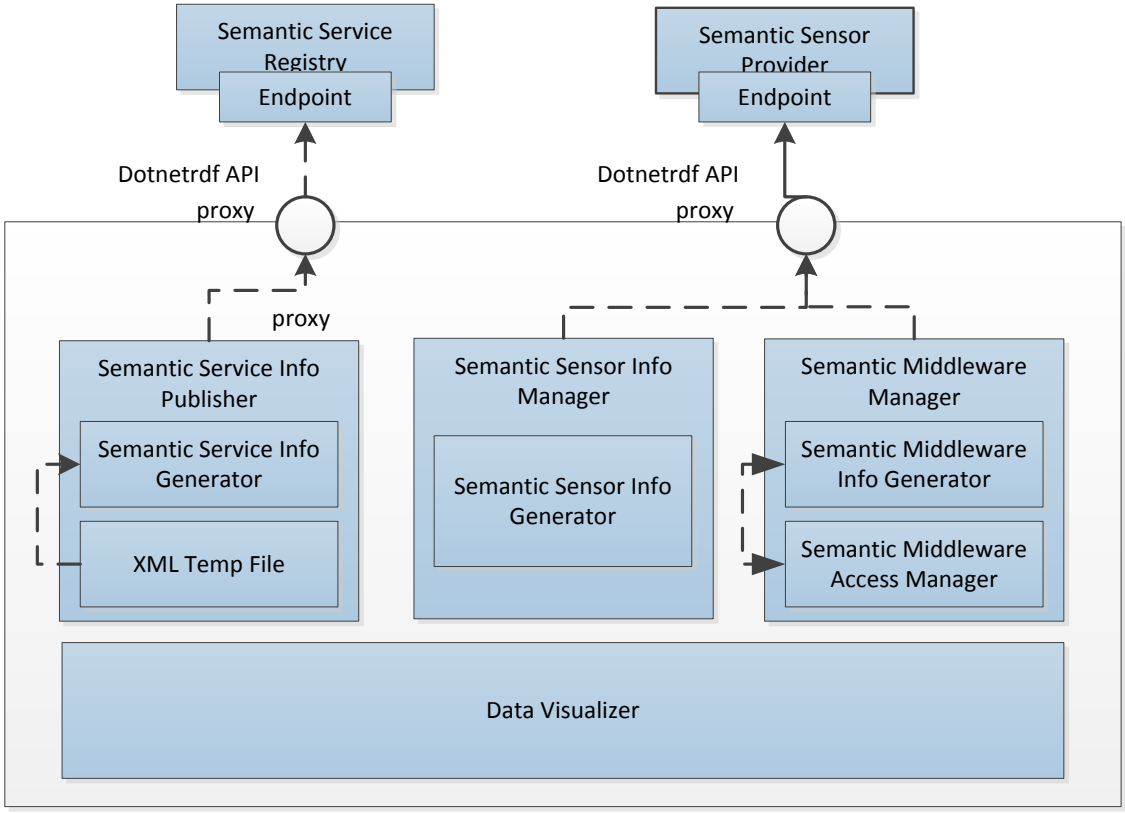

Figure 12 Semantic Sensor Support Toolbox

Figure 13 shows the interaction of the semantic sensor service provider module with semantic sensor support toolbox and the sensor ontology model. Sensor Provider has three services namely: sensor web provider service, content service and sensing service. Sensor support toolbox uses the content service to manage middleware, sensor and provider information. The sensor service provider module interacts with the ontology model using the dotNetRdf API. As the main window of the semantic sensor service provider module is loaded it starts the timer to record the amount of time for which the services are run. Next it starts the content service, the sensing service and the provider service. The functionality provided by this module can now be used as services by the

semantic sensor support toolbox module. The semantic sensor support toolbox asks for the content service proxy to connect and perform sensor management which includes getting the content service proxy for enabling semantic queries on the sensor service provider ontology.

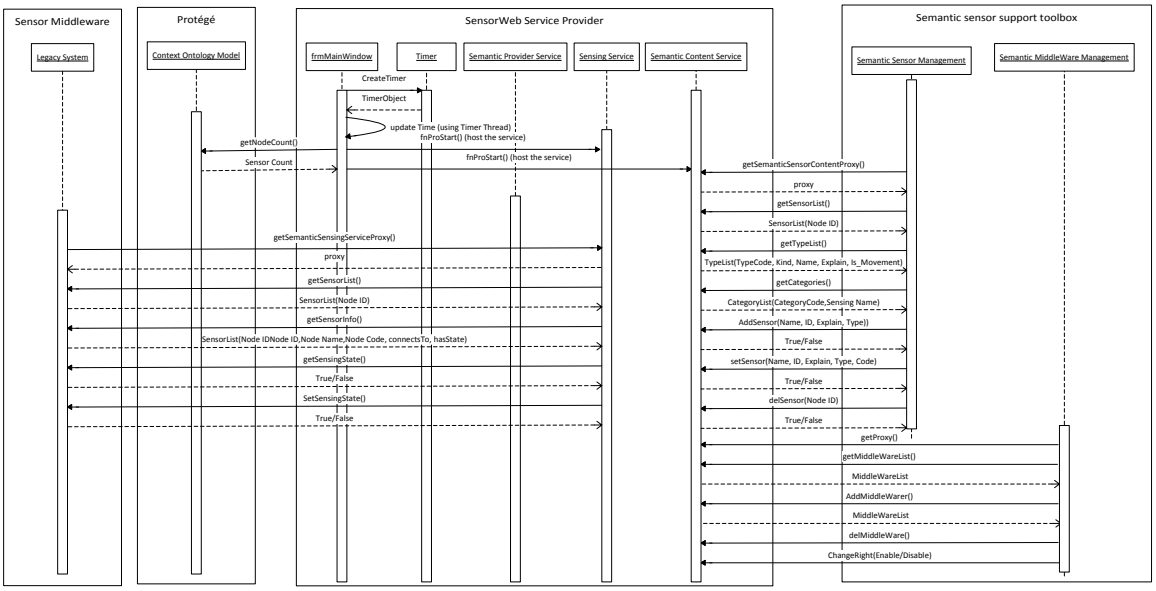

Figure 13 Semantic Sensor Provider Sequence Diagram

Next it performs the following queries 1) get the sensor list which returns a list of the sensor ids registered in the ontology, 2) getting the type list which returns a list of sensor types, 3) get the category list which returns a list of all the sensor categories registered in the ontology. It also performs the following SPARQL Update queries 1) Add sensor, 2) Update sensor, and 3) Delete sensor. The semantic sensor support toolbox also performs sensor middleware management which includes the following SPARQL queries 1) Getting the middleware list, 2) Adding a middleware, 3) Updating a middleware, and 4) Deleting a middleware. The sensor middleware or module asks for the sensing service proxy to connect and perform sensor state management which includes 1) Getting the sensor list, 2) Getting the sensing state, 3) Updating the sensing state, and 4) Saving the sensing state. Finally when the sensor state and the sensed data is stored the services are closed.

4.1.4 Semantic sensor provider ontology modelling

In this section figures are used to describe the ontology models for semantic sensor provider module. As mentioned earlier it uses some of the concepts of the SSN ontology. Figure 14 represents the Onto Graph of the ontology generated by Protégé. Onto Graf gives support for interactively navigating the relationships of your OWL ontologies. Different relationships supported are: subclass, individual, domain/range object properties, and equivalence. The figure shows that all the classes in the ontology are derived from the Thing class. For the purpose of simplicity we have shown the relationships of one of the sensors registered in the ontology. Sensor class is defined in the SSN ontology and reused here. TIP700SM is an instance of the Sensor class. It is connected to ssn:MeasurementCapability class by ssn:hasMeasurementCapability property which shows that this sensor has the capability to measure some property. In this particular class TIP700SM has the capability to measure Humidity, Temperature and Illumination values which is shown by the instances TIP700SMTemperatureMeasurementCapability, TIP700SMHumidityMeasurementCapability, and TIP700IlluminationMeasurementCapability respectively. The state of the TIP700SM is shown as idle which the instance of the State class. It also shows that TIP700SM is connected to Sensor Middleware1 which is an instance of Sensor Middleware Class.

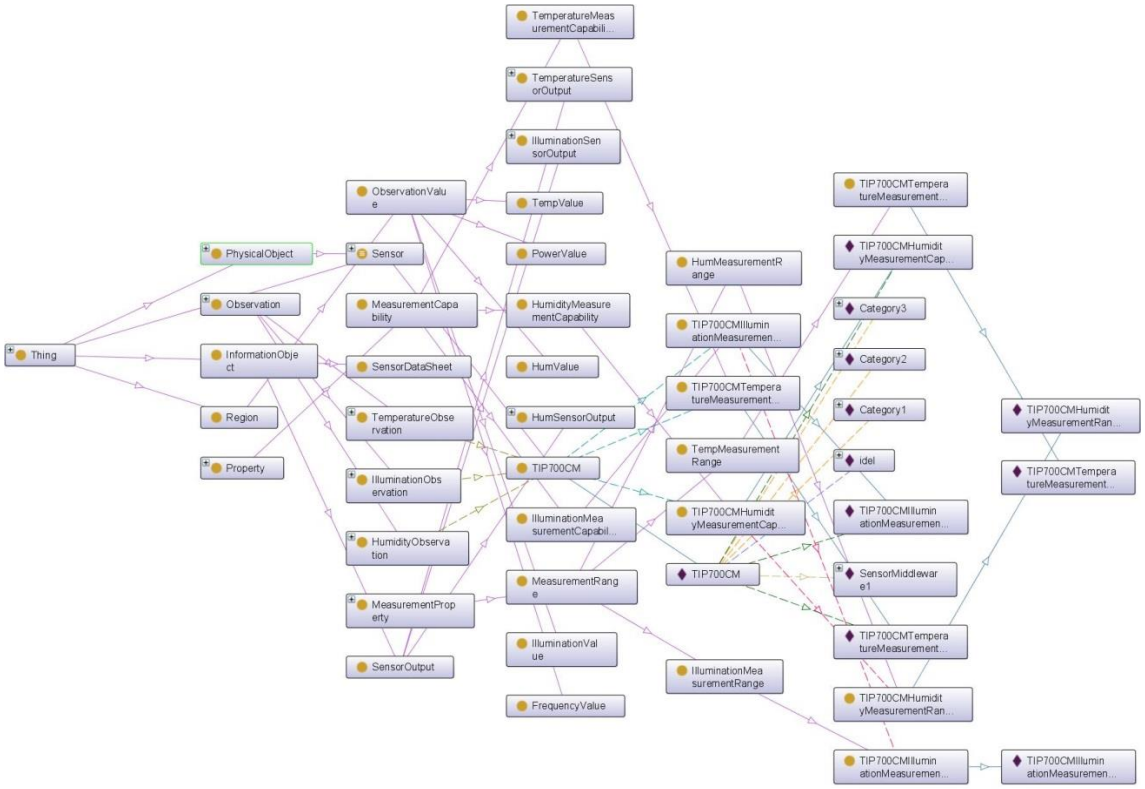

Figure 14 semantic sensor ontology graph

Figure 15 is a description of the reuse of SSN ontology as done in this study. It shows three instances namely TIP700SMHumidityMeasurementRange, TIP700SMTemperatureMeasurementRange, and TIP700SMIlluminationMeasurementRange. These instances are related to the ssn:MeasurementCapability class through the ssn:hasMeasurementPropery relation. The sensor output is represented through the ssn:SensorOutput class which contains HumiditySensorOutput, TemperatureSensorOutput, and IlluminationSensorOutput classes. The values of these outputs are represented by the observation class and the relationship between these classes is ssn:hasValue. ssn:isProducedBy object property is used to relate to the device these outputs are produced by.

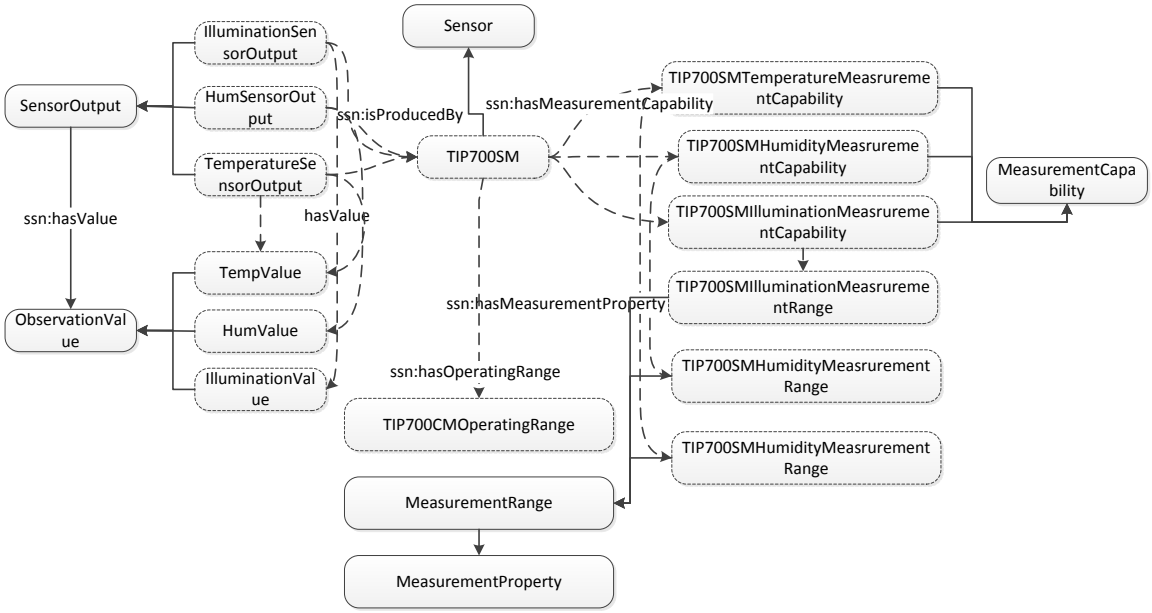

Figure 15 SSN ontology concepts

4.2 Semantic Actuator Platform based on Actuator Network

4.2.1 Semantic Actuator Service Provider

The semantic actuator module consists of semantic actuator service provider, semantic actuator middleware, and semantic actuator support toolbox. This module performs actuator information management, as well as supplying and monitoring actuator control and operation status. The actuators available in the system can be controlled via the semantic application server. This actuator operation control is based on the environmental state. Semantic actuator middleware receives the command message from the semantic actuator provider and sends it to the actuator. Figure 16 shows the actuator service provider configuration. It shows the details of the services provided by the semantic actuator service provider. It also shows the actuator database as well as the actuator ontology maintained by the provider. The semantic actuator content service handles the semantic

configuration of the actuator middleware, and actuator information management. Semantic configuration means creating an ID and IP address for each middleware.

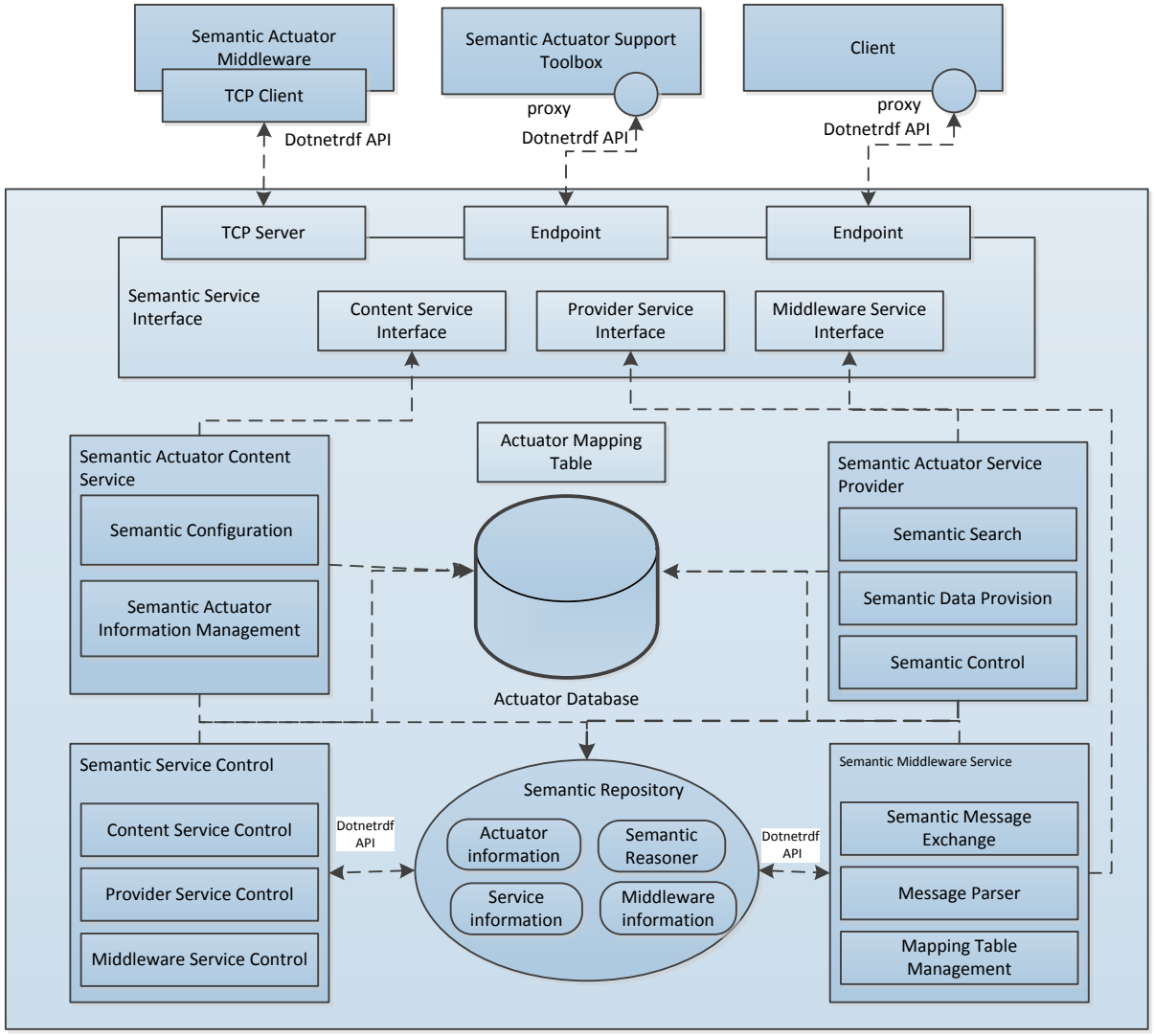

Figure 16 Semantic Actuator Service Provider

A mapping table is maintained for each actuator that stores the actuator id and the IP address associated to it. Actuator information management means adding new actuator information to the ontology and deleting/updating existing information. The next service offered by this module as shown in the figure is semantic actuator provider service. Its functionalities include semantic search, semantic control and data provision. Semantic search retrieves actuator id from actuator information in the ontology based on a search keyword. Data provision functions by providing

actuator information e.g. actuator name, explain, code etc. based on actuator ID. Semantic control service offers actuator remote control interface.

The next service is the semantic middleware service, it manages parsing the middleware mapping table. Mapping table saves actuator ID in ontology and manages middleware address information. Service control controls the start and stop functionality for the services in this module. Semantic repository shown in the figure represents the actuator ontology. It consists of actuator information, actuator middleware information and semantic actuator provider's service information. A reasoner is run on the above information the results in improved resource discovery.

4.2.2 Semantic Actuator Middleware

Semantic actuator middleware as shown in Figure 17 is the intermediary module between the actuators in the physical layer and the semantic actuator service provider in the service layer. Its basic function is to precisely send the messages sent by Actuator Web provider to the specific actuator and send messages sent by actuator to service provider. This module implements two-way communication using TCP sockets. Its basic functionality is to exchange messages between these two layers. It consists of seven functional units i.e. service message processor, client message processor, mapping table manager, message manager, middleware connection manager, semantic viewer, and actuator mapping table. Server message processor controls the messages sent by the semantic actuator service provider to the actuator, where Message manager is responsible for management of the messages flowing between the semantic actuator providers and the actuators. It consists of two components message parser parses these messages and message processor decides the type of the processing from the parsed message. The messages can be a connection request message, control request, mapping table renewal request, control response and connect response. As client message processor receives messages from actuators available in the system. Middleware manager manages the connection and setup of the middleware to the semantic actuator service provider. It consists of semantic middleware configuration and server connector.

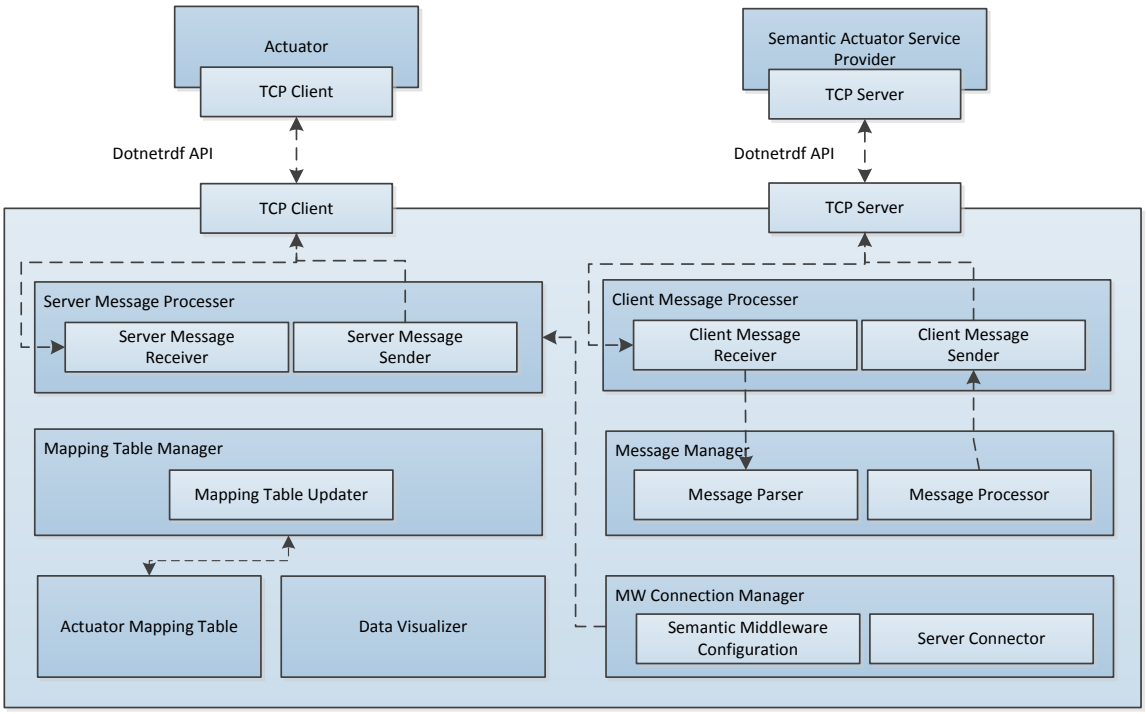

Figure 17 Semantic Actuator Middleware

Semantic middleware configuration functions by running a SPARQL query to provider actuator id and IP information to the service provider. The functionality of the server connector is to verify middleware access right, and to decide whether service provider can connect to the middleware. The mapping table manager updates mapping table with actuator mapping information in real time

4.2.3 Semantic Actuator Support Toolbox

Figure 18 shows the detailed architecture of the semantic actuator support toolbox. The functionality provided by this module includes creating actuator and middleware information, middleware access management, and registration of service information. It consists of three units the semantic actuator information manager, semantic middleware information manager, and semantic service information publishing. The first two units connects to the semantic actuator provider through dotNetRdf API and the third unit connects to the semantic service registry through the dotNetRdf API. Semantic actuator information generator is responsible for creating information

for an actuator discovered in the system, whereas semantic actuator model information generator creates and manages actuator model information. Semantic middleware information management consists of semantic middleware access right and semantic middleware information generator. Semantic middleware access right management manages the middleware access rights for the semantic actuator service provider. Semantic middleware information generator creates and manages middleware information stored in the ontology. Semantic service information publishing creates service information through service information generator, and saves it to local XML file as well as registers it with the semantic service registry module.

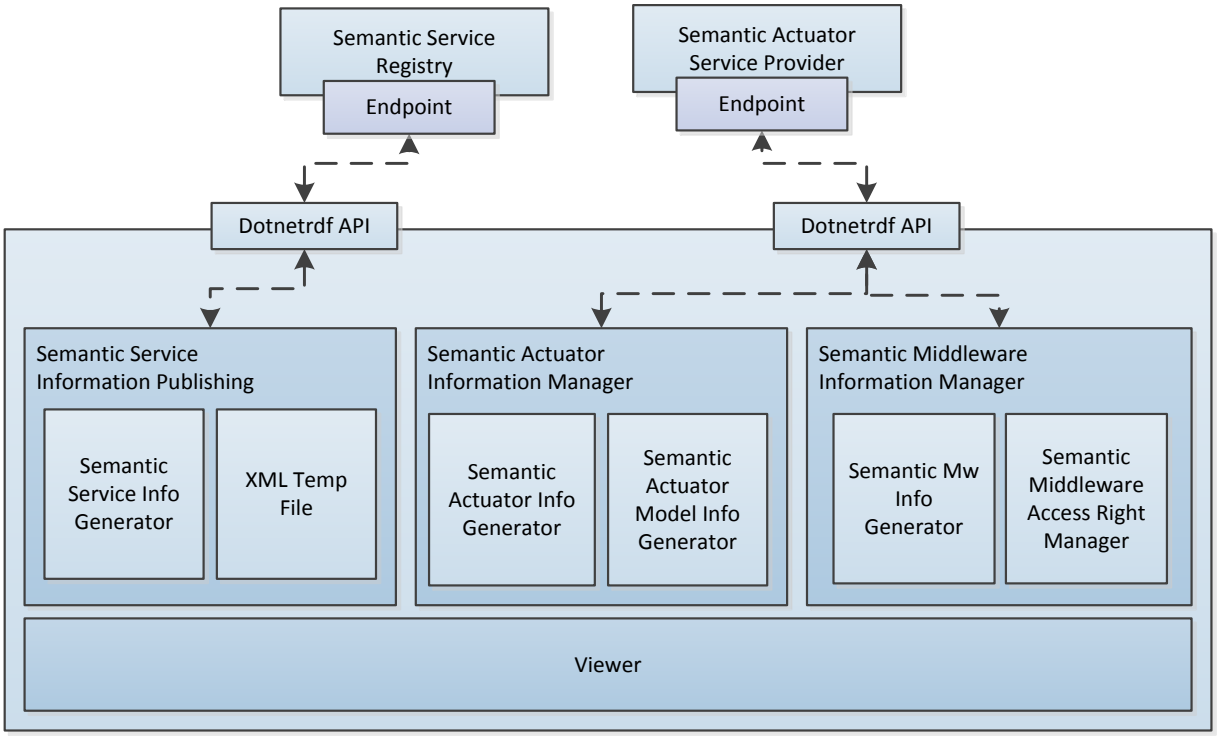

Figure 18 Semantic Actuator Support Toolbox

Figure 19 shows the sequence diagram that illustrates the interaction between the semantic actuator service provider and semantic actuator support toolbox, and the interaction of these modules with the actuator ontology model. These modules use the dotNetRdf API and SPARQL queries to interaction with the actuator ontology. It retrieves data from the ontology as well as stores data to it.

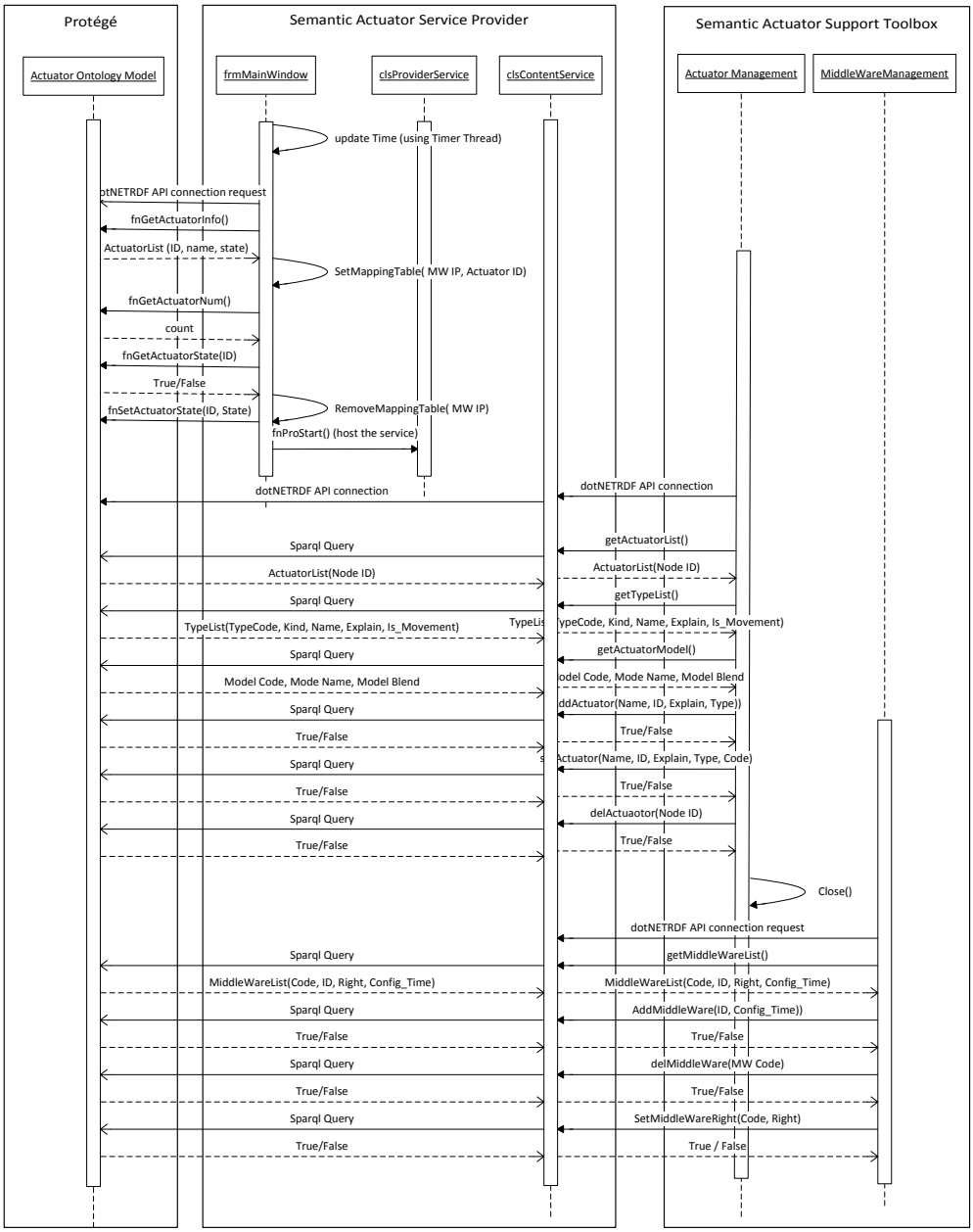

Figure 19 Semantic actuator service provider and support toolbox sequence diagram

The main window after loading connects to the ontology model to retrieve the following information i.e. Actuator info, Actuator no, Actuator state and to set the actuator state. The semantic actuator support toolbox connects to the content service to perform actuator management which includes the following functionalities: 1) Getting actuator list, 2) Getting type list, 3) Getting actuator model, 4) Adding an actuator, 5) Updating an actuator, and 6) Deleting an actuator. The

semantic actuator support toolbox connects to the content service to perform actuator middleware management which includes Getting actuator middleware list, Updating an actuator middleware, and Deleting an actuator middleware.

4.2.4 Actuator Service Provider Ontology Modelling

Actuator service provider ontology is maintained in the semantic actuator service provider module. Figure 20 shows the ontology graph as generated by protégé. This graph shows the classes and their respective instances and the relations between them. The classes defined in the ontology are Device, Actuating Device, Actuator Middleware, Actuator Model, Type Information, Time Duration, Switch State, Range Attribute, Multistep Attribute, Provider Service, Content Service, Middleware Service, Management, Actuator Middleware Management, Actuator Model Management, Actuator Support Toolbox, Service Information Management, and Actuator Management.

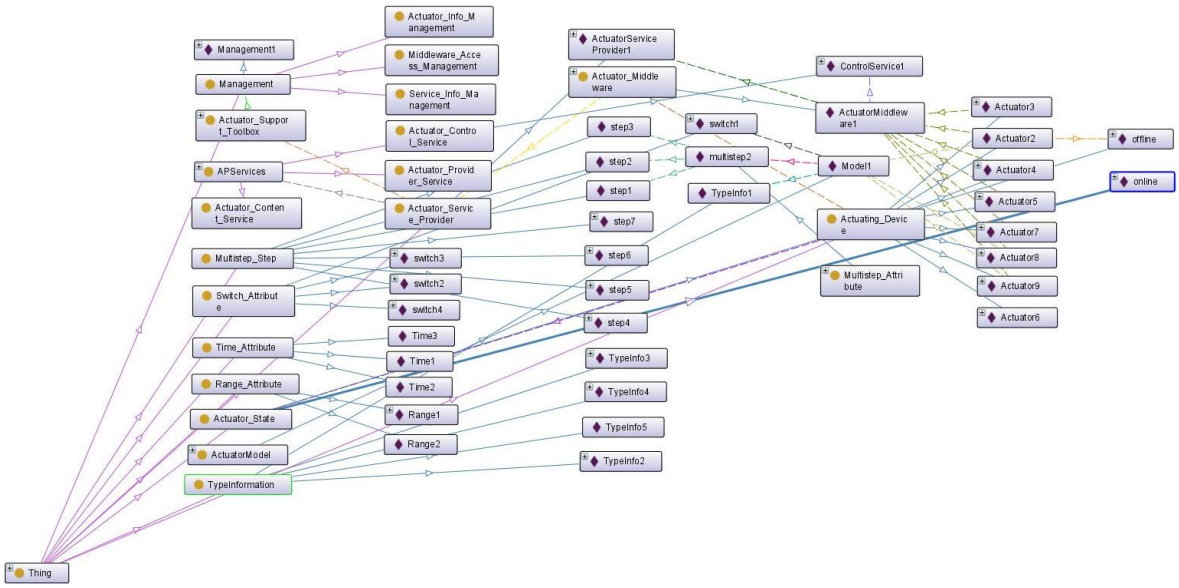

Figure 20 Actuator Ontology Graph

Figure 21 shows the ontology model developed for the actuator service provider ontology showing the object properties in detail. It models the actuator information, actuator middleware

information and the different components of the actuator service provider and the relationships between them. The object properties involved in the ontology are Provides, connectsActuatorToMiddleware, connectsActuatorMiddlewareToProvider, connectsActuatorProviderToSupportToolbox, subClassOf, hasDuration, hasSwitchState, hasType, hasRange, hasMultiStep, hasModel, and performs. Device class represents any hardware device that the modeler wants to talk about. Actuating Device is the subclass of the Device class. Each actuator added to the ontology is a type of Actuating Device and Device class. Each Actuating Device has a model which is represented by the Actuator Model, and is connected to it through the hasModel object property.

Each Actuator has a number of attributes based on its model. All these attributes are connected to the Actuator Model class. hasDuration connects the Actuator Model class to the Time Duration class which represents the Time attribute of an actuator. hasSwtichState connects the Actuator Model class to the Switch State class which represents the power switch attribute of an actuator hasType connects the Actuator Model class to the Type Information class which represents the type of an actuator hasDuration connects the Actuator Model class to the Time Duration class which represents the Time attribute of an actuator hasRange connects the Actuator Model class to the Range Attribute class which represents the operating range of an actuator hasMultistep connects the Actuator Model class to the Multistep Attribute class which represents the multistep attribute of an actuator. connectsActuatorMiddleware connects the Device class to the Actuator Middleware class.

connectsActuatorMiddlewareToProvider connects the Actuator Middleware class to the Actuator Service Provider Class. connectsActuatorProviderToSupportToolbox connects the Actuator Service Provider class to the Actuator Support Toolbox class Content service, provider service and middleware service are the subclasses of the Actuator service provider class, and are connected to it through provideActuatorServices object property. Actuator support toolbox module

performs management of the information storeed in the ontology. This is represented by connecting the Actuator support toolbox class to the management class.

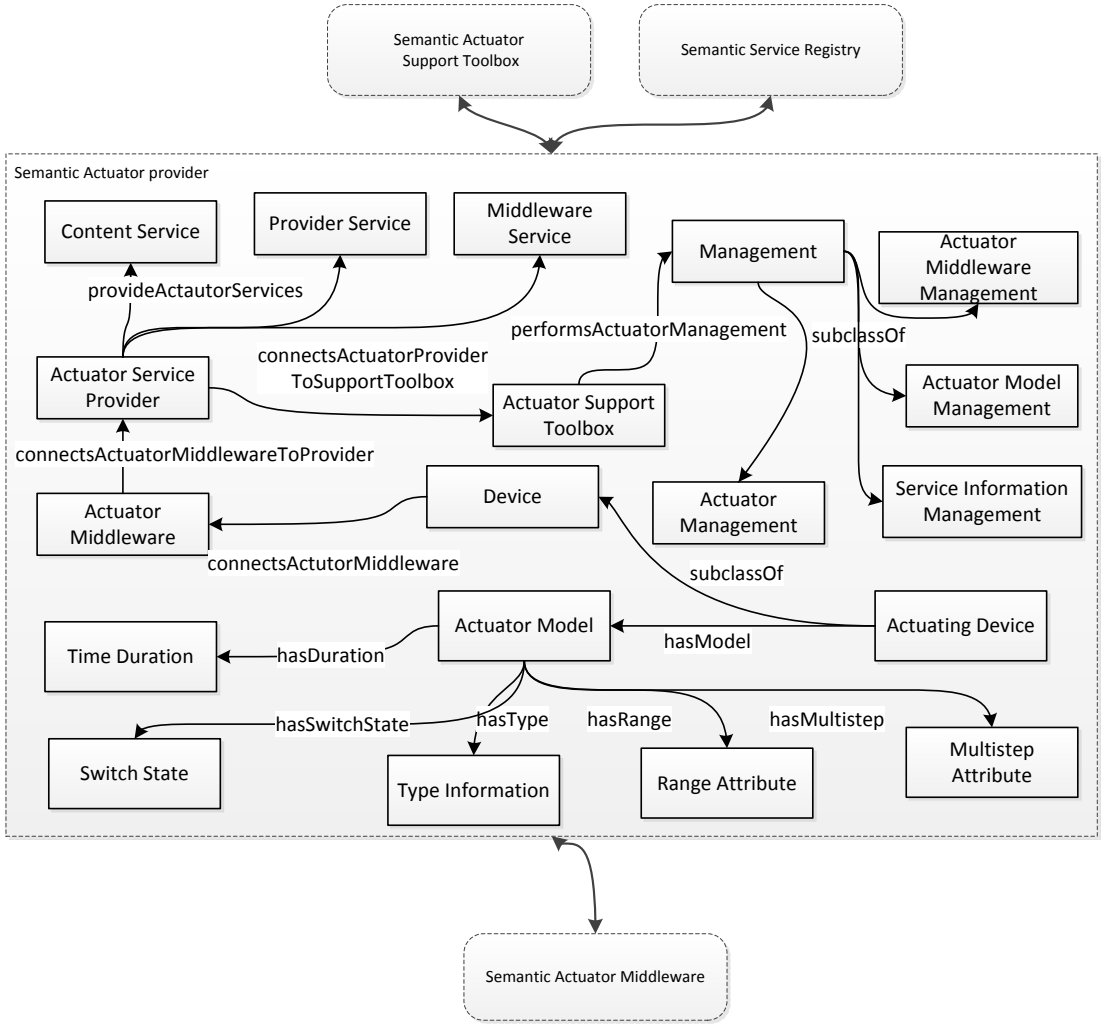

Figure 21 Actuator Ontology Model

Management class has three subclasses i.e. Actuator management class, Actuator middleware management class, actuator model management class, and service information management.

4.3 Semantic GIS Platform based on Indoor Location Information

4.3.1 Semantic GIS Service Provider

Figure 22 shows the detailed architecture of the semantic GIS service provider module. It presents the database and the ontology used by this module to store sensor and actuator's location data. Other modules use the services provided by this module using the semantic service interface through dotNetRdf API.

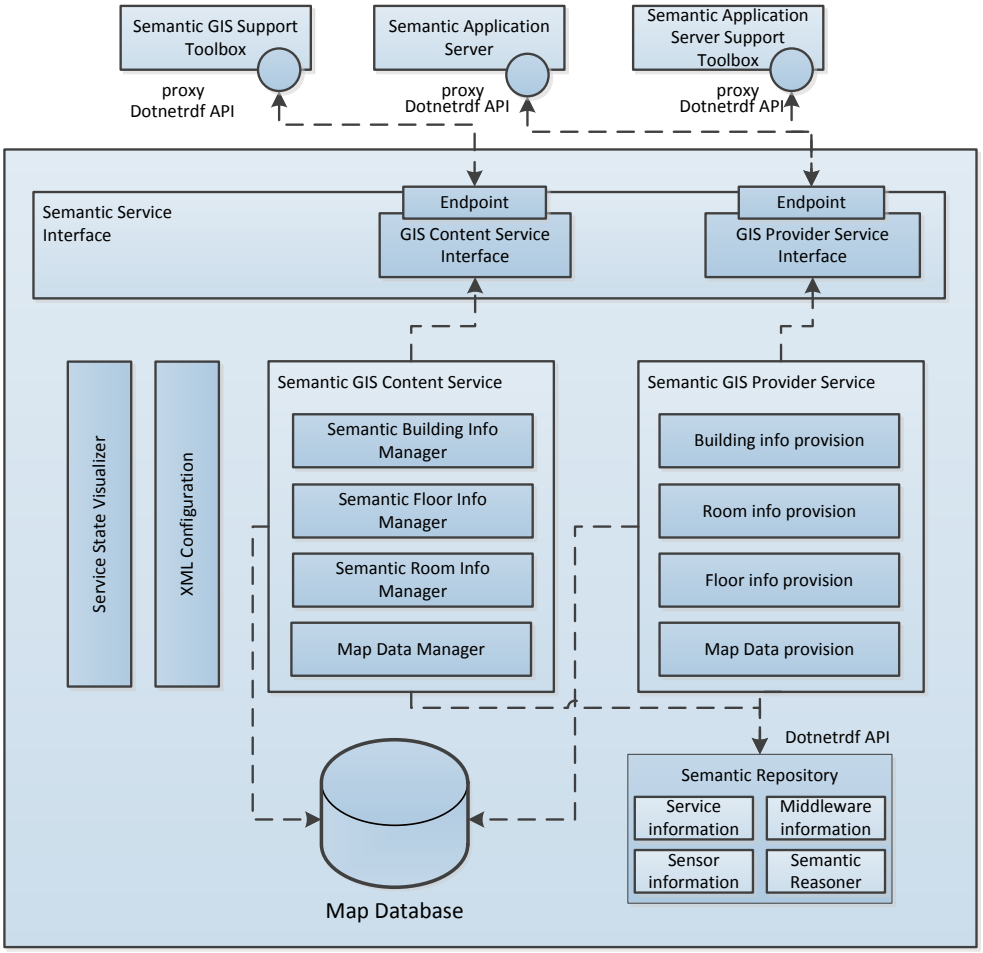

Figure 22 semantic GIS Service Provider

This module provides two services namely semantic GIS content service, and semantic GIS provider service. The former provides functionalities for management of building, floor, and room information. Map data manager performs the management of map images. Semantic GIS provider service performs map information provision, building information provision, floor information provision, and room information provision. Map data provision supplies the graphic map data from the database, whereas building, room, and floor data provision supplies building, room and floor information respectively from the ontology. Service state viewer shows the service operational state management.

4.3.2 Semantic GIS Support Toolbox

Figure 23 shows the detailed architecture of the semantic GIS support toolbox. It performs functions like creating service contents, managing these contents though semantic GIS content service and for registering GIS service information to the semantic service registry. For performing these functionalities it connects to the semantic GIS service provider and the semantic service registry. Building information generator creates and manages the building information (building name, building code) in the ontology. Managing includes deleting or updating the information. Building mark generator creates notations or markers to specify building area and location on the map. Floor information generator creates floor map information (floor no, floor name), and image loader creates mini map division and division map of the floor map image using the Map image file processor and stores it to the database. Room mark generator creates notations to specify the rooms on the floor map while room info generator creates room information (room name, room no, room explain) and stores it to the ontology.

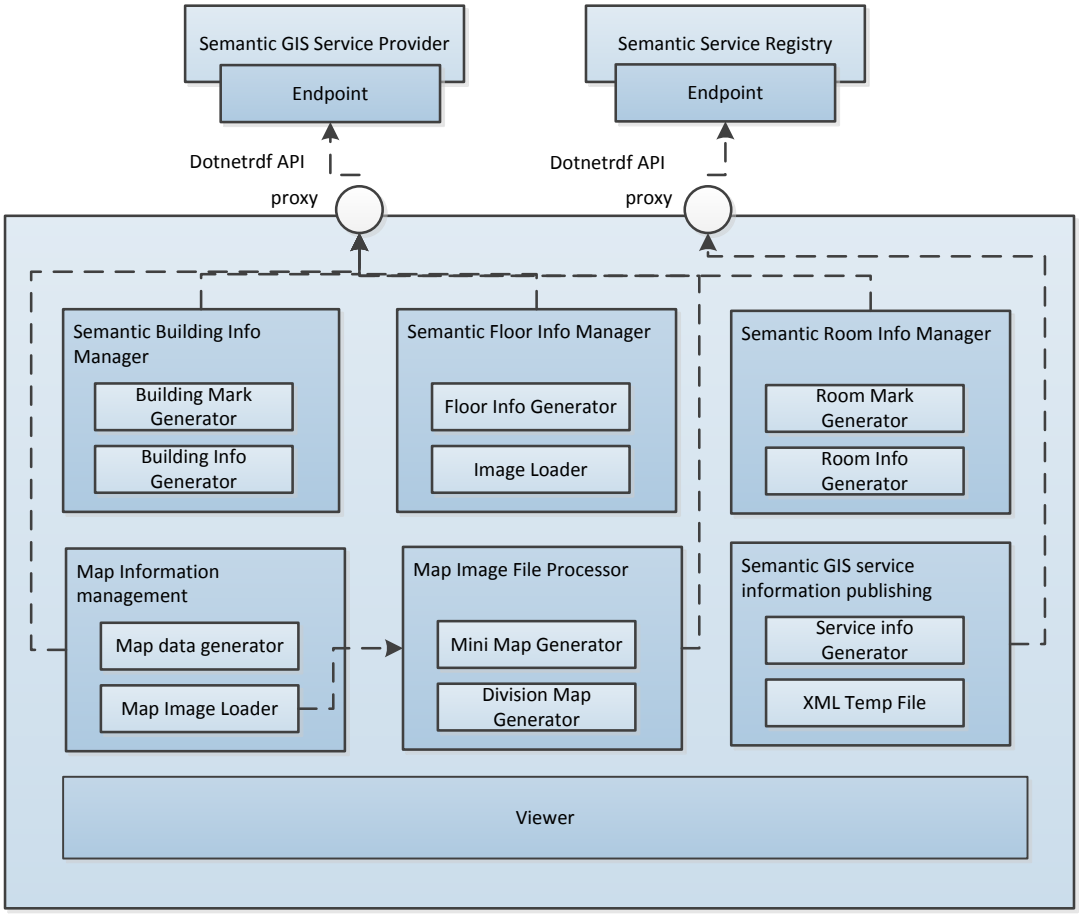

Figure 23 Semantic GIS support toolbox

Map image loader creates mini map and map division and stores it to the semantic GIS service provider database through the map image file processor. Map data generator creates total map information (map size, mini map size, etc.) and saves it to the database. Finally semantic GIS service information publishing creates service information (service name, service Uri, search key word etc.) through the Service info generator, saves it to the xml local file and registers it to the semantic service registry.

The sequence diagram for semantic GIS service provider shows how the semantic application server, and client interacts with this module to retrieve location information of a device. First of all the main window interacts with the GIS ontology model using dotNetRdf API. It gets the building,

floor and room count from the ontology and displays it to the user. Data manipulation methods can be run with the help of semantic GIS support toolbox.

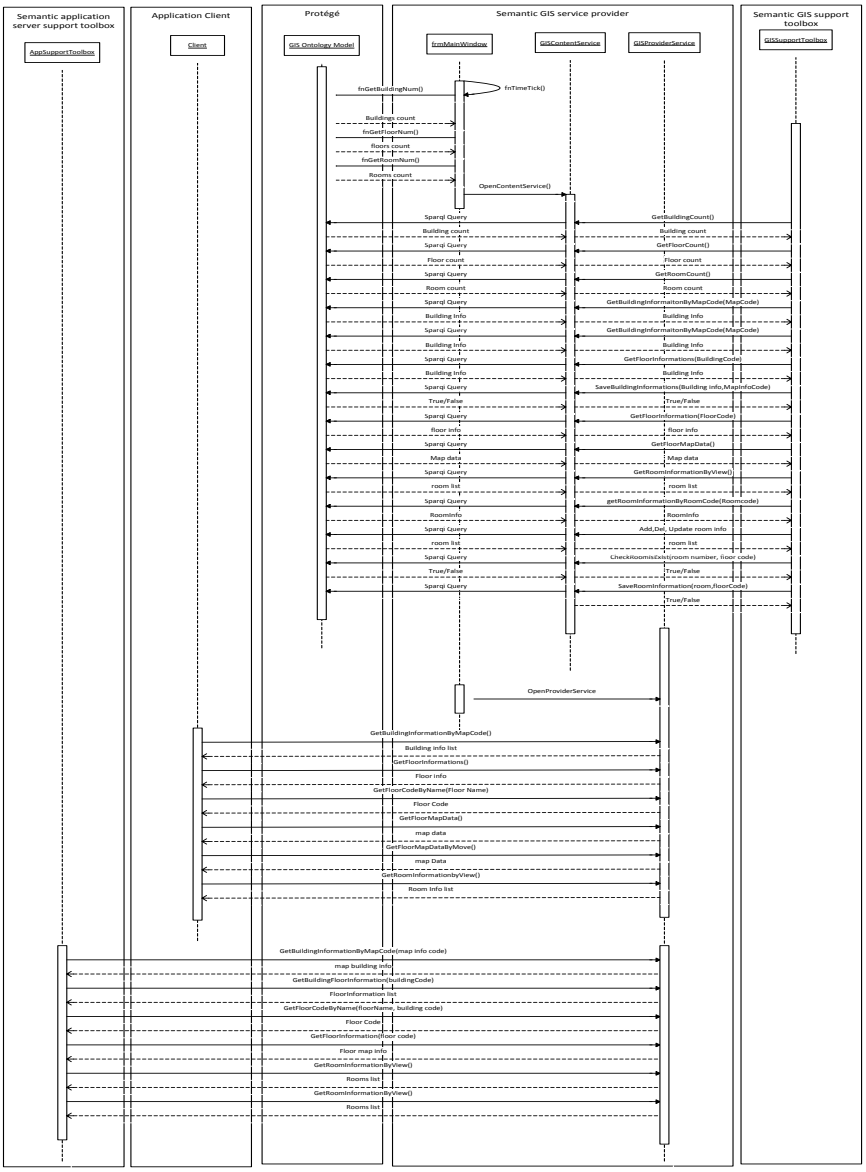

Figure 24 semantic GIS sequence diagram

Figure 24 shows the sequence of information (building, floor, and room) being added, retrieved, updated and deleted using the methods provided by the content service. For each method a SPARQL query is run on the ontology and the results are returned to the support toolbox module. The methods implemented in the provider service are used by the application server and client. The semantic application server support toolbox module and the application client uses the following

functions in the provider service: 1) Get the building information, 2) Get the room information, and 3) Get the floor information. First of all the main window interacts with the GIS ontology model using dotNetRdf API. It gets the building, floor and room count from the ontology and displays it to the user. Data manipulation methods can be run with the help of semantic GIS support toolbox. The figure shows the sequence of information (building, floor, and room) being added, retrieved, updated and deleted using the methods provided by the content service. For each method a SPARQL query is run on the ontology and the results are returned to the support toolbox module. The methods implemented in the provider service are used by the application server and client.

4.3.3 GIS Provider Ontology Modelling

Figure 25 shows the protégé generated Onto graph for the GIS Service provider ontology. It shows relationship between classes, subclasses, equivalence and object properties. Semantic IoT indoor system requires storing and representing the location information of the area it is monitoring. It uses the GIS provider module for this task. The classes defined in the ontology are building information, floor information, and room information, room management, building management, management, GIS support toolbox, GIS provider, provider service, and content service. Figure 26 shows the detailed ontology model for the GIS provider. It shows the classes, the object properties and the data properties related these class's instances. The object properties defined in the ontology are provides, performs, connectsGisproviderToSupportToolbox, managesRoomInformation, managesFloorinformation, managesBuildingInformation, hasRoom, uses, and hasFloor.

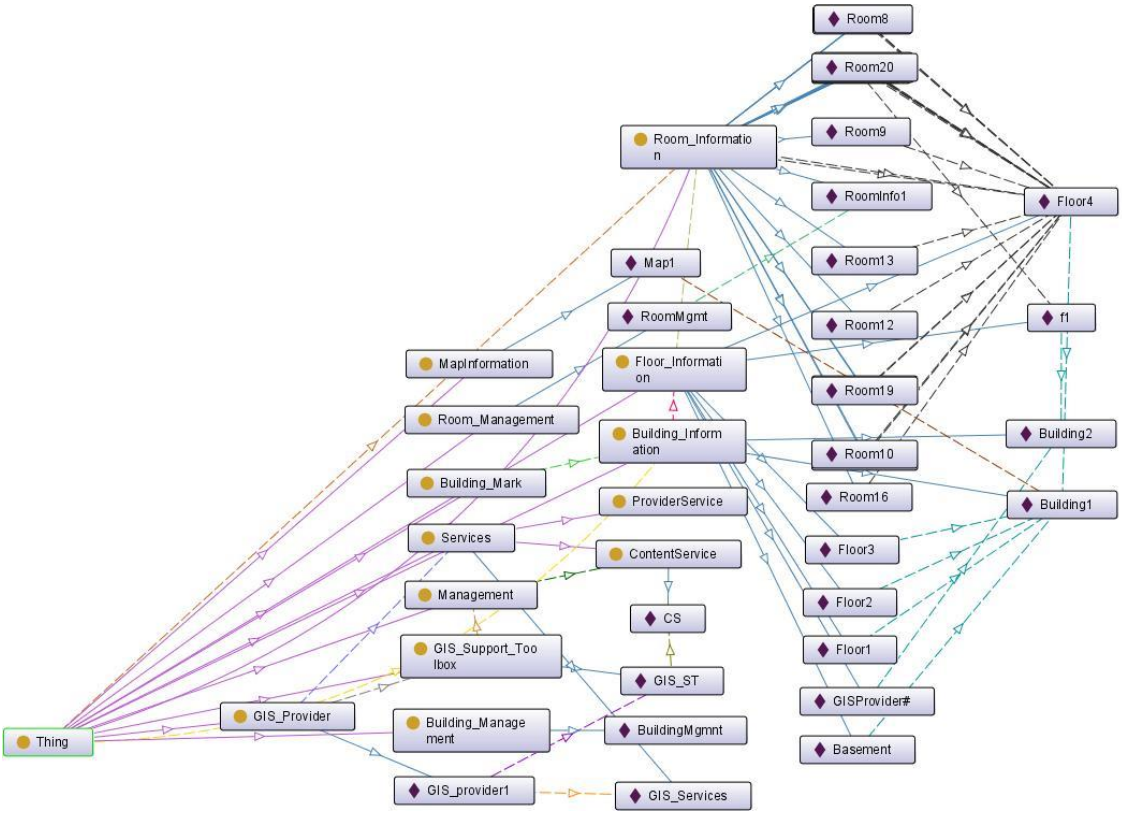

Figure 25 GIS provider ontology graph

The ontology model also shows the data properties and their data types which include Room Number (long), Room Code (long), Room Explain (string), Floor Code (long), Floor Name (string), Floor Explain (string), Floor Number (long), Building Code (long), Building Explain (string), Building Name (string), Content service and provider are the subclasses of the GIS provider class and are connected to it through the provideGisServices property. GIS Support Toolbox class is connected to the Management class through performsLocManagement class. The building management classes uses object property managesBuildingInformation to connect to the building information class. The room management class uses the object property managesRoomInformation to connect to the Room Information class. The floor management class uses the object property managesfloorInformation to connect to the floor Information class. hasRoom connects the Floor Information class to the Room Information class. hasFloor connects the Building Information class to the Floor Information class.

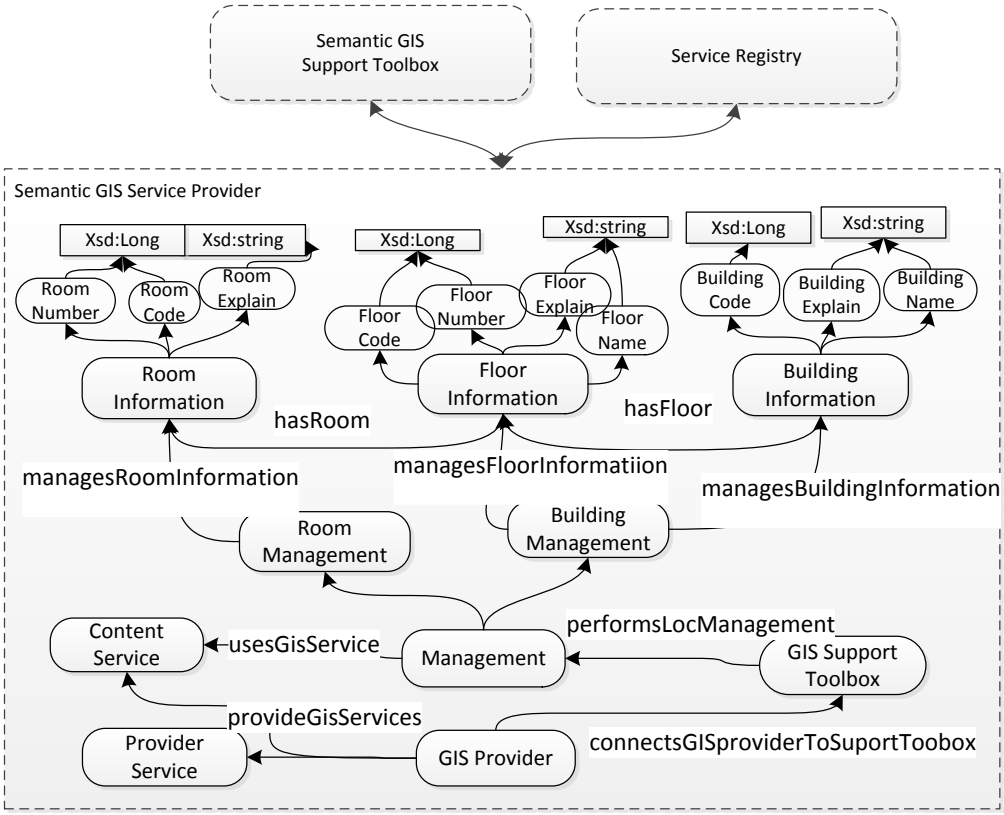

Figure 26 GIS provider ontology model

4.4 Semantic Service Registry

Figure 27 shows the service registry detailed architecture. Service registry is composed of Publish Service and Search Service. It offers publish service to each system composite tool for registering the provider service information Service information includes service name, service type, service address, service search key word Search Service is the registry interface for other system to search service information. Semantic Service Control controls the operation of Publish Service and Search Service (on/off). Semantic Publish Service controls whether to show a registered service as available or to hide it. XML Configuration offers WCF Service location and other information. Semantic repository saves registered service information.

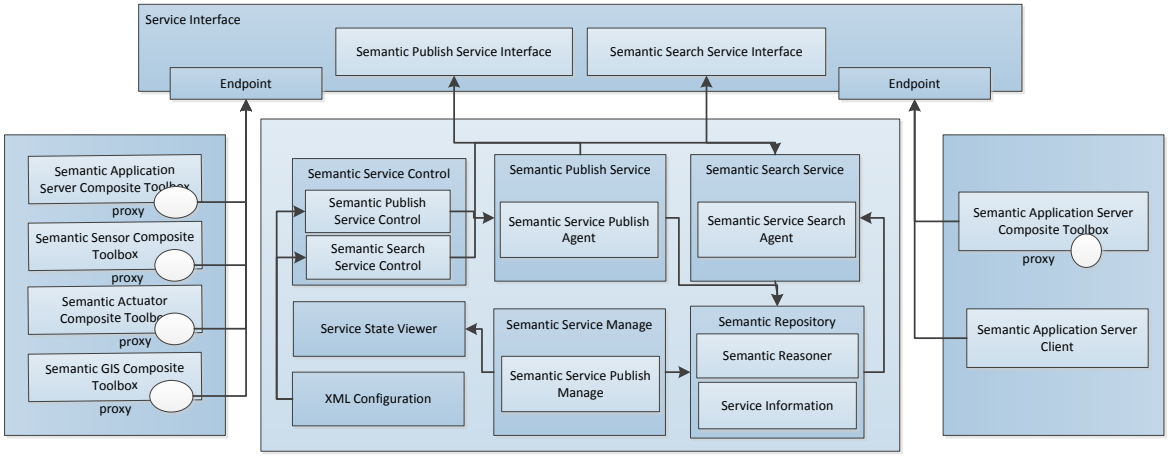

Figure 27 Service Registry Configuration Diagram

Figure 28 shows the interaction between the service registry, service registry ontology model, and the other modules of the system. Semantic service registry provider provides a publish service that allows the service providers to publish their service information to the service registry ontology. The service publish is used by the provider service of the semantic GIS server support toolbox, semantic actuator support toolbox and semantic sensor support toolbox module. All these modules publish their service information to the service registry ontology through the publish service. The semantic app server module interacts with the service registry to use the search service. The search service provides a list of the services registered in the service registry ontology to the app server module. The service registry module interacts with the service registry ontology model to publish service information and to search for services published using the dotNetRdf API.

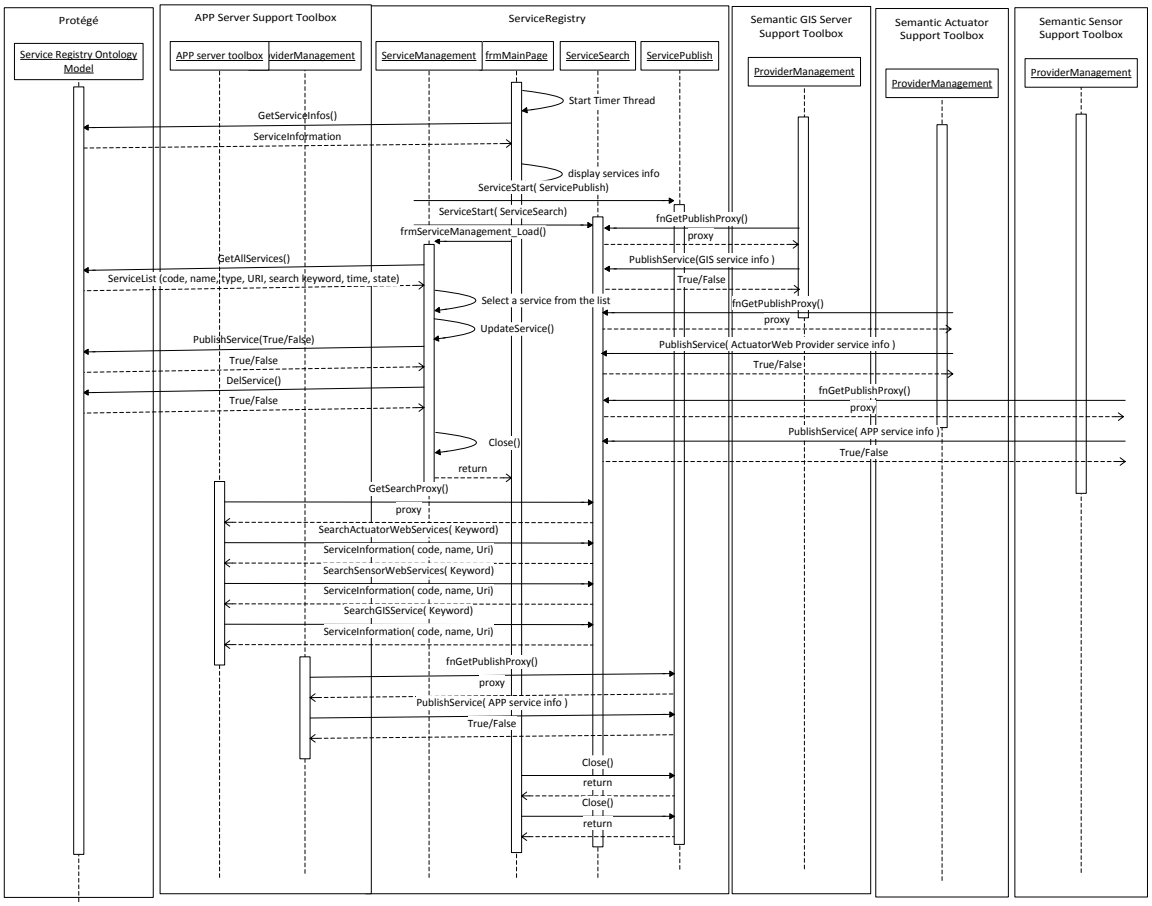

Figure 28 Service Registry Sequence Diagram

4.4.1 Service Registry Ontology Modelling

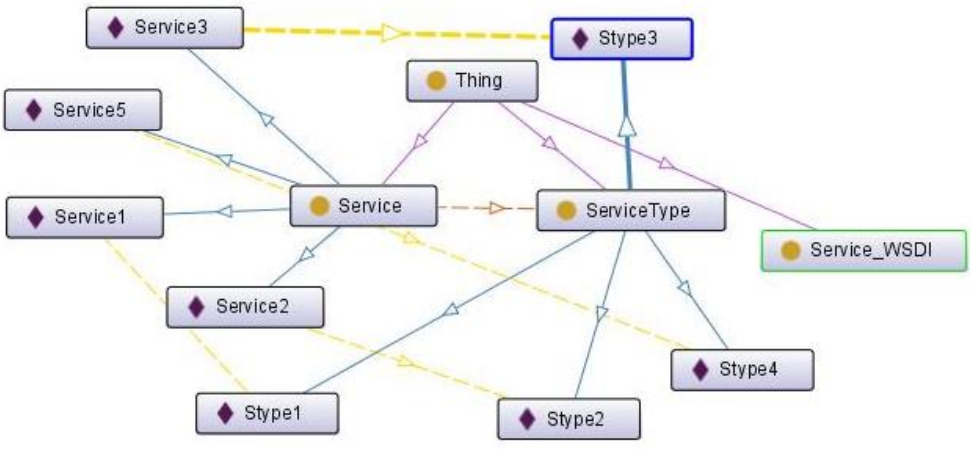

Figure 29 Service Registry Ontology Graph

Figure 29 shows the onto graph generated for the service registry ontology. It describes the classes defined for representing the concepts of service registry module. The classes in the ontology are Service Information, Service Type, and Service WDSL. The service registry ontology model is shown in Figure 30. It is a detailed model showing the data properties as well as object properties in the ontology. As shown Service information class includes the information of the services published in the service registry module by Sensor, Actuator, and GIS provider module. Service type class includes the type of service that is available. WSDL is an XML format for describing network services as a set of endpoints operating on messages containing either document-oriented or procedure-oriented information. Service WSDL class shows the availability of the WSDL document for a service registered in the ontology.

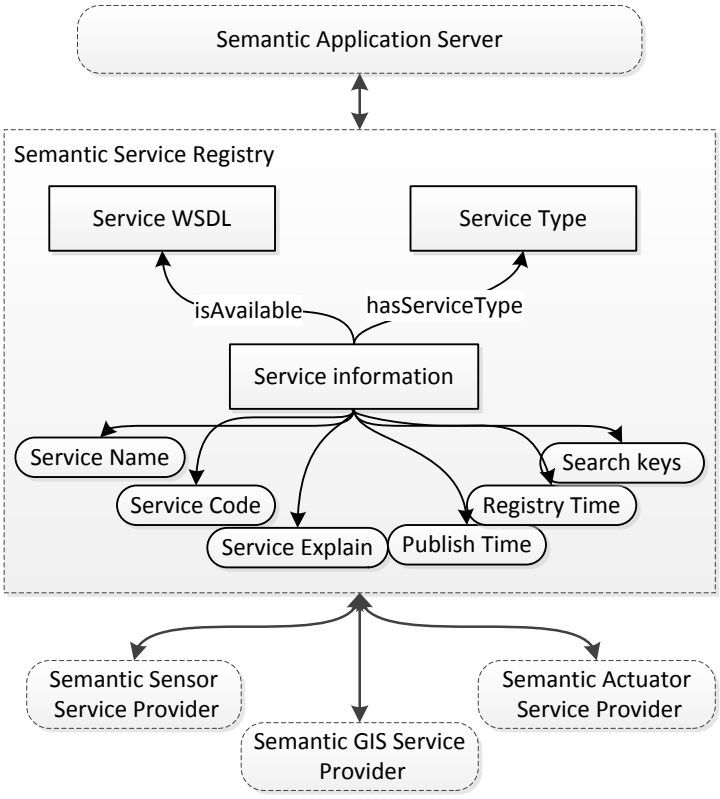

Figure 30 Service registry ontology model

4.5 Semantic Application Server

The detailed architecture of Semantic Application Server module is shown in Figure 31. It shows the API's used by this module to connect to the rest of the subsystems. To provide sensing data, and sensor information to the client it uses the dotNetRdf and the sensor provider API. For environment control and actuator information provision it uses the dotNetRdf and actuator provider API.

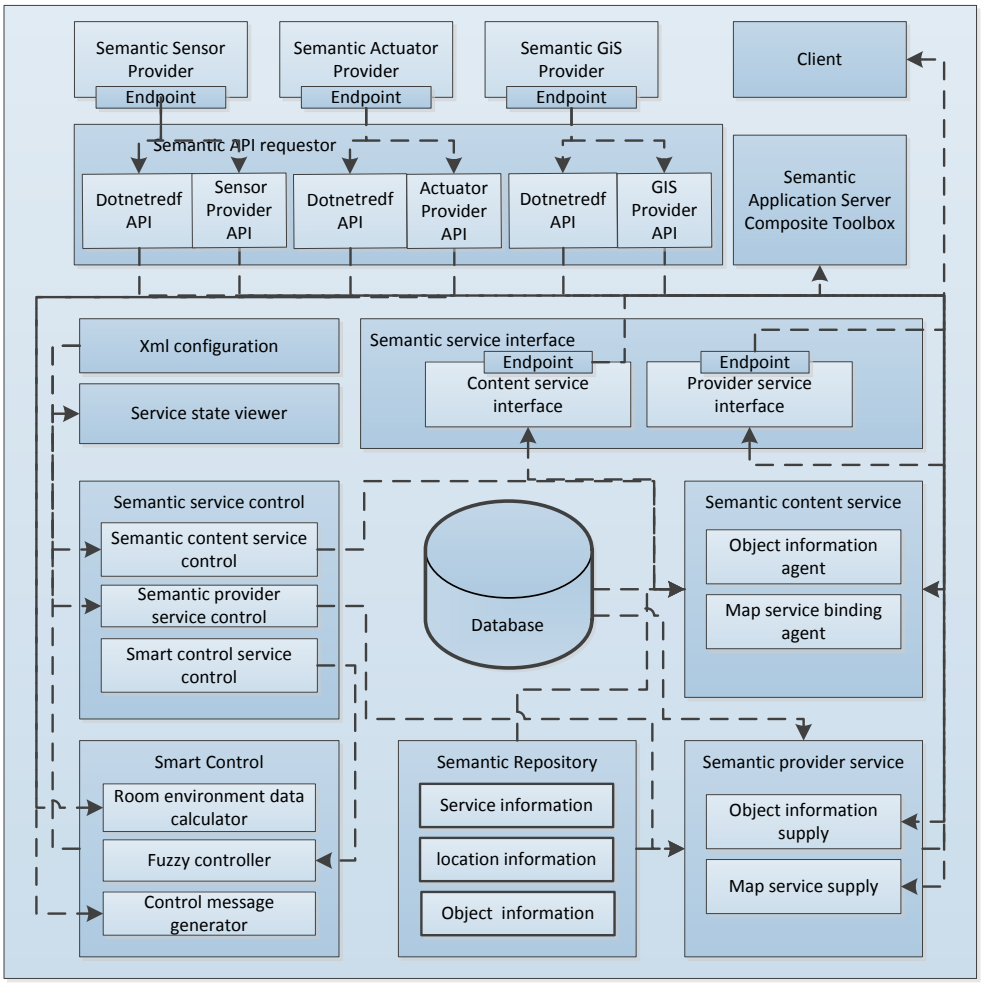

Figure 31 Semantic Application Server Configuration Diagram

Similarly for map image provision and location information it uses the dotNetRdf and GIS provider API. Content service interface is used to connect to the application server support toolbox for data manipulation and processing. The client connects to the semantic application server module

through the provider service interface. The core of the semantic application server module is the smart control unit. It requests the sensor provider module for state of a specific area by utilizing the location information. Depending on that state it controls the environment of the area using actuator provider module. Semantic Application Server offers three services provider service, content service, and smart control. Provider service offers data provision to the client.

It provides complete object information i.e. location information, type, map service, and Service Uri. Content service is utilized by semantic application support toolbox for object information management and map service binding. Smart control requests environment data for a target area from semantic sensor provider, calculates the environment state, and controls the state through fuzzy controllers. Control message generator depends on the control contents to create control messages to be sent to the semantic actuator provider. XML configuration offers information related to WCF services and dotNetRdf API's. Database is used to store the map images retrieved from the semantic GIS provider, whereas semantic repository contains service, location and object information.

4.5.1 Semantic Application Server Support Toolbox

Figure 32 shows the application server composite tool detailed configuration diagram. This module provides visualization of map and objects to manage map service binding and object location binding. First the management searches GIS Service for specific map service from service registry and binds it. Next, the Total Map Viewer requests all map data and building information from GIS provider. Management then selects a building floor on the map viewer and Floor Map Viewer shows the floor map data, room information and objects associated with the floor map.

Object Binding Management provides the functionality of Sensor Node Binding and Actuator Node Binding to map locations. Info Generator creates Service information (search key word, name, address etc.) and the App Server Info Publish saves it to XML Temporary File and then registers it

to service registry. The sequence diagram shown in Figure 33 illustrates the interaction between the semantic app server module and the other modules of the system.

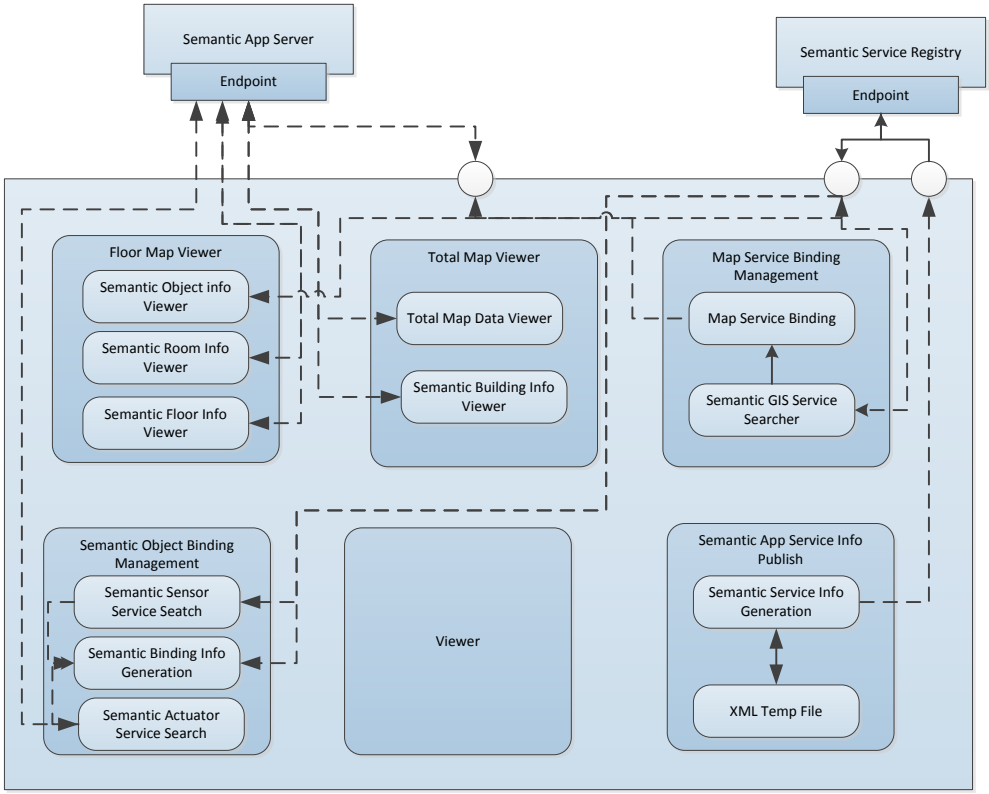

Figure 32 Application Server Support Toolbox Configuration Diagram

First of all the module is initiated by running the main window, which starts the timer. The application server uses the semantic sensor and semantic actuator proxy's to call the following functions: Get sensor num (), Get actuator num ().The room control service is also started which follows the following steps: Gets the object information from the ontology model based on the room code that is selected, Creates fuzzy system using fuzzy variables and rules, Gets the information of the sensor registered in the room from the semantic sensor provider, Gets the sensing state of the registered sensor, Gets the sensing data of the generated by that sensor , Processes the sensing data and finds the average, Gets the season information from the ontology, Calculates the room comfort index using PMV, PPD and ET indices and saves the room comfort index calculation details.

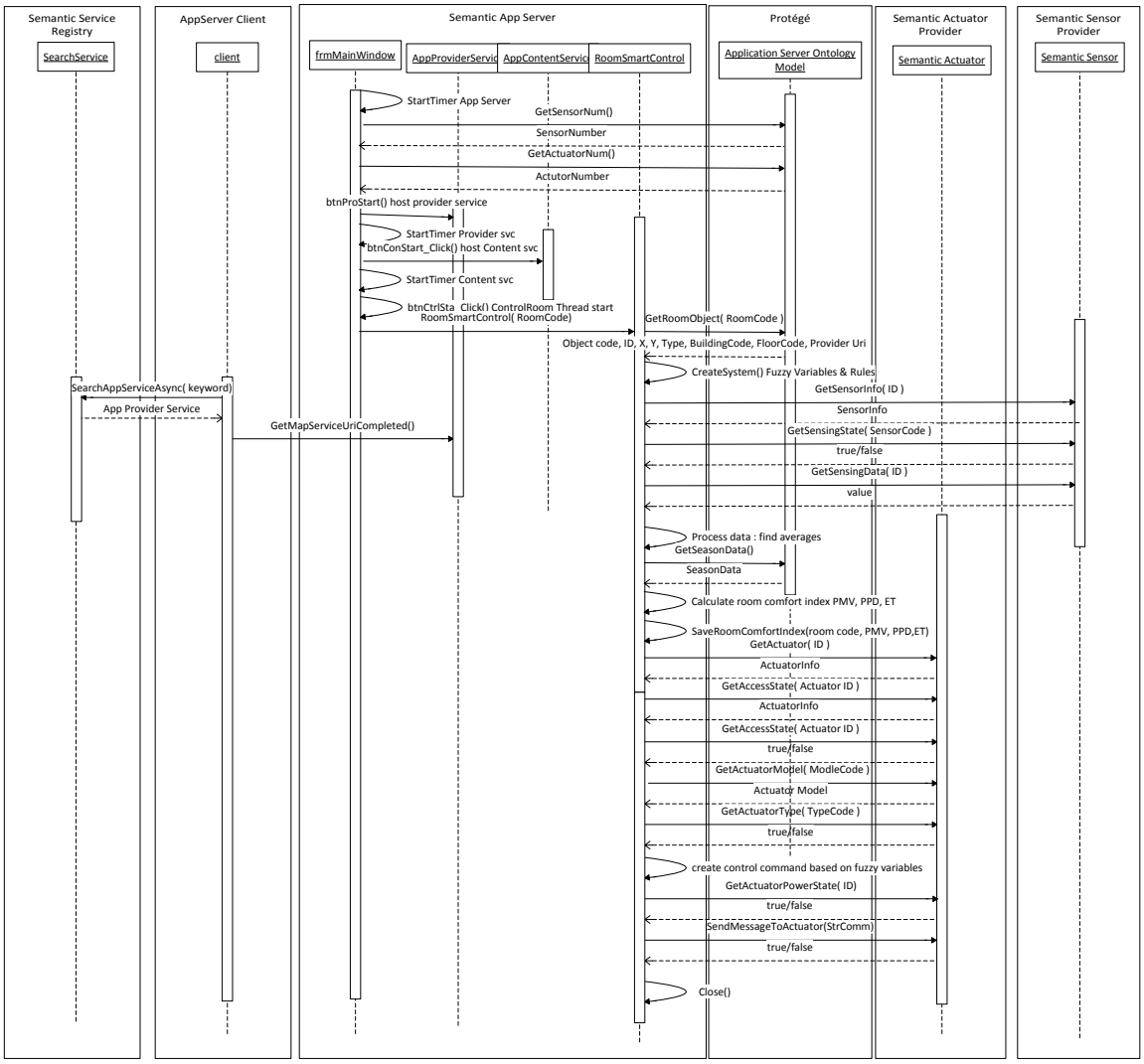

Figure 33 Semantic App Server Sequence Diagram

Next it gets: the information of the actuator registered in the room, the model and type of the actuator, creates control command based on fuzzy variables, and gets actuator power state, if it is not according to the comfort index then send the control command to the actuator.

4.5.2 Smart Control

As mentioned before the application server offers smart control service through which it automatically controls the actuators to maintain a pleasant indoor environment. Sensor nodes connected to sensor network sends sensing data to the semantic service provider in real time via sensor middleware and save it. Depending on the sensor network connected and actuator

connecting condition, actuator middleware and semantic actuator service provider creates mapping table and manages it. Application server requests indoor environment sensing data (temperature, humidity, illumination etc.) based on the area and object information it has in its semantic repository. Next, it receives environment data and through fuzzy control module, it creates control object (actuator) and control command to execute. Semantic actuator service provider receives the control message from application server and according to its mapping table with actuator routing information through actuator middleware, it forwards the message.

4.5.3 Smart Control Concept Design

Figure 34 shows the course of collecting indoor environment. It has already been explained that sensor service provider provides sensing data along with an external service interface to get that data and that Application server ontology has the sensor node location information and indoor space information. Using this information it is to find sensor nodes in a particular area. Smart Control module gets sensor node sensing type information based on a particular space because it is possible that multiple sensor nodes performs the same function in the same space. For such case obtain the average of data and identify the state of environment. Even single sensor detection can operate the entire system properly. Send final calculation related to an area's environment i.e. average temperature, humidity and illumination information to Controller which will process it for the next step.

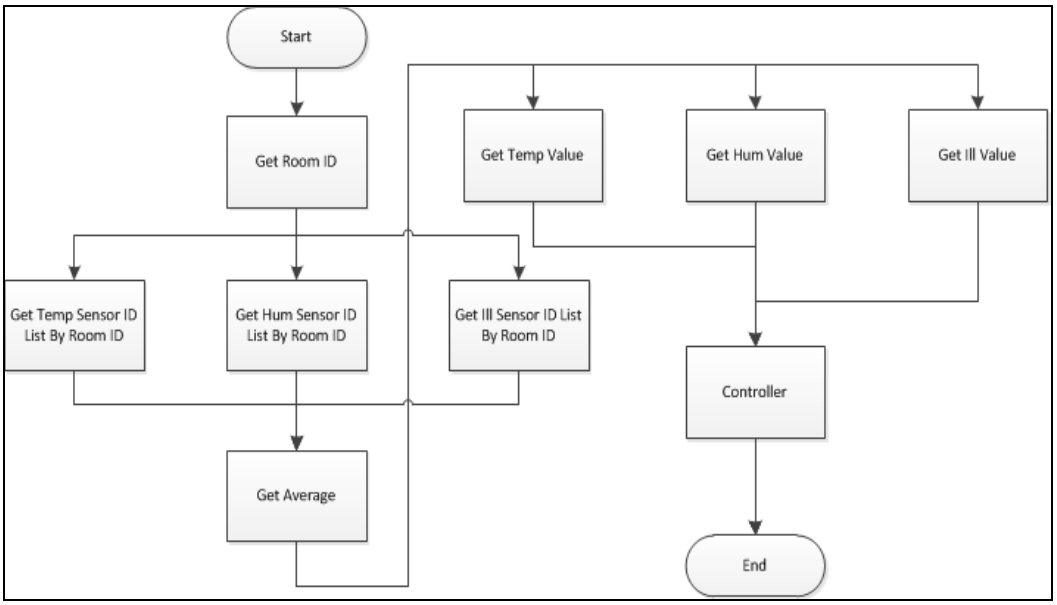

Figure 34 Indoor environment calculation

Figure 34 shows the course of controlling indoor actuator. From the sensor service provider, the environment information for a specific space is received. Then comfort index PMV and ET index is computed. Using table 5 describing the fuzzy rule, get the combined comfort state as a result which is used for controlling fan, Air conditioning and boiler etc. Temperature adjustment, controlling light appliance through illumination information and adjusting the humidifier through indoor humidity conditions. Comfortable indoor environment for normal human illumination is maintaining 1000lx, for humidity the range is 50%~60%.

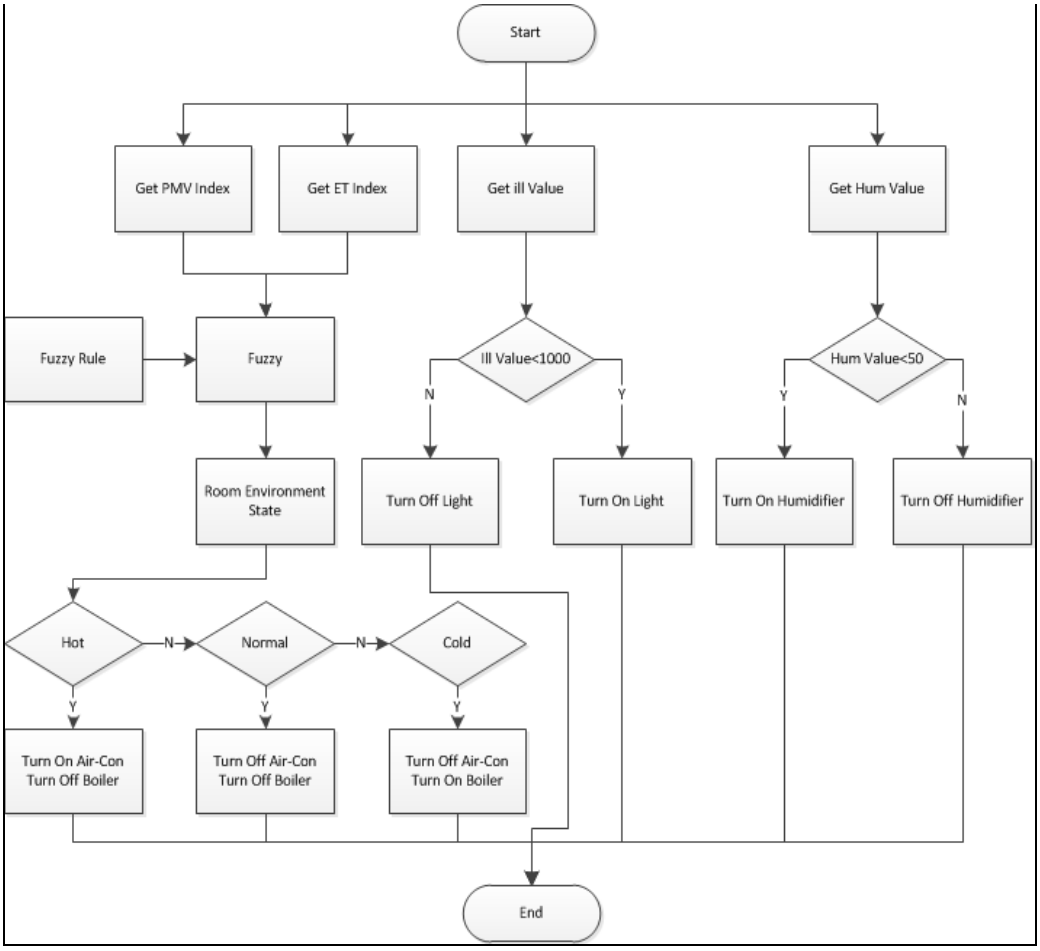

Figure 35 Indoor actuator control process

4.5.4 Application Server Ontology Modelling

This section provides graphs and models to illustrate the basic concepts, object properties, and data properties of the app server ontology. Figure 36 represents the basic classes and relationships of the ontology in the form of a graph generated by protégé. The classes included in the Application server ontology are object information, location information, rule information, application server provider, content service, provider service, application support toolbox, object information management, and location information management.

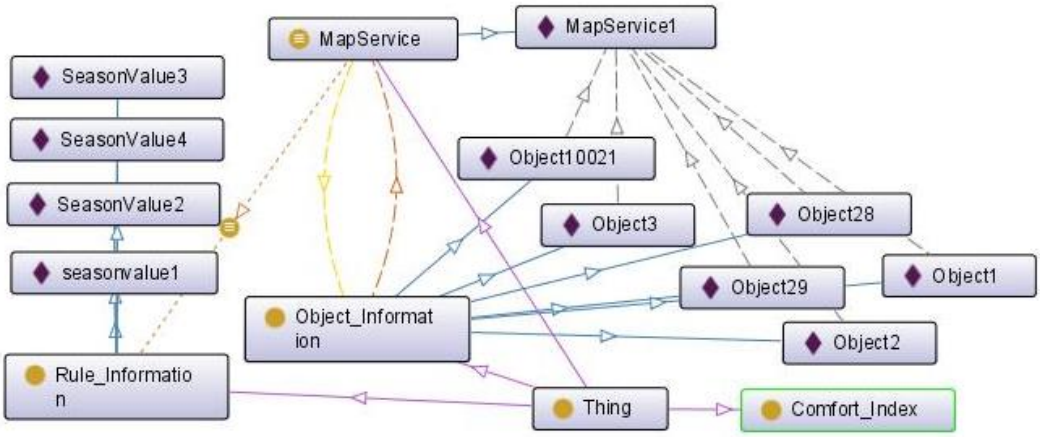

Figure 36 App Server Ontology Graph

The model in Figure 37 is the ontology model of the semantic application server module. Object information represents the combined information of a resource (sensor or actuator) and its location. The information stored about an object is 1) Object type: the type of a resource, whether it is a sensor or an actuator, 2) Object id: the unique identity of an object, 3) Provider Uri : the Uri of the object's service provider, 4) Object code: the code of the object, 4) Room code: the code of the room in which the object is located, 5) Floor code: the code of the floor on which the room is, and 6) Building code: the code of the building in which the floor is.

The object properties connecting these classes are createRuleInfo, createLocInfo, createObjInfo, providesAppServices, manageslocInfo, managesObjInfo, performObjManagement, and performLocManagement. createRuleInfo connects the content service to the Rule Information class. createLocInfo connects the content service to the map service information. The map service information consists of the following data properties 1) Map code: the code of the map that is registered in the database and, 2) Map service URI: stores the URI address of the map service. The application support toolbox connects to the object information management and the map service management through the performObjManagement object property and performLocInfo object property.

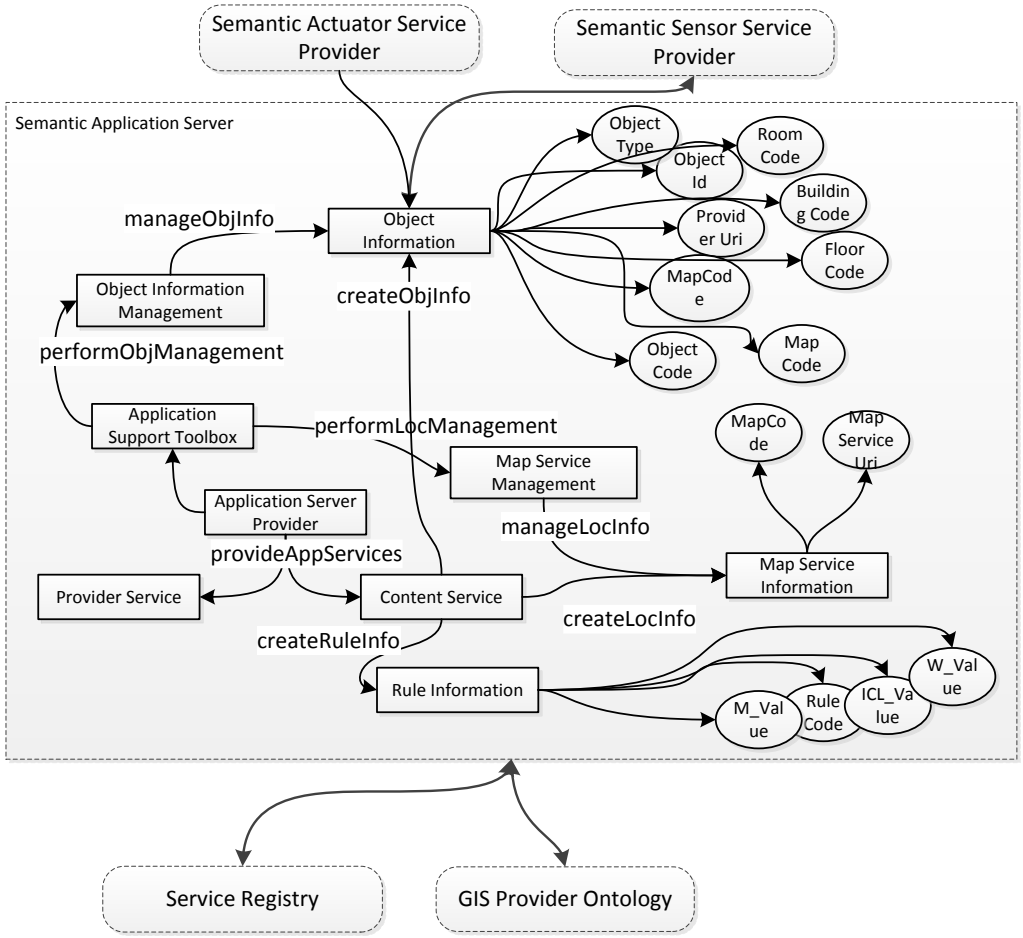

Figure 37 Application Server Ontology Model

Object Information Management class is connected to the object information class through manageObjInfo object property. It is responsible for adding an object, updating an existing object and deleting an object. manageLocInfo represents the relationship between the map service information class and the map service management class. Application server provider class uses provideAppServices property to relate to the services it provides.

4.6 Semantic Application Client

Figure 38 shows the application client detailed configuration diagram. Client offers a simple visualization. First, the App Service Searcher searches application server from service registry and accesses selected service.

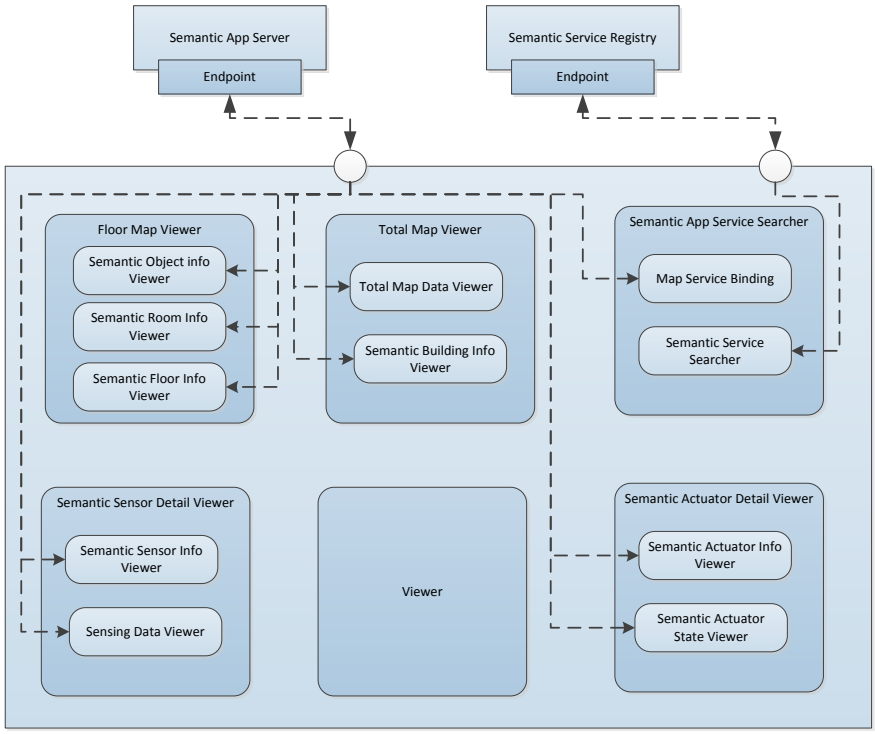

Figure 38 App Server Client Configuration Diagram

Then Map Service Binding requests Map Service information from application server and binds it. Next, Total Map Viewer requests all map data and building information from GIS provider. Management selects a building floor on the map viewer and Floor Map Viewer shows floor map data, room information and objects. When user selects sensor object, Sensor Info Viewer displays the sensor name, ID, Sensing Type and Attributes.

The sensing data Viewer displays real time sensing data. If the user selects actuator object, Actuator Info Viewer displays actuator name, ID, actuator type, model information and Attribute etc. The sequence diagram for the App Server client shown in Figure 39 describes the internal sequence of interactions among the internal objects of the client module and the interaction with

other modules such as App Server, Service registry, GIS provider and Sensor web provider. The client uses the search service of the registry to get service list of available App server provider service. Client connects to a server provider to get the binding information.

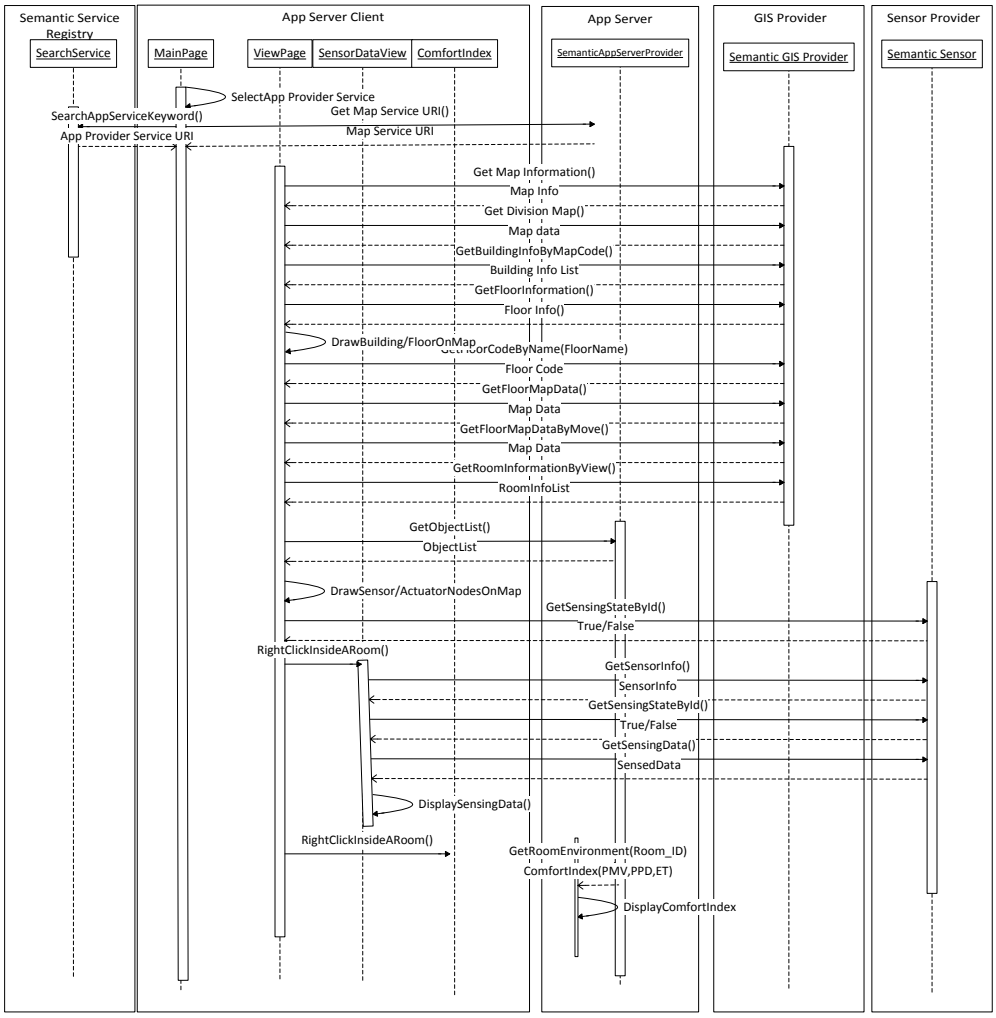

Figure 39 App Server Client Sequence Diagram

Client connects to the GIS service to: Get map information, Get map data, Get building information list, Get floor information list, Get floor map data, Get room information list, Client connects to the App Server to get the object list. After getting the object list, the client displays sensor and actuator node based on the list. By right clicking on a node the client connects to the specific service provider to: Get the state, Get the node information list, Get the node data, and to display the node data. By clicking inside a room the client connects to the room comfort index which calculates the room

comfort index of the room and returns to the client. The client then displays the comfort index to the user.

5 Implementation and Performance Analysis of Semantic IoT System

Advance research is being carried out by OGC (Open Geospatial Consortium) related to sensor web and semantic web technologies that report real-time context state using the information collected from various sensors in a sensor network. OGC members are specifying interoperability interfaces and metadata encodings that enable real time integration of heterogeneous sensor webs into the information infrastructure. The SSW annotates sensor data with spatial, temporal, and thematic semantic metadata. This technique builds on current standardization efforts within the Open Geospatial Consortium's Sensor Web Enablement (SWE) [29] [30] and extends them with Semantic Web technologies to provide enhanced descriptions and access to sensor data. The real-time context reported by these sensors is managed using the concept of SOA (Service Oriented Architecture). Recently, OGC research interests not only on outdoor, but also indoor sensor web standardization.

In addition, recently IoT technology is being developed as a strategic industry in major countries of the world such as Europe, China, America, Japan, and South Korea. The aim of IOT is to interconnect objects, which have their own addresses based on standardized communication protocol, located world-wide. In particular, IoT architecture, communication model, application of business model, mutual and test model construction etc. are presented in the 7th framework program (FP7) in Europe. In this study we have developed a semantic IoT indoor system based on semantic sensor module as well as semantic actuator module. Semantic sensor module provides services that uses sensors to collect the context information of the indoor environment, semantic actuator module provides services to intelligently control an object according to the environment of the world. We also present GIS service module that uses semantic technologies to store and represent the spatial information of the physical space and is used to locate sensors and the actuator.

The application server module is the top most module that can use the services provided by the bottom modules and display them to user using client application. Table 1 shows the requirements for implementing this system. For developing the modules we used WCF services which is a framework for building service-oriented applications. Using WCF, you can send data as asynchronous messages from one service endpoint to another. The WCF programming model provides various capabilities, such as SOAP services, web HTTP services, data services, rich internet application (RIA) services, and workflow services. In this study we are using SOAP services. SOAP services support interoperability between systems that are built with Java, other platforms, and those that use messaging standards that are supported by Microsoft®.

Table 1 System Implementation Requirements

Implementation requirements	
Tools	Version
Protégé	4.3.0 (Build 304)
Pellet Reasoner	2.2.0 (Plugin)
OWL API	3.4.2
Protégé SPARQL	1.0.0 (Plugin)
Microsoft .NET Framework	4.0 full
Microsoft SQL Server Management Studio	11.0.2100.60
Operating System	Windows 7 (64 bit)
Processor	Equal or above 3.30GHZ
Microsoft Visual Studio	Community 2015
dotNetRdf API	1.0.9(Build 1.0.9.3683)

We have developed the modules using Microsoft Visual Studio Community 2015. Microsoft SQL Server Management Studio is used for creating the database to store map image information, and real time sensor data. For developing the ontologies we have used Protégé which is a free, open-source platform that provides a growing user community with a suite of tools to construct domain models and knowledge-based applications with ontologies. Protégé's plug-in architecture can be adapted to build both simple and complex ontology-based applications. For performing reasoning on the ontologies to infer new relations we have used the Pellet reasoner. A reasoner is

a piece of software able to infer logical consequences from a set of asserted facts or axioms. The notion of a semantic reasoner generalizes that of an inference engine, by providing a richer set of mechanisms to work with. The inference rules are commonly specified by means of an ontology language.

5.1 Semantic Sensor Service Provider

5.1.1 Implementation of Semantic Sensor Service Provider

This sections presents an array of figures to show the execution of the semantic sensor service provider module. It consists of semantic sensor service provider service manager, semantic sensor support toolbox control panel, semantic sensor management, semantic middleware management, and semantic service publish. Figure 40 shows the service manager interface for the semantic sensor provider module. It describes the service start up and end process. It consists of start and end buttons for the services that are available.

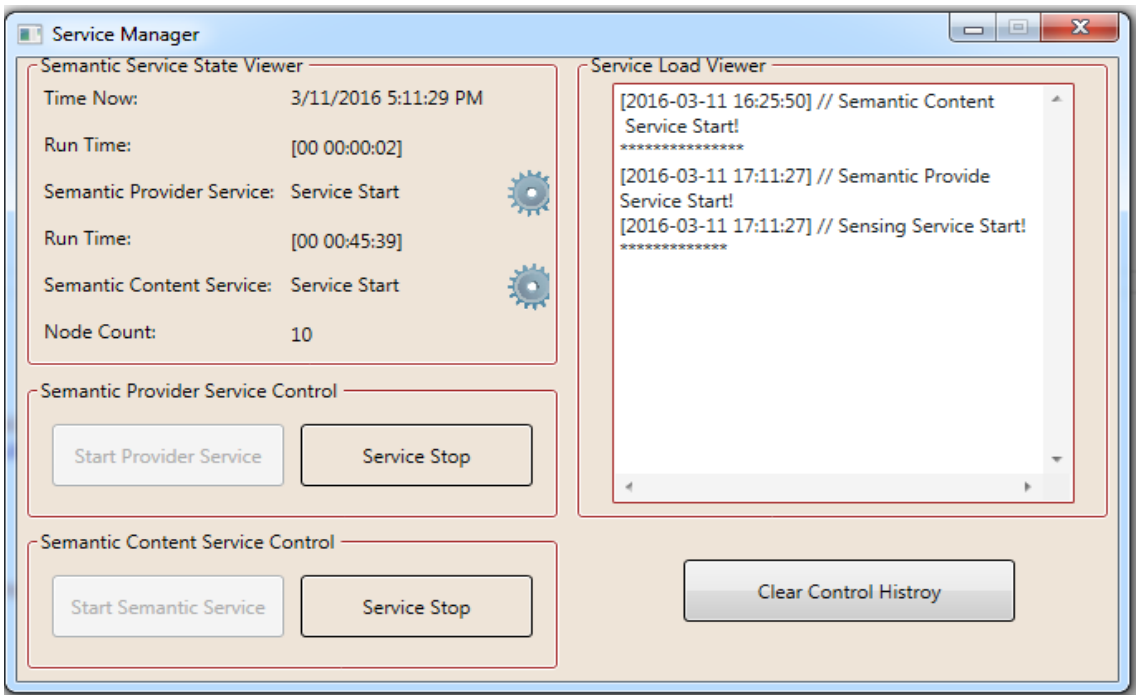

Figure 40 Semantic Sensor Service Provider Manager

It consists of two viewer's i.e. semantic sensor state viewer and service load viewer, and two controls i.e. semantic content service control and semantic provider service control. The semantic sensor state viewer shows the running state of both the services, it records the time for which the services are in the running state and it also shows the total number of sensors that are registered in the semantic registry of the sensor provider module. Semantic provider service control has the controls to start and stop the semantic provider service. The provider service start control starts both the provider and the sensing service. With the sensing service sensing data based on category is collected from attached sensors and stored in the database. The provider service implements functions for getting the sensing category of the sensors from the sensor ontology. Service load viewer keeps log of the service start up and stop timings in a list. This log can be cleared by the user by using the control clear control history. Once the content service is initiated the data in the ontology can be manipulated using SPARQL queries and dotNetRdf API.

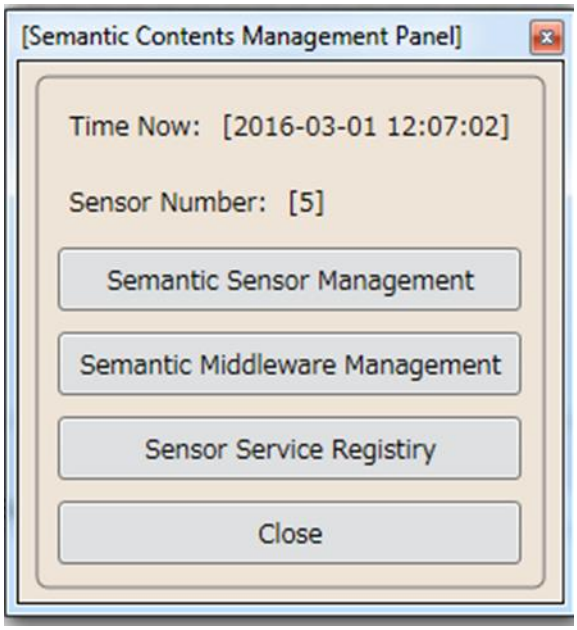

Figure 41 Semantic management control panel

Figure 41 shows the management panel for manipulating different data in the sensor semantic registry. It consists of a label that shows the no of sensors registered in the ontology. It also consists of four button controls through which the respective windows can be accessed. The windows accessible from this panel are semantic sensor management, semantic sensor middleware management, and semantic service registry. Figure 42 shows the form for the sensor management in the semantic registry of the sensor provider module. It provide controls that allows the user to manipulate sensor context data in the ontology. As the window loads it retrieves sensor ids from the semantic registry and show the results in a list on the left side of the form.

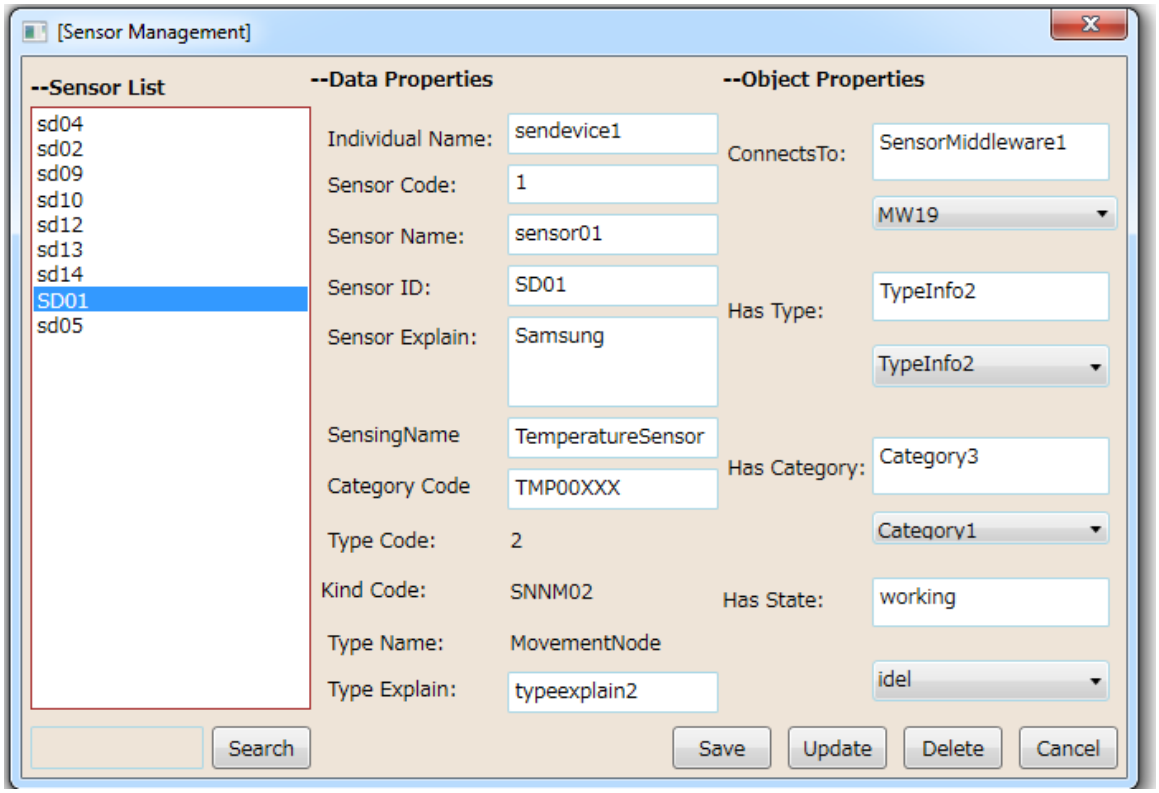

Figure 42 Semantic sensor management

As the user selects a sensor from the list its properties are displayed in the textbox controls of the form. The semantic sensor management form is arranged according to the way data is stored in the ontology. On the right hand side of the form it displays the object properties of the selected sensor. In the middle of the form the data properties of the selected sensor are displayed.

Figure 42 shows how this data is displayed in the form. All the data is retrieved from the ontology using SPARQL queries and dotNetRdf API. Content service also provide functions for storing new sensor information, updating and deleting existing sensor, and middleware information from the ontology. All these functionalities are performed based on the unique id of each sensor in the ontology.

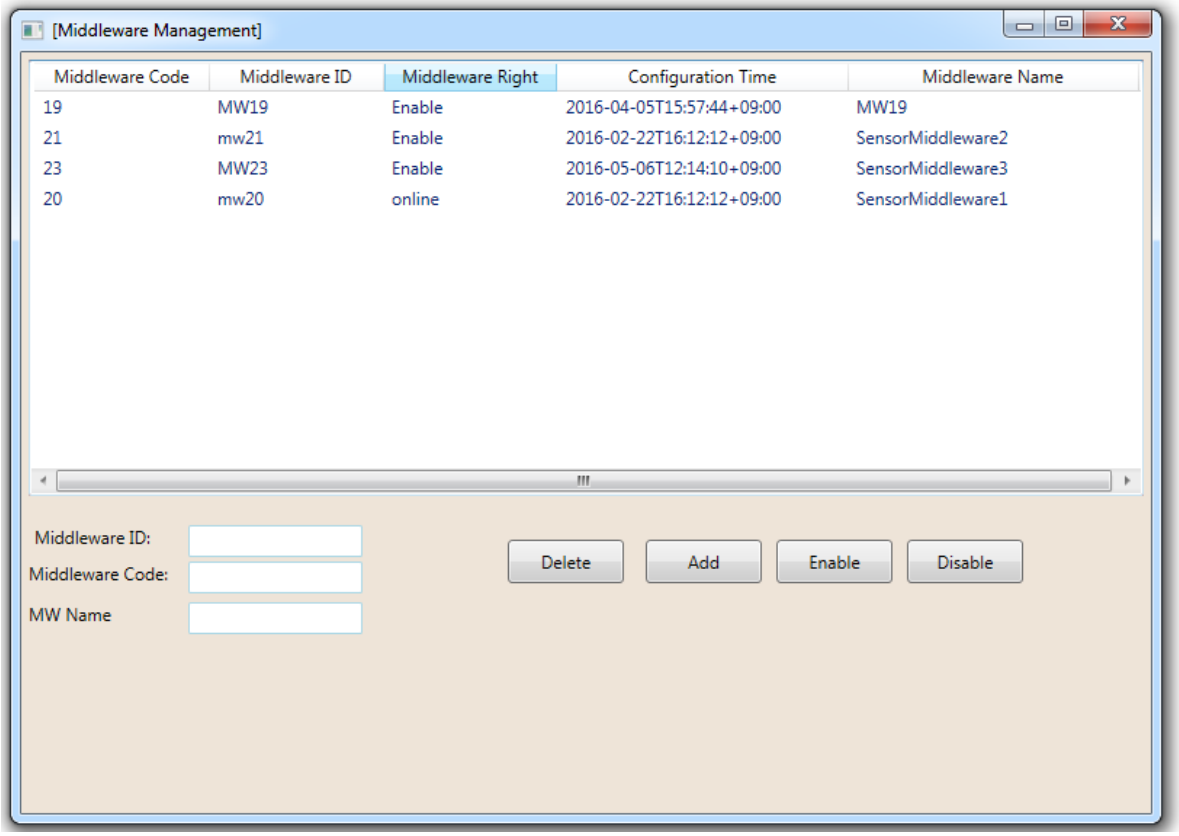

Figure 43 Semantic Middleware Management

Figure 43 shows the sensor middleware management window. From here the user can manipulate the middleware context data stored in the ontology. This is performed with the help of SPARQL queries run through the semantic sensor content service. New middleware data can be added and existing can be deleted or updated. The list in the window displays all the sensor middleware context data available in the ontology. For each middleware it shows the middleware code, middleware id, access right, configuration time, and the middleware object. The enable and disable button controls are used to enable the access rights of a specific middleware. When user clicks the add button the data is added to the ontology through SPARQL insert queries. With each new middleware the value for middleware right is disable. The value for the middleware access right can be updated using enable and disable buttons.

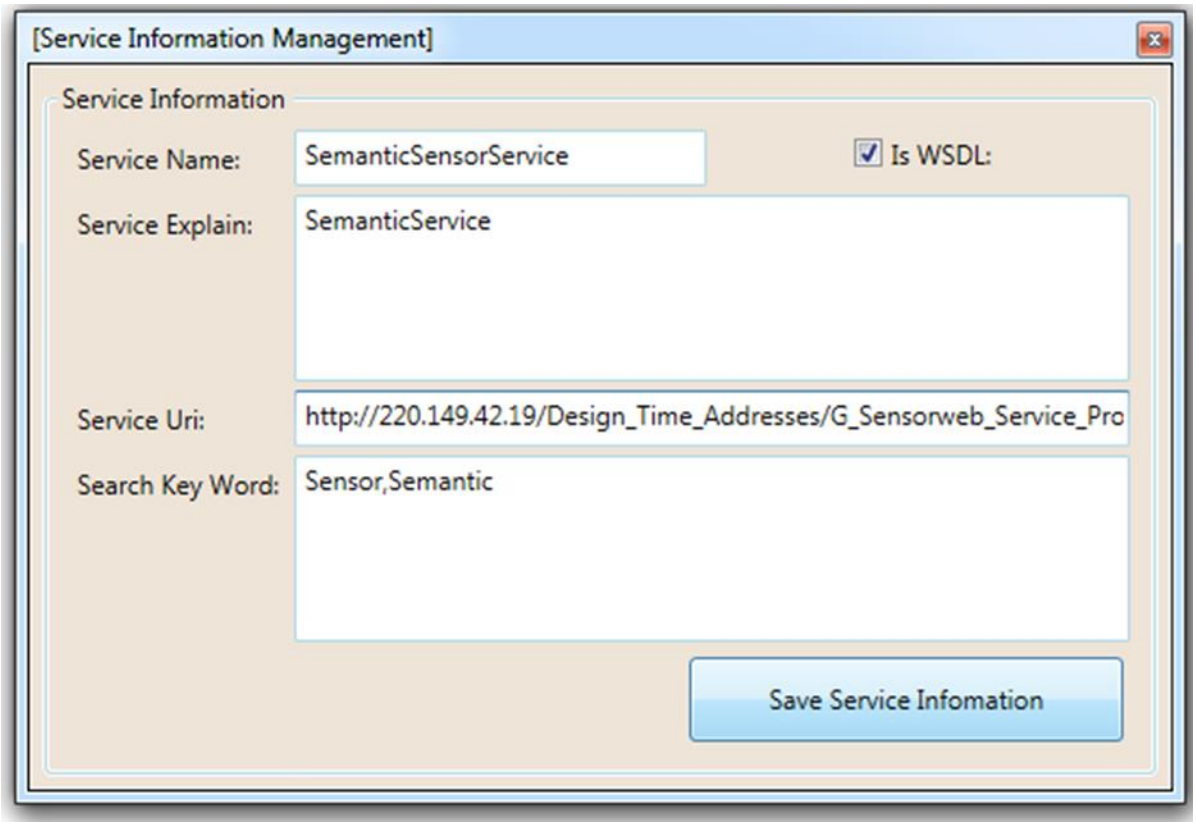

Figure 44 Service information management execution screen

Figure 44 shows the service information management screen at runtime. This interface is used for creating service information and to register it to semantic service registry. Service information includes service name, offers WSDL or not, service explanation, service access address and service searching key word. Service Name is area entering service name. Is WSDL provides option to select whether the provider's service offers WSDL (Web Services Description Language) or not. Service Explain is explanation of service. Service Uri is service provider's access address. Search Keyword assigns keyword that is used for searching service provider. Save Service Information is a button that registers service provider's information to service registry ontology.

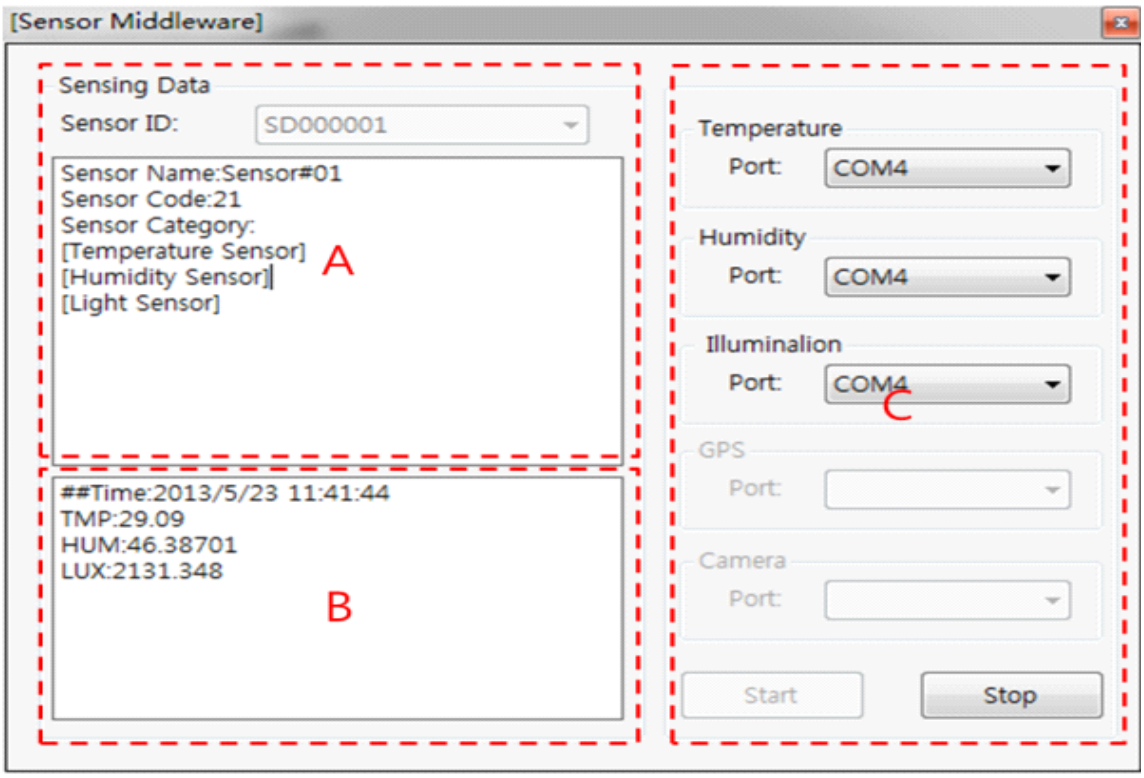

Figure 45 Sensor middleware execution screen

Figure 45 shows the runtime screen for sensor middleware. Sensor middleware connects to sensor and collects sensing data. Area A of the screen shows the information about connected sensor. Sensor ID provides a list of selectable sensor ID from the provider ontology. Upon selecting a sensor, the sensor information (sensor name, code, sensing type) is displayed in area A. Area B shows real time sensing data. Sensing data includes sensing time and the sensor's measured values. C provides list of ports for selecting sensor connecting port. Figure 46 displays the sensor network module this study has used to collect sensing data. The image shown on the right side is the TIP700SM which is the Wireless Sensor Network Module using MSP430F1611 MicroController Unit of TI and CC2420 of ChipCon.

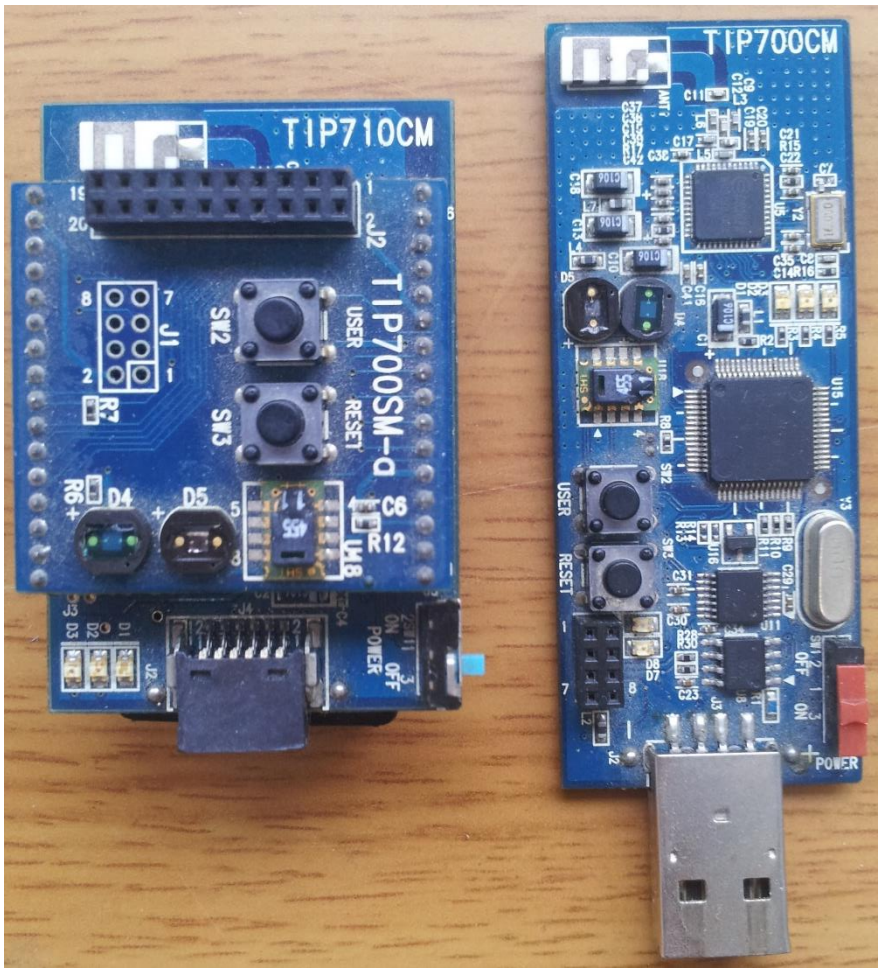

Figure 46 Sensor Network Module

The image on the right shows the sensor board used with the sensor module. The sensor board consists of humidity and temperature sensor: SHT11 (by sensirion), Par (photosynthetically active radiation- photodiode for visible range: S1087 (by Hamamatsu). In this study we have collected and used temperature, humidity and illumination data.

5.1.2 Development and Reasoning of the Sensor Ontology

The sensor ontology is modeled for the semantic sensor service provider module. The ontology is developed using protégé. Protégé is a free, open source ontology editor and a knowledge management system. Protégé provides a graphic user interface to define ontologies. It also includes deductive classifiers to validate that models are consistent and to infer new information based on the analysis of an ontology. As mentioned before in this system we have reused the SSN ontology

for basic definitions of a sensor and its observations. Figure 47 represents a screen shot taken from protégé. It displays the classes and the instances in our ontology. The classes in protégé are all subclasses of the root class i.e. Thing.

In this ontology we have focused on the sensor board TIP700SM from the TIPxxx series, as we have used it to collect the sensor data for our semantic IoT system. The TIP700SM have temperature/humidity sensor, PAR (photosynthetically Active Radiation) sensor and a TSR sensor (Total Solar Radiation). The humidity/temperature sensors are from the SHT1x sensirion's family. And we have used the PAR sensor for illumination data collection. The PAR sensor belongs to s1087 series by Hamatsu. The classes that are reused from the SSN ontology are; Sensor which is defined as an entity that can implement sensing and observe some property, Sensor Output class represents a piece of information that is produced by some sensor.

In our ontology we have define three Sensor Outputs i.e. Humidity Sensor Output, Temperature Sensor Output, and Illumination Sensor Output. And we have reused the ssn:isProducedBy object property to connect Sensor Output Class to the Sensor (TIP700SM). Each Sensor Output has some value represented by SSN Observation Value class, which is defined as the value of the result of an Observation. We have defined 3 values to be observed by the sensor, humidity value, temperature value, and illumination value. Observation class is defined as a Situation in which a Sensing method has been used to estimate or calculate a value of a Property. We have defined three subclasses of Observation class. Humidity Observation which is connected to the Humidity Sensor Output class by the observation result property, and uses the observed by property to connect to the sensor class.

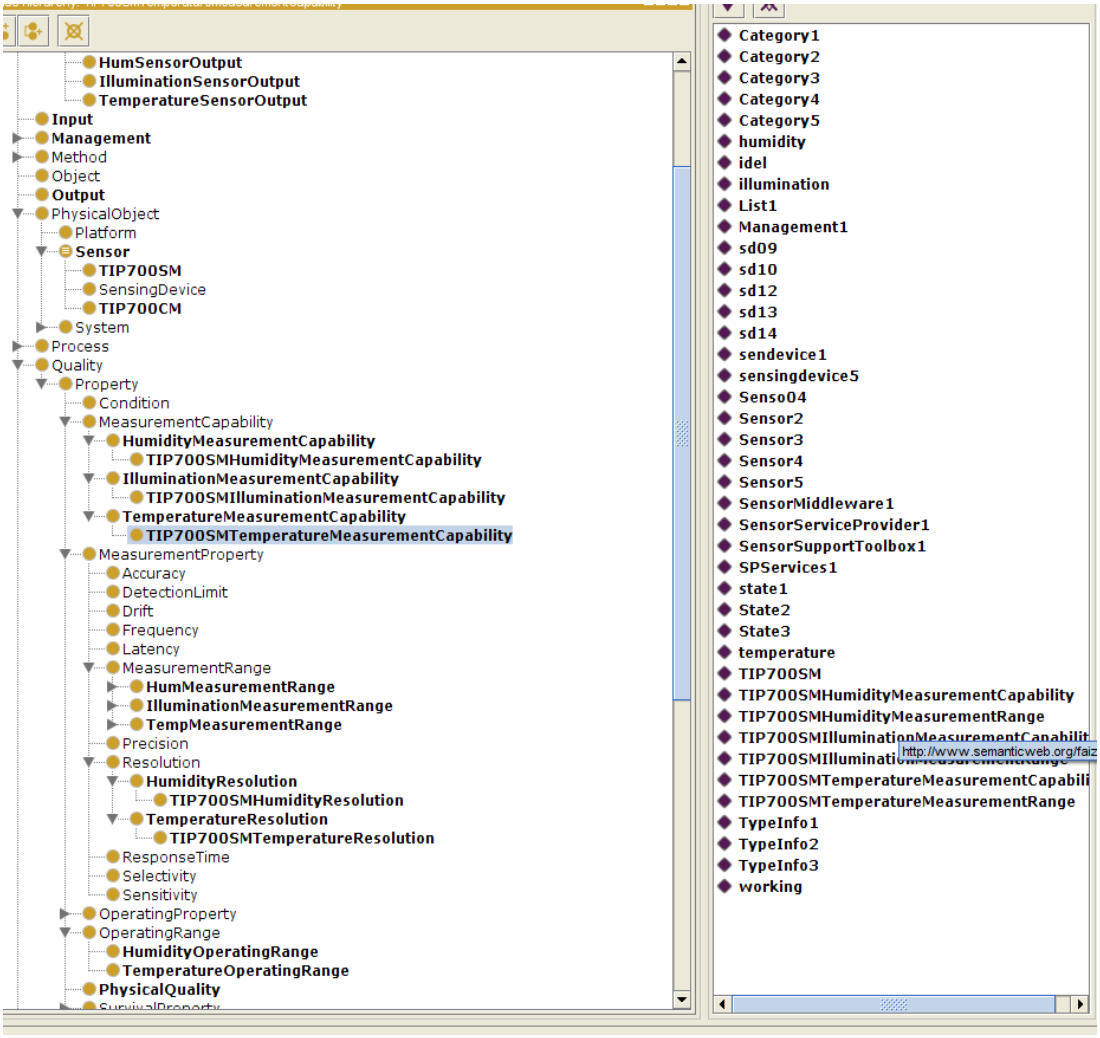

Figure 47 SSN and SSP ontology

The next class reused from the SSN ontology is the Measurement Capability class, which collects together measurement properties and environmental conditions in which those properties hold. It represents a specification of a sensor's capability in those conditions. We specified that the class TIP700SM must have 3 properties for ssnhasMeasurementCapability belonging to the classes Humidity Measurement Capability, Temperature Measurement Capability, and Illumination Measurement Capability. These classes define possible configurations of measurement capabilities for TIP700SM sensor board: Resolution value for humidity is (min 0.4%, max 0.05% relative humidity), and for temperature it is (min 0.04, max 0.04), we have also specified operating range

for these sensors by associating them with the Operating Property class. The operating ranges for humidity are (0-100%), for temperature (-40-123.8 C) and for illumination its (320-1100).

Reasoning in ontologies and knowledge bases is one of the reasons why a specification needs to be a formal one. By reasoning we mean deriving facts that are not expressed in ontology or in knowledge base explicitly. A reasoner performs the following validation checks on an ontology.

- Satisfiability of a concept - determine whether a description of the concept is not contradictory, i.e., whether an individual can exist that would be instance of the concept.
- Subsuming of concepts - determine whether concept C subsumes concept D, i.e., whether description of C is more general than the description of D.
- Check an individual - check whether the individual is an instance of a concept
- Retrieval of individuals - find all individuals that are instances of a concept
- Realization of an individual - find all concepts which the individual belongs to, especially the most specific ones.

Figure 48 shows a screen shot from protégé displaying the reasoning performed on sensor ontology. In this study we have used the Pellet reasoner. Pellet is a complete OWL DL reasoner and has extensive support for reasoning with individuals, user-defined types, and debugging ontologies [32]. The area A shows the individuals added in the sensor ontology, whereas the area B shows the inferred facts about a particular individual. In the figure the individual Category 3 is highlighted in the red outlined area which means the inferencing results shown are based on the facts asserted for this specific individual. E.g. the reasoner infers the isCategoryOf property for the individual Category 3. This property lists all the sensors that are associated with this specific instance of a category. These inferred facts can be queried using SPARQL queries with some special operators which can help users in resource discovery.

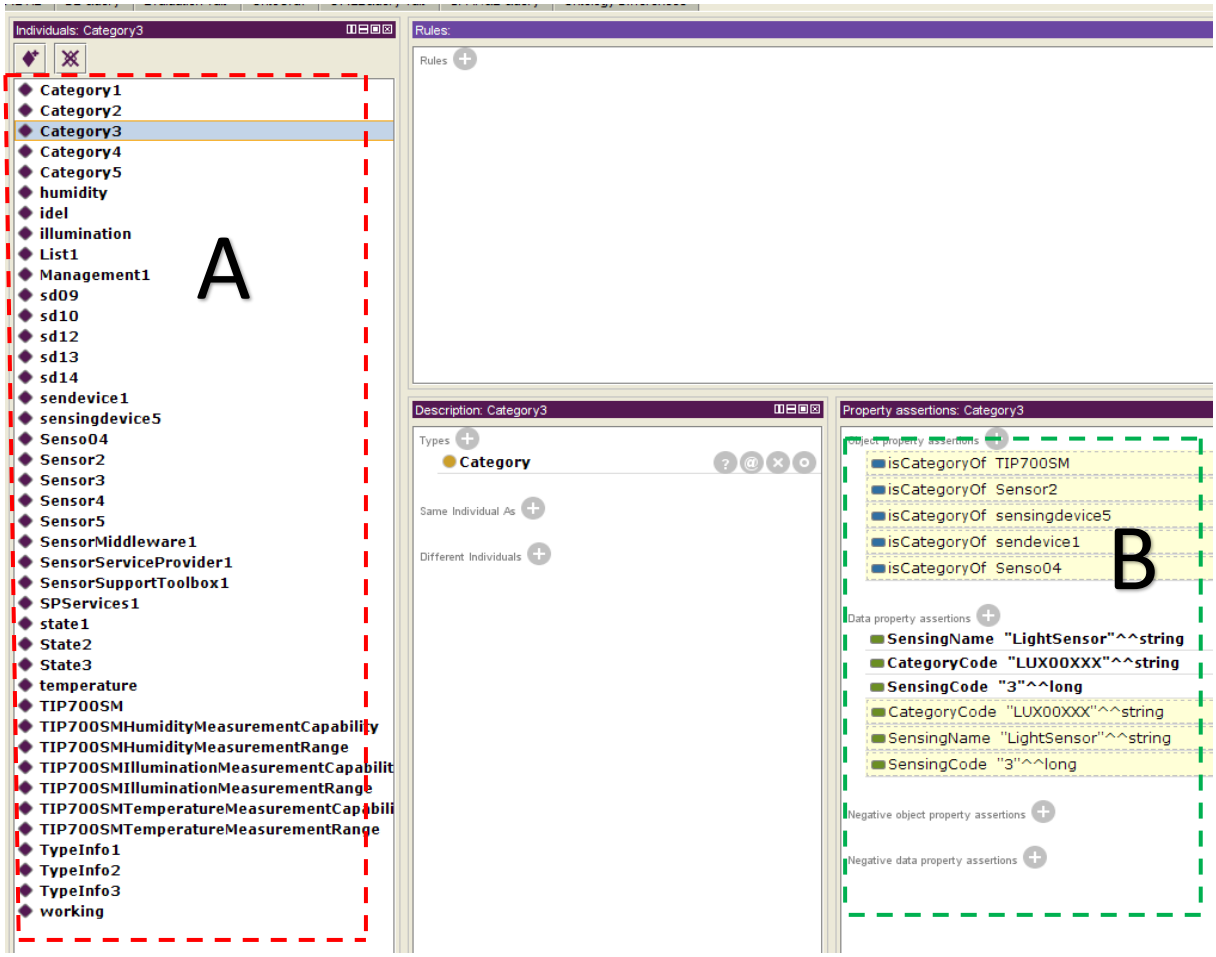

Figure 48 Reasoning

5.1.3 Performance Analysis of Semantic Sensor Service Provider

The experiments analyze the efficiency of performing SPARQL queries versus SQL queries. Both of these languages give the user access to create, combine, and consume structured data. SQL does this by accessing tables in relational databases, and SPARQL does this by accessing a web of Linked Data. The graph shown in Figure 49 illustrates the comparison of SPARQL and SQL queries for adding new sensor information to the sensor service provider repository.

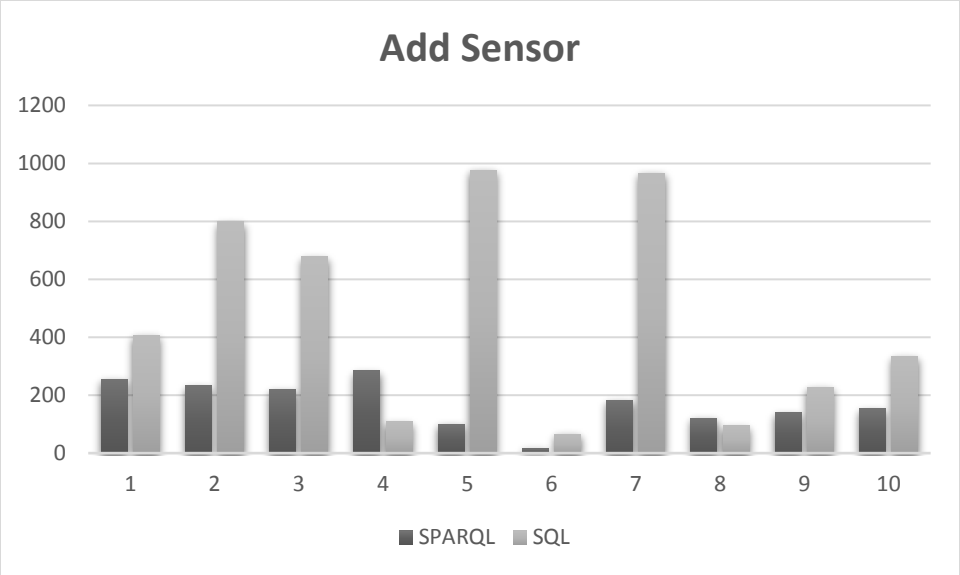

Figure 49 Query comparison for adding a sensor

10 iterations are taken at random resource utilization level of the host system. The difference between the two queries have been recorded in terms of min, max and average time in milliseconds for the 10 iterations. For the SPARQL query the min time in milliseconds taken for adding an iteration is 16ms, max time taken is 874ms and average time taken is 227.65. Whereas the min time taken for an SQL query to add an iteration is 65ms, the max time taken is 976ms and the average time taken is 555.45. The graph shown in Figure 50 describes the comparison results for SPARQL and SQL update queries. The update query takes the data updated from the user interface and updates the entry in the service provider repository. The graph shows the time taken in milliseconds to update 10 iterations each for SPARQL and SQL. The comparison of the two queries is based on the min, max and average time taken in milliseconds to update an iteration.

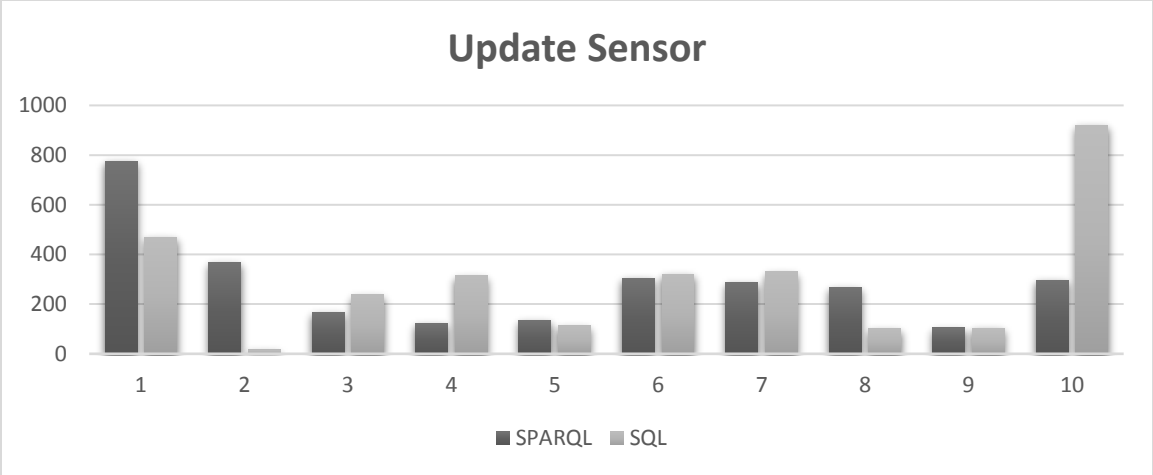

Figure 50 Query comparison for update sensor

The min time taken in milliseconds by the SPARQL query to update an iteration is 104ms, the max time taken is 776ms and the average time taken is 227.88. Whereas the min time taken in milliseconds to update an iteration using SQL query is 16ms, the max time taken is 920ms and the average time taken is 272.66ms. Figure 51 shows the graph displaying the results of SPARQL and SQL queries for deleting a sensor from the sensor service provider repository.

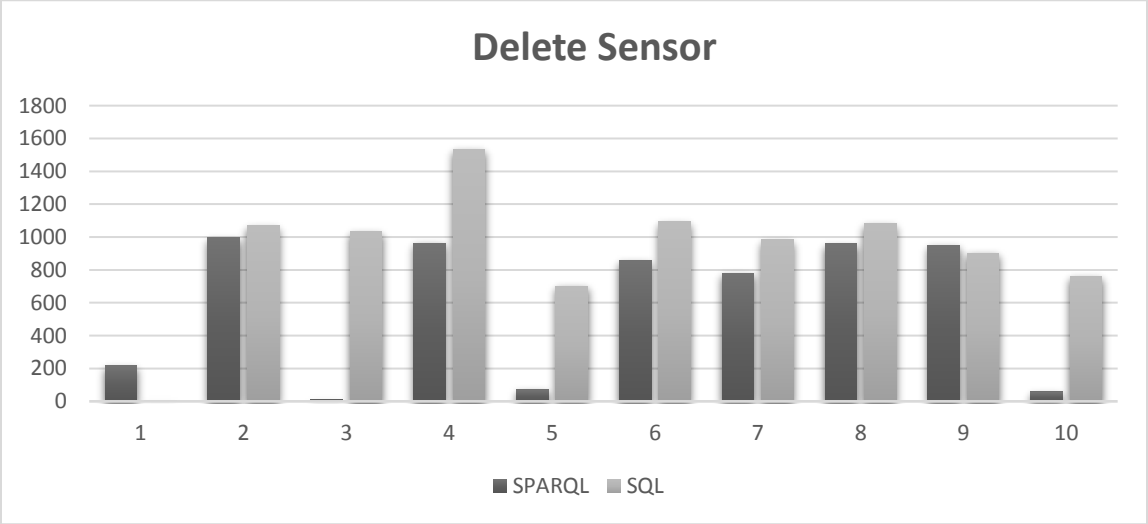

Figure 51 Query comparison for delete sensor

The queries are compared based on the min, max and average time in milliseconds taken to delete an iteration. 10 iterations have been taken for each query. The min time taken in milliseconds by a delete SPARQL query is 9ms, the max time taken is 997, and the average time taken is 585.5. Whereas the min time taken by a SQL delete query is 4ms, the max time taken is 1534ms and the average time taken is 915.6ms.

5.2 Semantic Actuator Service Provider

5.2.1 Implementation of Semantic Actuator Service Provider

Semantic actuator service provides actuator related information, control and management. The execution screen for the actuator service provider is shown in the Figure 52.

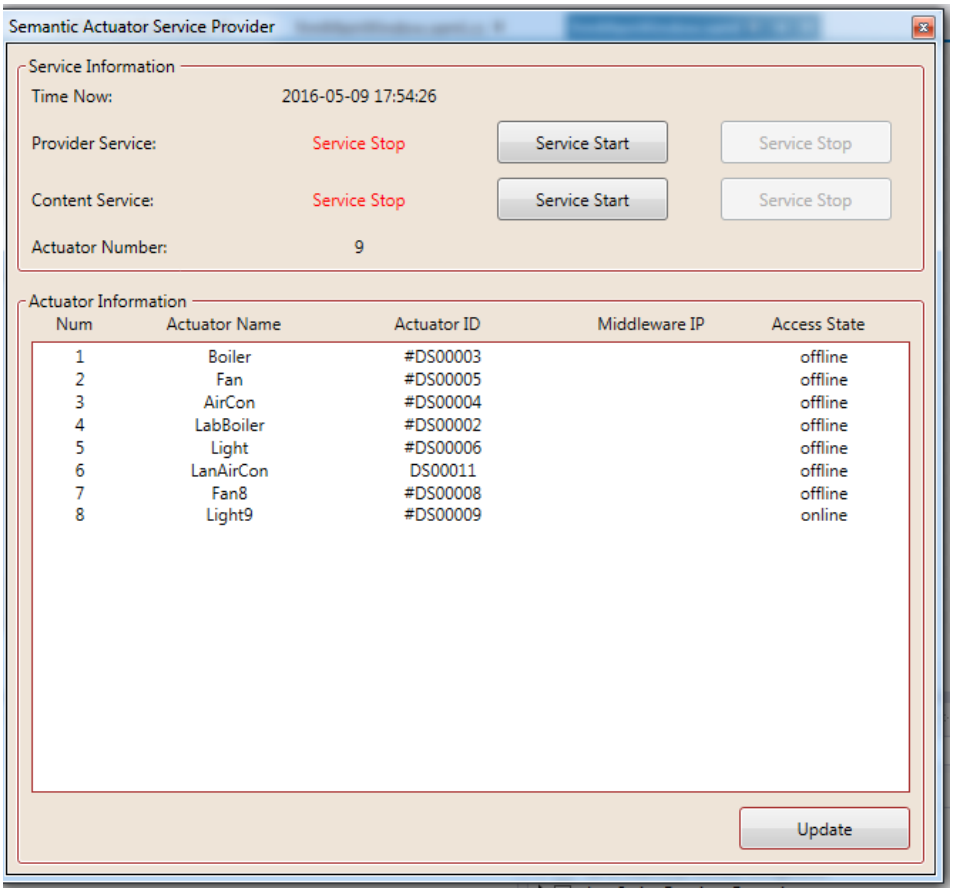

Figure 52 Semantic Actuator Service Provider Manager

It consists of two panels; the service information and the actuator information. The service information panel contains controls for starting and ending the content service and provider service. Time now shows the current date and time. Actuator information shows the actuator name, its id, and its access state. Middleware IP is the IP address to which the object is associated.

Each module consists of a toolbox for management of the data maintained by that module's ontology. Figure 53 shows the execution screen for the semantic actuator support toolbox control panel. It provides the following functionality: actuator information generation and management, actuator model information generation and management, middleware information generation and management, and service information generation and management. Controls are available on the execution screen to carry out each of the above functionality.

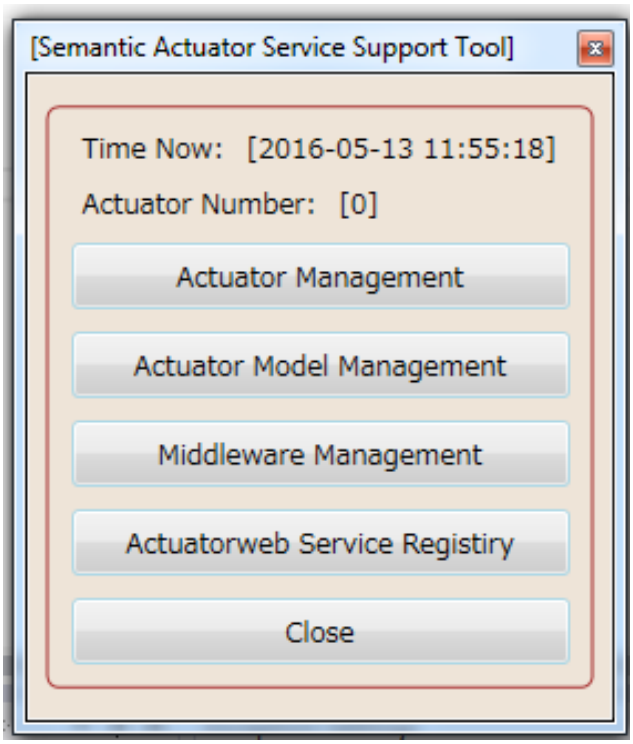

Figure 53 Semantic Actuator Service Support Toolbox

The Figure 54 shows the execution screen for the semantic actuator information management. It displays an Actuator list that consists of the ids of all the actuators registered in the ontology. The execution screen also displays a form for generating and manipulating actuator information.

The information that is required to add a new actuator includes 1) Actuator ID; a unique id that uniquely identifies each actuator in the ontology, 2) Actuator Name; a string name assigned to each actuator, 3) Actuator Explain; description of an actuator, 4) Power Consumption; corresponds to the energy requirement of an actuator, 5) Access state; describes the status of an actuator it takes two values i.e. online and offline, 6) Individual Name; defines an instance for each actuator added to the ontology, 7) Connects to; a combo box that shows the registered actuator in the ontology,

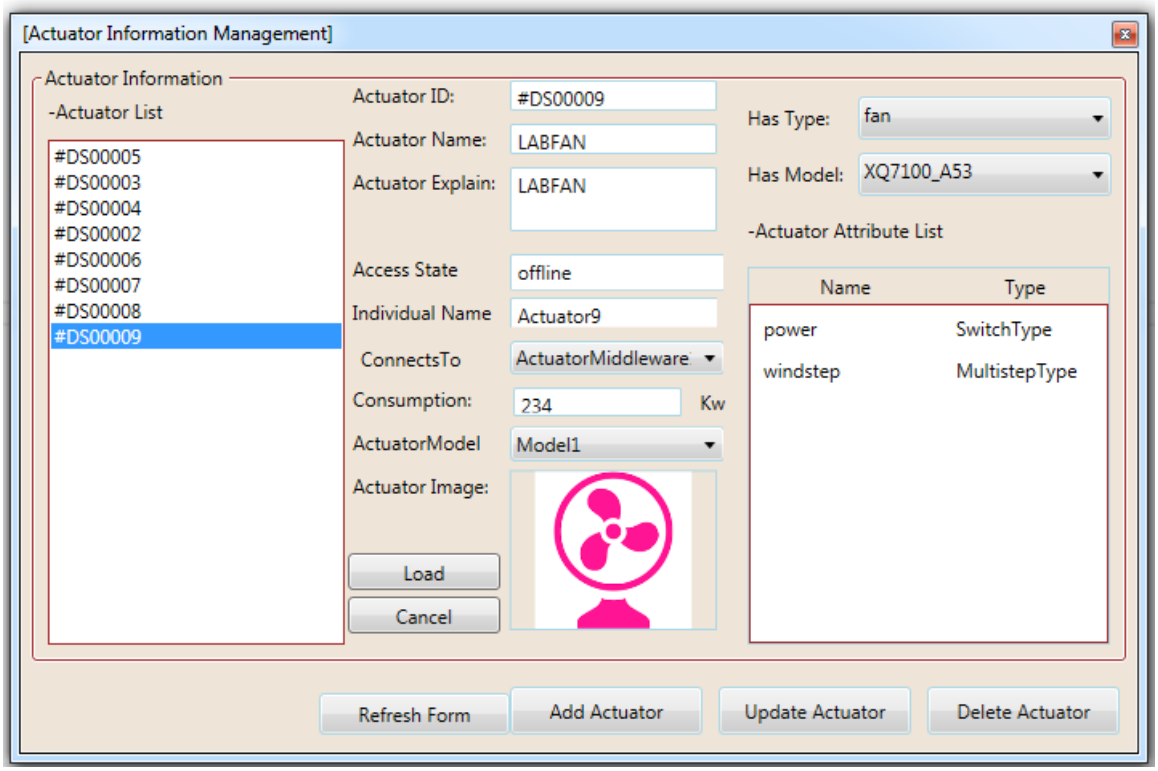

Figure 54 Semantic Actuator Information Management

8) Has type; a combo box displays the list of actuator types registered in the ontology, and 9) Has model; a combo box that shows the actuator models registered in the ontology.

Actuator Attribute list shows the attributes related to a particular model of an actuator e.g. the actuator model shown in the figure is fan it has two attributes associated with it, the first one is the power attribute that represents the off and on switch type, whereas the second one is the wind step

attribute that represents the wind steps available in the fan. The execution screen shows a list for the actuator attributes.

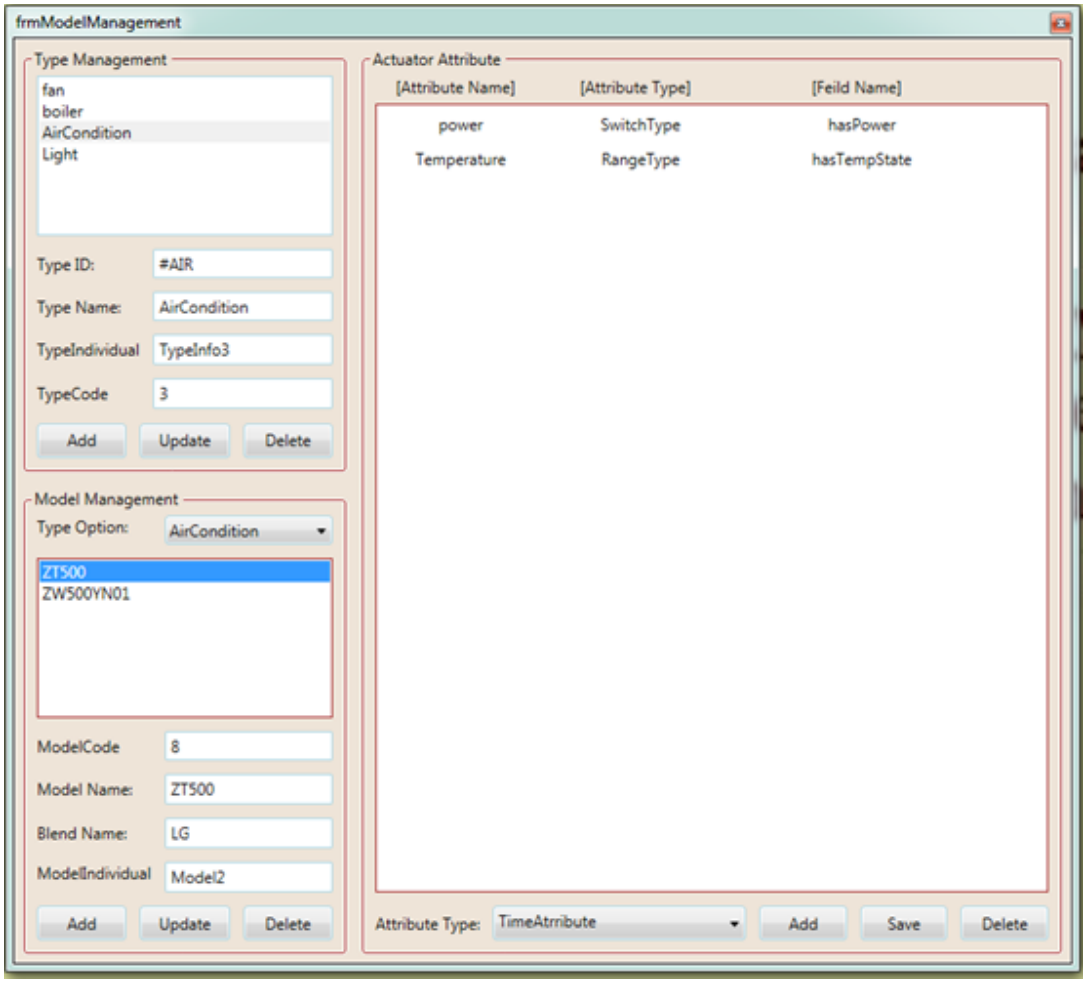

Figure 55 semantic actuator model management

The attributes of an actuator in the list are displayed based on the actuator model and actuator type values selected from the combo boxes. The controls displayed on the screen are Refresh Form: it reloads the list of actuator ids registered in the ontology, Add Actuator: it adds the information from the textboxes into actuator ontology, Update Actuator: it is used to update existing information in the ontology, and Delete Actuator: it is used to delete a registered actuator from the ontology. For each actuator registered in the ontology, a model and a type is associated with it based on its attributes.

Figure 55 shows the execution screen for manipulating actuator model and type information. It consists of Actuator type management form which shows: a list that displays the type information of actuators registered in the ontology, it displays textboxes for generating new type for an actuator. The information needed to add a new actuator type is: 1) Type ID: a unique identification for each type, 2) Type Name: name of the type, 3) Type Individual: an instance of the type class and, 4) Type Code: a code for each type. It consists of controls for add, update and delete functions.

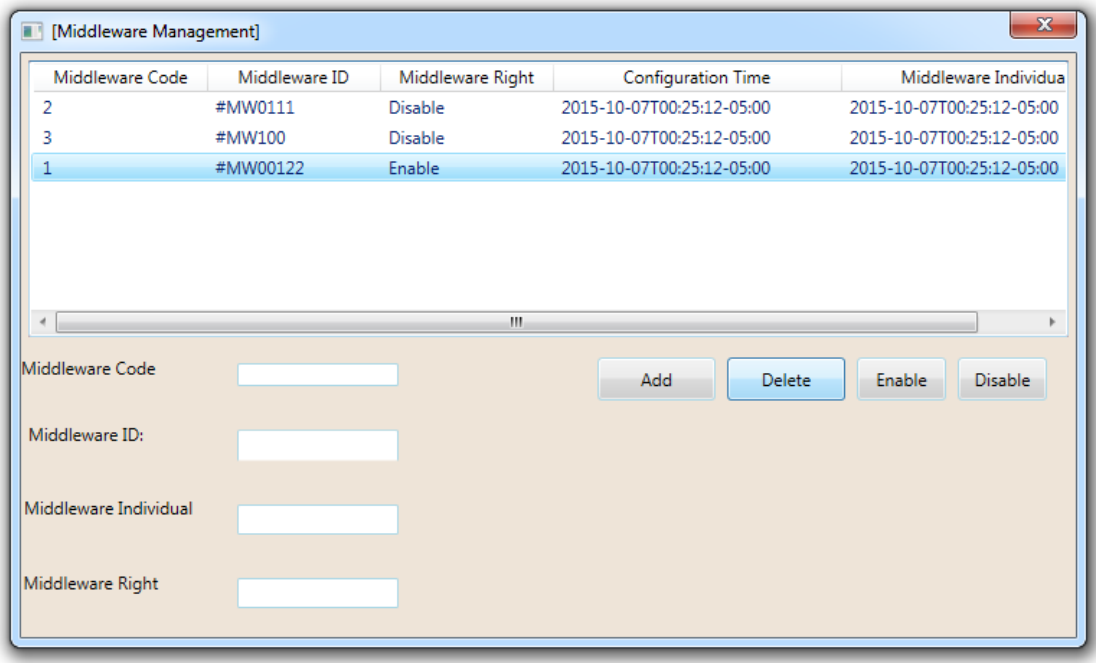

Figure 56 semantic actuator middleware management

The figure also displays an actuator model management form which consists of a list and a combo box named type option. On selection of an actuator type it displays the model associated with the type in the list. It consists of textboxes to add new information for actuator model. The information needed to add a new model is: 1) Model code: define a code for each model, 2) Model Name: defines the name of a model, 3) Model Individual: defines the instance for the model and, 4) Blend Name: defines the name of the brand for the model. It consists of controls to manipulate actuator model data in the ontology. Actuator attribute list consists of: 1) Attribute Name: displays the name of the attribute associated with the selected model, 2) Attribute Type: displays the type

of the attribute and, 3) Field Name: displays a unique key for each attribute. Controls are provided for the user to add, save, and delete attributes.

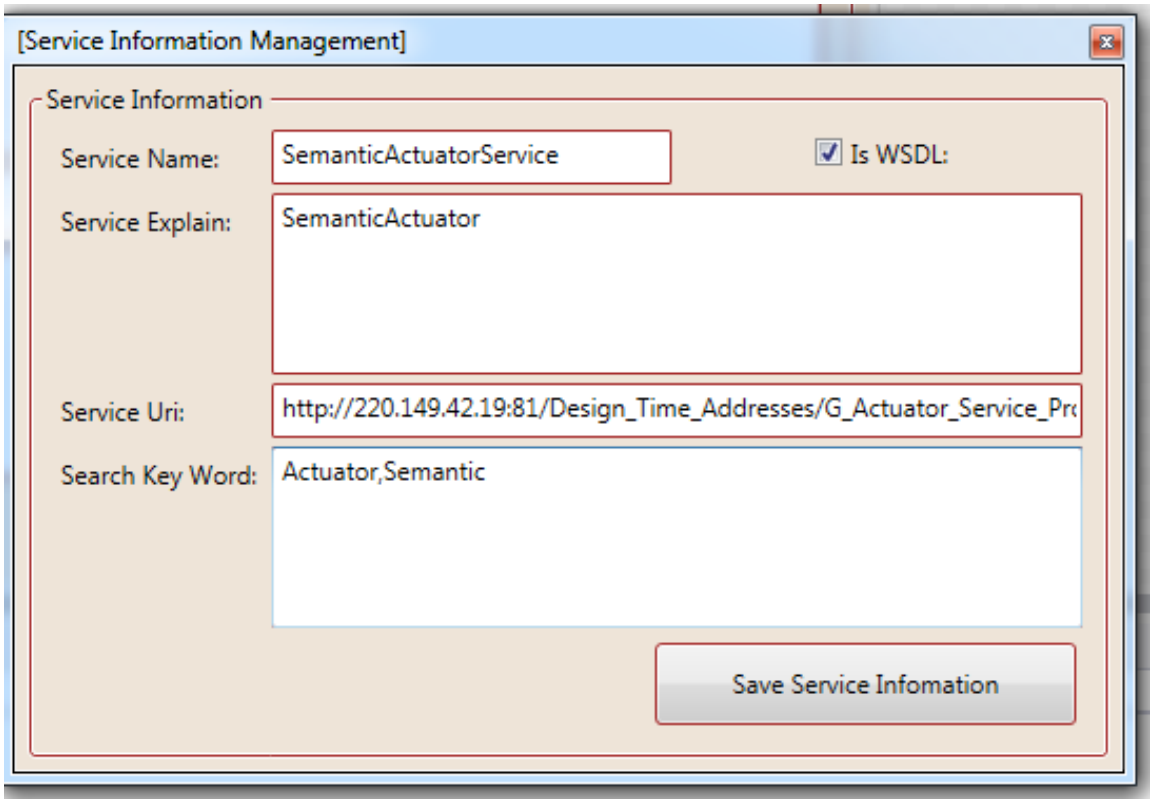

Figure 57 semantic service information management

Figure 56 shows the middleware management execution screen. It consists of a list to display the middleware's that are registered in the ontology. The list contains information: 1) Middleware Code: a code value for each middleware. It takes a long value. 2) Middleware ID: a unique identity. It takes a string value that can consists of letters and numbers. 3) Middleware access right: value to define if the service provider can access the middleware or not. It takes two string values i.e. enable and disable. 4) Configuration Time: defines the time the middleware is configured and added to the ontology and, 5) Middleware Individual: defines the instance for each middleware information.

The execution screen also displays text boxes for adding new middleware information in the ontology. It also consists of controls to manipulate the middleware information in the ontology. The screen shown in Figure 57 is used to create service information and register it at service registry

ontology. Service information include service name, WSDL offering or not, service explanation, service access address and service search keyword.

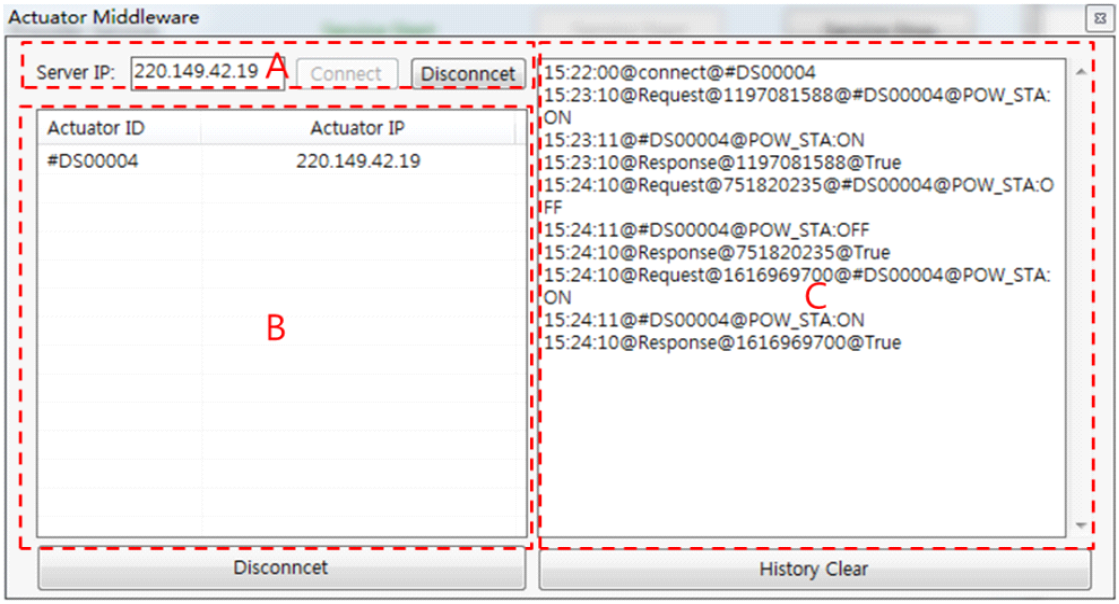

Figure 58 Actuator middleware execution screen

Service Name is area for entering service name, Is WSDL is option for selecting the service provider as a WSDL (Web Services Description Language) offering service or otherwise, Service Explain is explanation about service, Service Uri is service provider access address and, Search Keyword assigns a keyword that is used for searching service provider. Save Service Information is a button which registers service provider information to service registry. Figure 58 shows actuator middleware execution screen. Area A is for setting Actuator service provider's IP address. Server IP represents the IP address of the actuator service provider to which this middleware is connected.

Area B shows the connection state information of the actuators connected to the middleware. The connection state information consists of actuator id and actuator IP. Actuator ID is actuator's unique identifier whereas Actuator IP is the current actuator's IP address. Area C shows messages exchanged between the actuator service provider and the actuator.

No	Type Code	Message Type	Object Id	Data		Description
1	00	Request	Actuator/Middleware ID	Key & connect		Connection request msg
2	00	Respond	Actuator/Middleware ID	Key & connect	True/ false	Connection acknowledgement
3	00	Request	Actuator/Middleware ID	Key & disconnect		Disconnect request msg
4	00	Respond	Actuator/Middleware ID	Key & disconnect	True/ false	Disconnect acknowledgement
5	01		Middleware ID	Actuator ID list		Service provider mapping table message
6	10	Request	Actuator ID	Key & command		Actuator control request message
7	10	Respond	Actuator ID	key	True / false	Actuator control respond message
8	11		Actuator ID	State list		Actuator work state message

Figure 59 Actuator message format

In this thesis, semantic actuator service provider provides service for remotely controlling the indoor Appliances. Based on the comfort index, the application client generates command through semantic actuator service provider. The actuator middleware executes the command on the actuator and the client receives the response message back through the actuator web provider. This study uses TCP Socket system and to communicate between actuator and semantic actuator service provider, a message format is needed for sending and receiving messages as shown in Figure 59.

Actuator message is composed of 4 parts namely Type Code, Message Type, object ID and Data. Using 4 part message satisfies the requirements of Actuator system. This is different from connection message, mapping information exchange message, control message and state information message. These Control message and connection message commands does not use simple push methods.

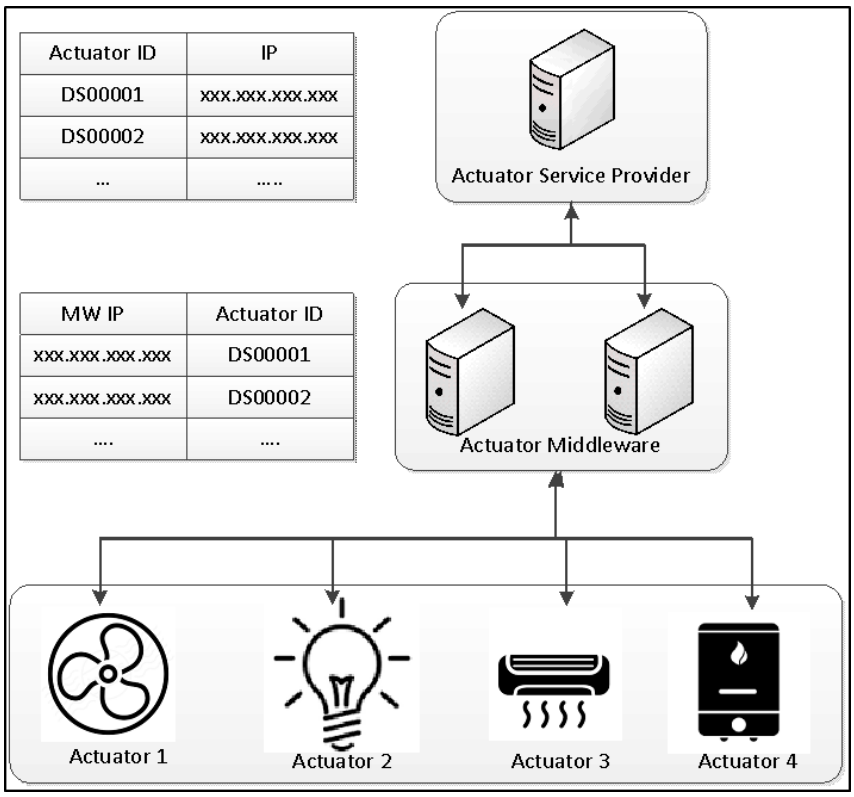

Figure 60 Mapping table management

But like HTTP, there must be acknowledgement for each request message. Using a filter called Type, the messages are divided into Request and Respond message. Object ID is unique identifier from the perspective of client. Using this system it is easy to know which component sent the message when client parses the message. Data has the actual contents of the message. When Type Code is “00” it indicates a connection request message, Type code “01” indicates mapping information exchange message, Type Code “10” specifies control message, and Type Code “11” indicated state information message. This study implements the Actuator system remote control message sending functionality with TCP Sockets.

But between service provider and middleware, middleware and actuator, one to many communication system is required. Thus from the perspective of service provider a mechanism is needed which meets the requirement as shown in Figure 60. This is very similar to network routing concepts. First the actuator middleware receives a connection request message from each actuator.

It creates mapping table from its memory, consisting of actuator id and IP information, and sends this information to the actuator service provider. The service provider has its own mapping table consisting of actuator id and middleware IP, it compares mapping table information from the middleware with its own mapping table and updates it accordingly.

5.2.2 Development and Reasoning of Actuator Ontology

In this section the development of the actuator ontology is discussed. This ontology is developed in protégé using Rdf/xml format. Figure 61 shows the screen shot of the ontology from protégé. It shows the classes and the individuals tab. The classes tab shows all the classes added to the ontology. The classes included in the actuator ontology are: Actuating Device class defines any actuating device that is registered to the ontology, Actuator Middleware class defines the middleware's that are in the proposed system, Actuator State class defines the state of each actuating device added in the ontology. We have defined two states for actuator online for actuator that is switched on and offline state for actuator that is switch off. Actuator Support Toolbox class represents the toolboxes in the proposed service. It is important to know which actuator, middleware and toolbox are connected, for maintaining the mapping table in the actuator service provider module.

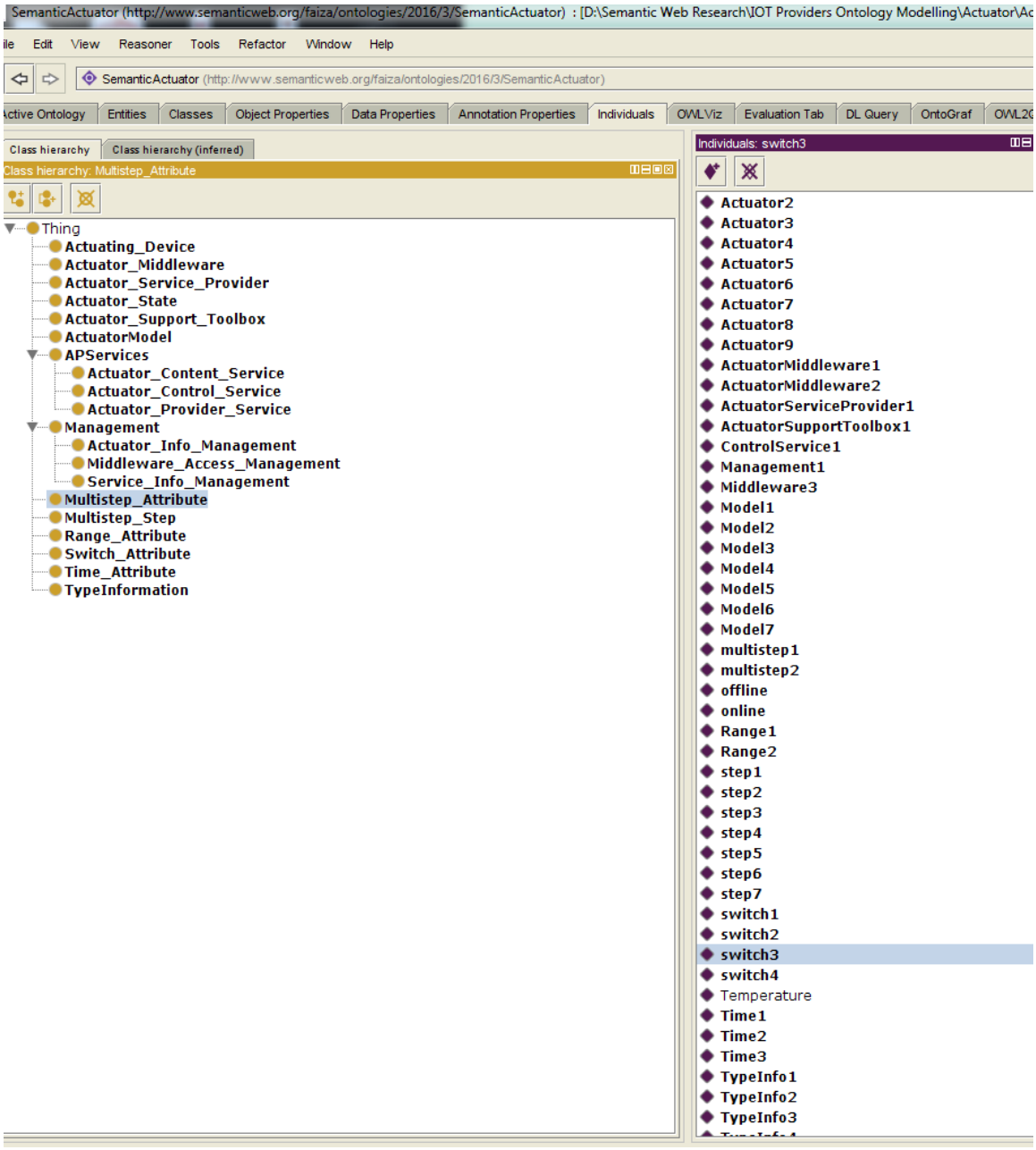

Figure 61 Actuator ontology

APServices class represents the services provided by the actuator provider. Each service is defined by creating a subclass of the APServices class. In the proposed system each actuator is defined to have multiple models based on which its functionality is determined. The models are represented by the Actuator Model class. As mentioned earlier the functionality of the support toolbox module is to perform management of the actuator and middleware information, this is

represented using the Management class. This class has three subclasses Actuator Information Management, Middleware Access Management, and Service Information Management, these classes represents the management performed by the actuator support toolbox.

The attributes to be defined for each new actuator in the system are the multistep function of the actuator, the range of the actuator, the switch function of the actuator, the time at which the actuator is manufactured and the type of the actuator. All these attributes are represented by Multistep Attribute class, Range Attribute class, Switch Attribute class, Time Attribute class, and Type Information class. For each class individuals or instances are added, individual instances are the most specific concepts represented in a knowledge base. In this ontology we have defined individual for each class. Results of reasoning performed on the ontology is shown in Figure 62 Area A shows the individuals added in the ontology whereas highlighted facts shown in Area B are the inferred facts for the individual Model 1. As shown the actuator model individual infers some new relations based on the object properties it is given. This helps in querying, if user wants to know how many actuator or which actuators are related to this specific instance of Actuator Model class, special operators with SPARQL query can be used.

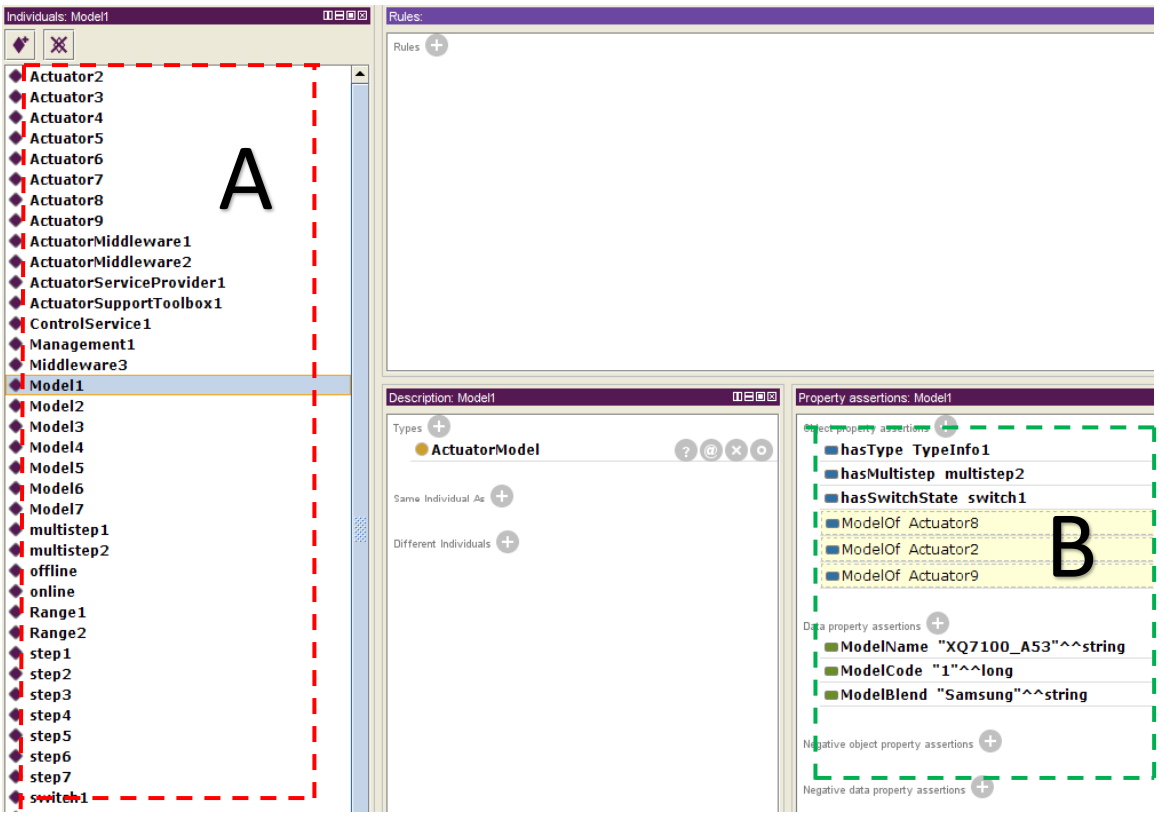

Figure 62 Reasoning results on actuator ontology

5.2.3 Performance Analysis of Semantic Actuator Service Provider

This section provides the results of SPARQL and SQL query comparison performed for analyzing the semantic actuator service provider system performance. All the comparisons are based on the min, max and average time taken in milliseconds for 10 iterations of each query.

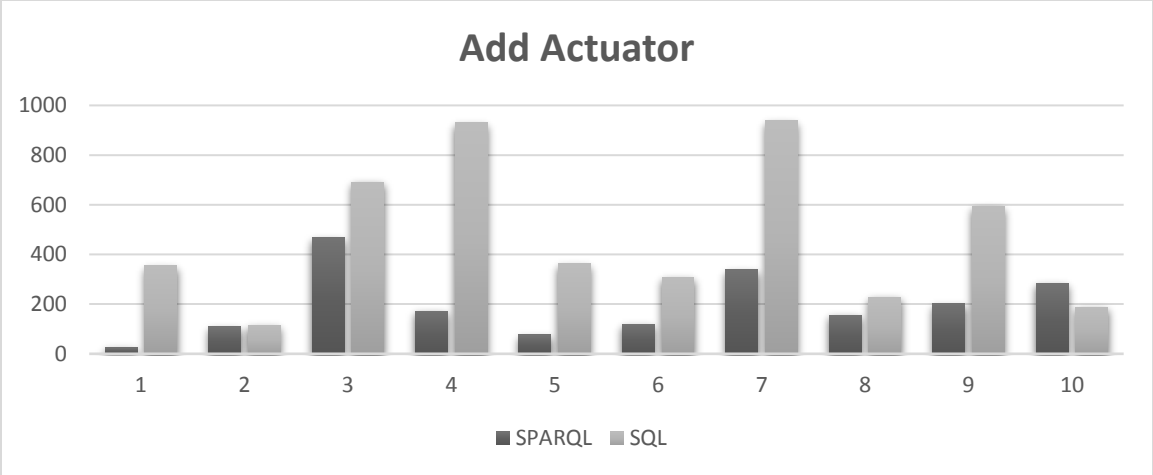

Figure 63 Query comparison for add actuator

Figure 63 displays a graph that compares the query results for adding an actuator information in the semantic actuator service provider repository. 10 iterations have been taken for each query based on random resource utilization of the host machine. The min, max and average time taken in milliseconds by these queries is compared. The min time taken by a SPARQL add query is 26ms, the max time taken is 467ms and the average time taken is 195.2. The min, max and average time taken for SQL add is 115ms, 939ms and 470.7ms respectively.

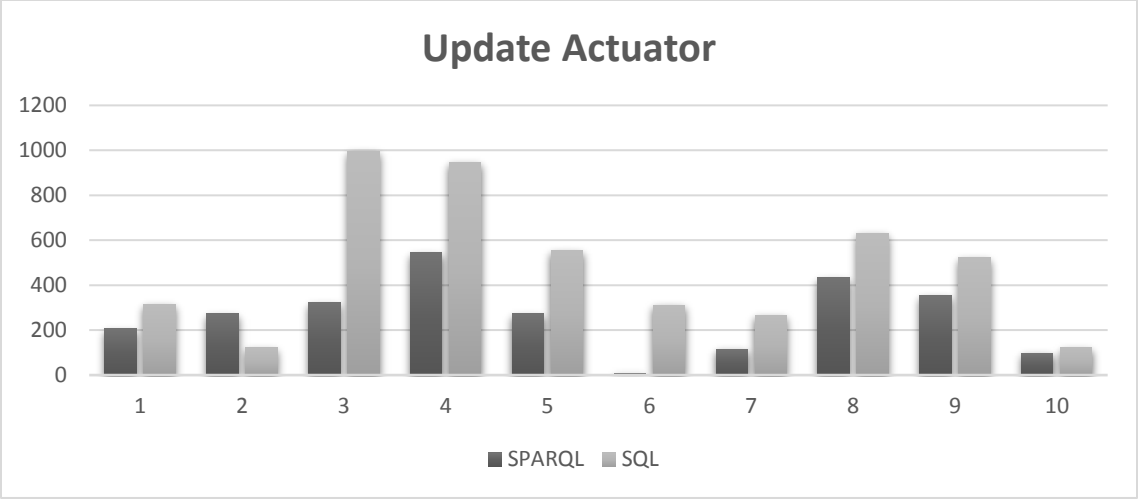

Figure 64 Query comparison for update actuator

Figure 64 shows the comparison results for SPARQL and SQL update queries. The comparison is based on the min, max and average time taken by these queries to perform the update to the semantic actuator service provider repository.

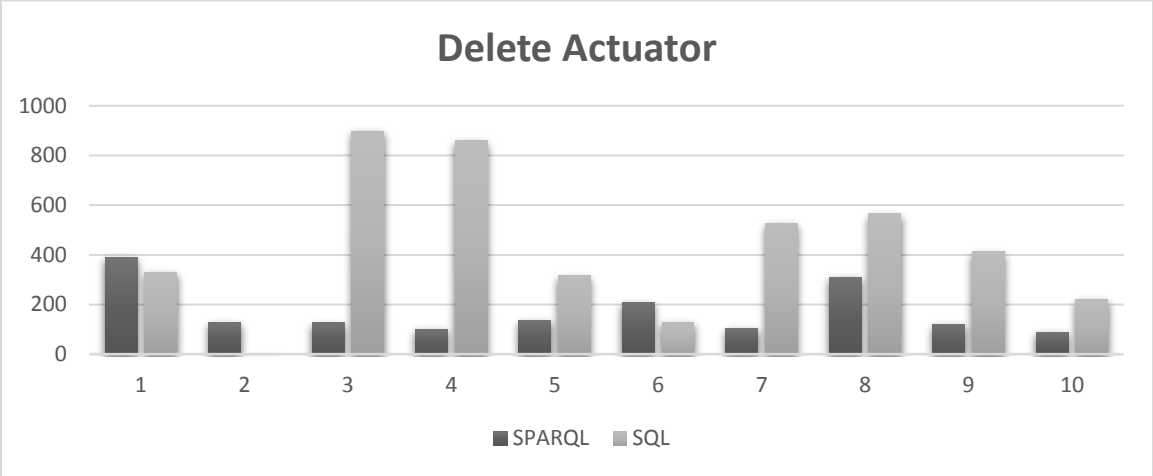

Figure 65 Query comparison for delete actuator

The min time taken for SPARQL update query is 9ms, the max time taken is 544ms, and the average time taken is 262.3ms. Whereas the min time taken by SQL update query is 122ms, the max time taken is 995ms and the average time taken is 478.7ms.

The graph shown in Figure 65 compares the SPARQL and SQL query results for deleting an actuator from the semantic actuator service provider repository. 10 iterations are taken for each query and the comparison for each iteration is based on the min, max and average time. The min time taken by a SPARQL delete query to delete an actuator information is 88ms, the max time taken is 390ms and the average time taken is 262.3ms, Whereas the min, max and average time taken by a SQL delete query is 3ms, 898ms, and 426.2ms respectively.

5.3 Semantic GIS Service Provider

5.3.1 Implementation of Semantic GIS Service Provider

The following figures illustrate the execution of the GIS service prodder module, and the GIS support toolbox module. The GIS support toolbox module uses the services provided by the GIS

service provider module to perform location information management. Figure 66 shows the semantic GIS provider execution screen. GIS provider offers Provider service and Contents service. Time now label shows the duration since GIS provider is started. Provider service label shows the operational state of GIS provider's Provider service. Content Service label shows the operational state of GIS provider's Contents service. Map Number label shows the number of outdoor maps available in the ontology. Building Number label shows the number of buildings information saved in ontology.

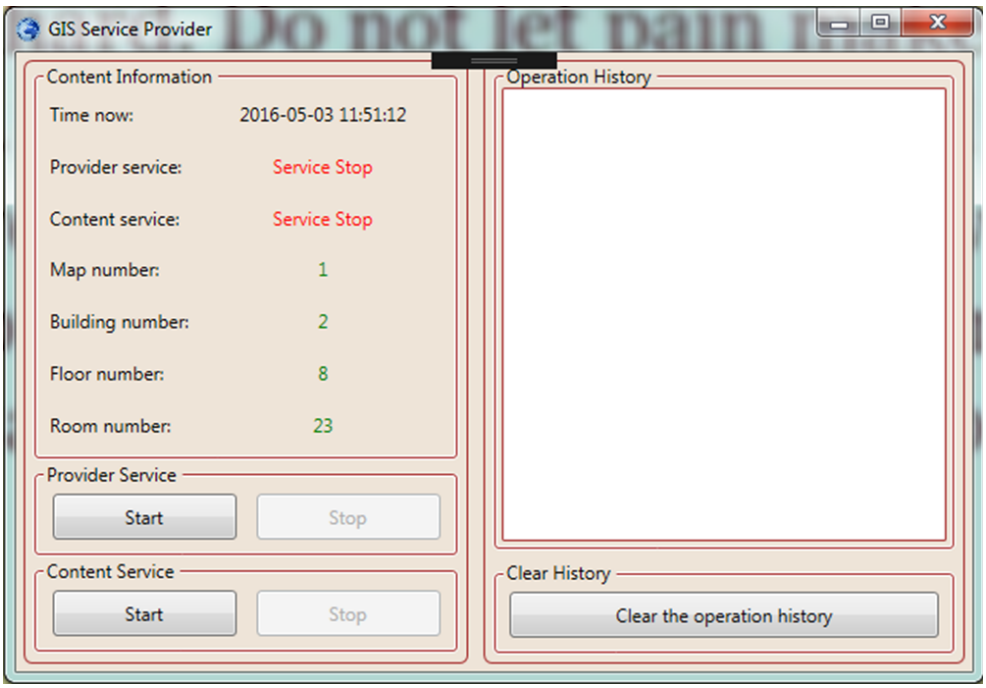

Figure 66 Semantic GIS Service provider

Floor Number label shows the total number of the floors information saved in ontology. Room Number label shows the total number of room's information saved in ontology. The Start and Stop buttons are used to start and stop the respective services. Operation History records the GIS service control work.

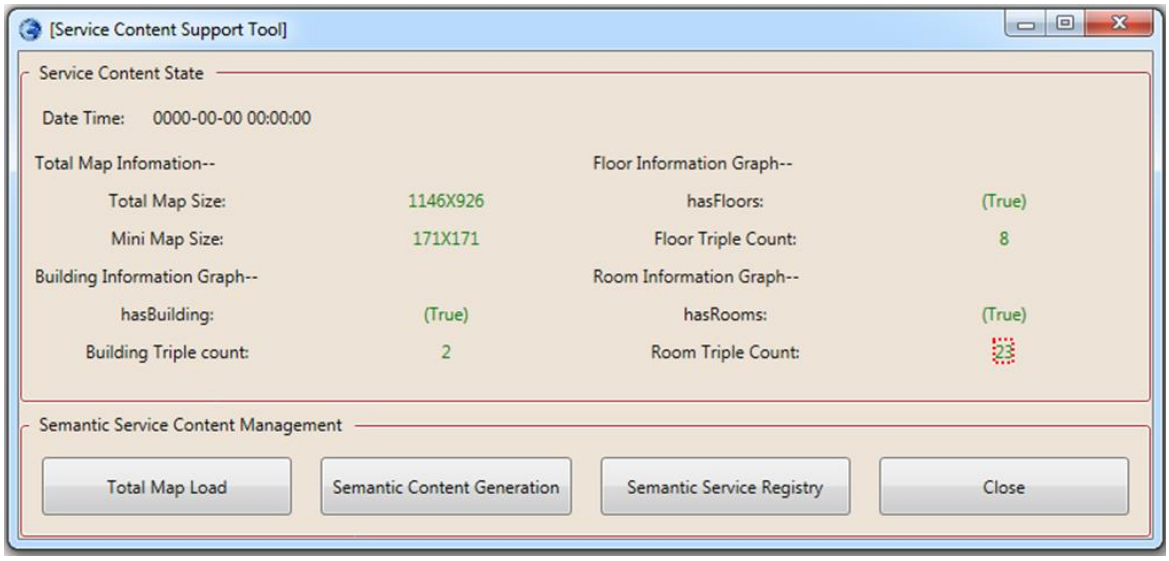

Figure 67 Semantic GIS Support Toolbox

Figure 67 shows the GIS service composite tool execution screen. Service contents tool screen shows service provider's content state information directly to the management. It also displays the total map information, building information, state floor information and state room information and state etc. Date Time is the GIS service composite tool's environment execution time. It also provides the interface to call semantic information management screens. Total Map Information show outdoor map image and mini map image sizes. Building Information shows the number of registered buildings, Floor Information shows the number of registered floors and Room Information shows the number of registered rooms in the ontology.

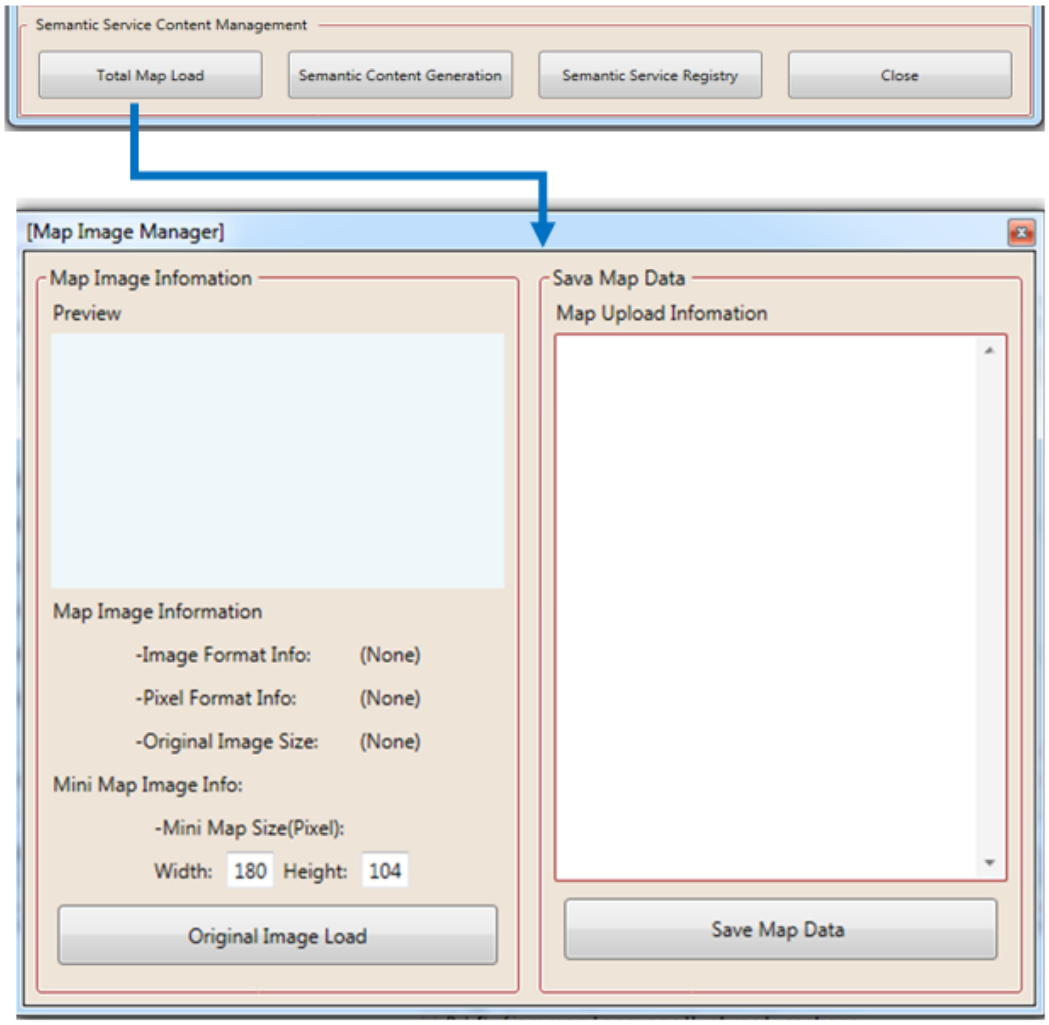

Figure 68 Map Image Manager

Figure 68 shows the map image management interface. Map image management loads total map image, divides it into parts and send it to service provider piece by piece. It is necessary because the map file size for the entire school will be a very large file. If client downloads and display this file, the delay becomes excessive. Thus the problem is solved here by dividing the map image file into smaller partition sand saving the partition information and partition image data. Area on the left side is the interface to load map image and save it to memory.

It also shows information about the image format, size etc. and provide controls to specify the mini map size. Area on right side displays the map image division and saving process. The image displayed by Figure 69 here loads by clicking the semantic content generation control from the

main window of the support toolbox module shown in Figure 67. It shows the outdoor map management screen execution.

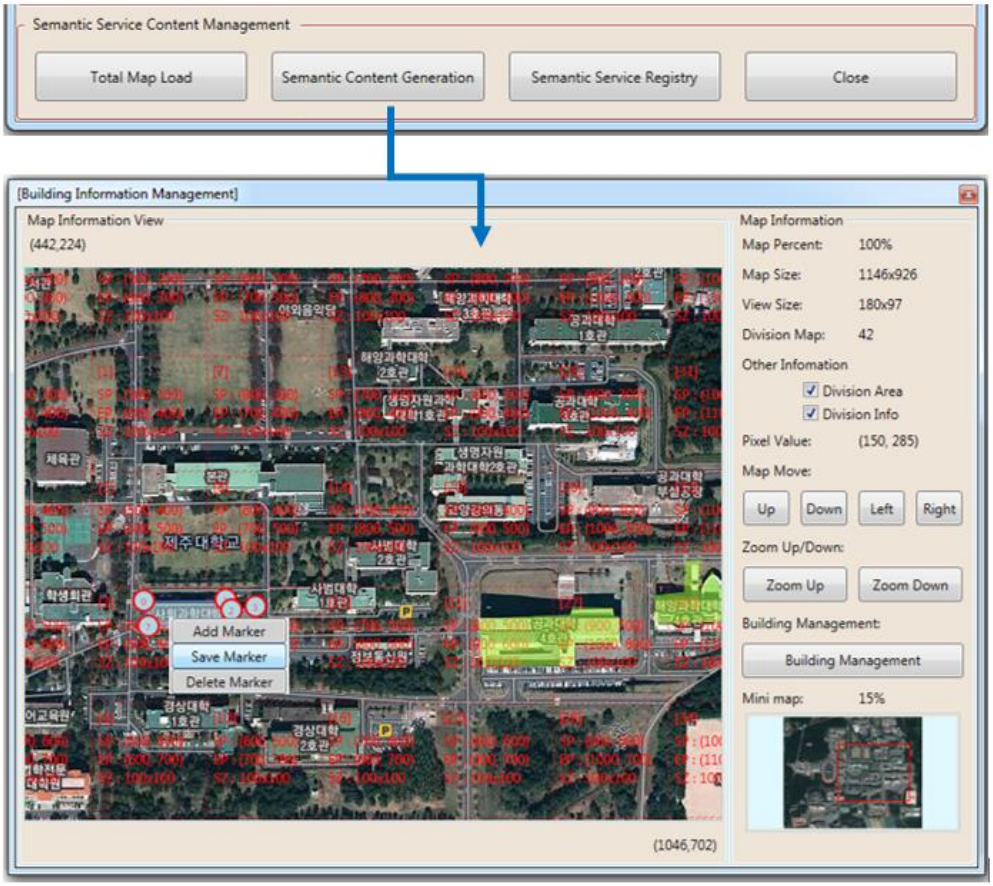

Figure 69 Building Information Management

Outdoor map is large range map viewer. Minimum visualization unit is building. Total map viewer provides the visualization and creation of building information on the map. It displays map information on the right side of the map. Map Percent shows current map percent. Map Size means current map total size. View Size means current map visible range. Division Map shows the number of map divisions within the range of the viewer. Other Information is for displaying division grid and division info on the map inside the map viewer. Pixel Value means mouse pointer location on the map. Map Move offers controls for map movement and adjustment. From this interface you can select a building by adding the building marks and saving them using the menu item displayed. Once the save marker button is clicked it displays the form shown in the next figure.

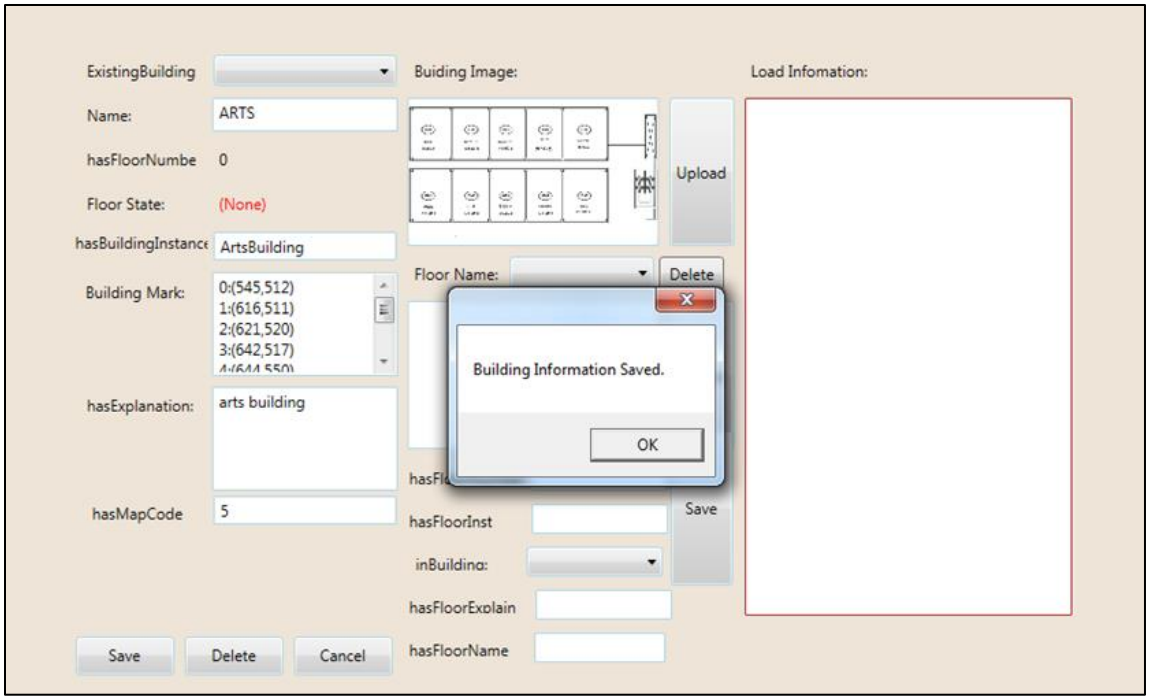

Figure 70 Semantic Building Management

Figure 70 shows the form for building management. The controls available are add building, and delete building. To add new building the information needed is

- Building name: a string typed name of the building
- Building instance: an instance of the building
- Has explanation: explaining the purpose of the building
- Has map code: the map that this building belongs to
- Building image: stores the image for the building.

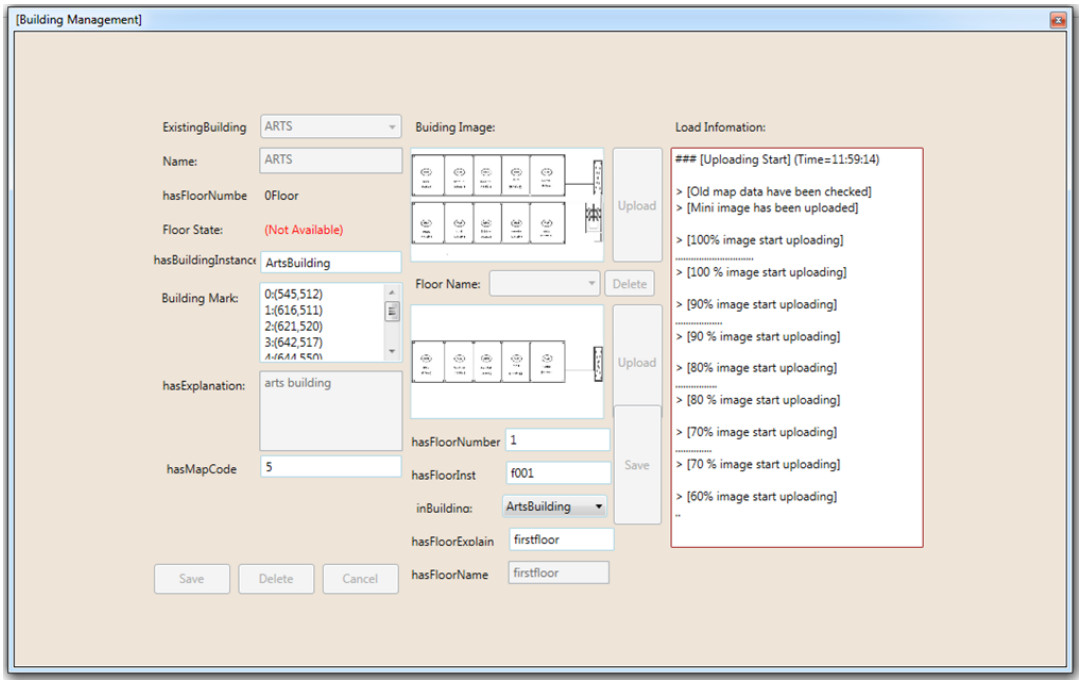

Figure 71 Semantic Floor Management

The same form as displayed in Figure 70 displays text boxes and controls to add floors to each building as shown in Figure 71. The information needed to add a floor is:

- Floor number: defines the number of the floor
- Floor Instance: defines the instance of the floor information class in the ontology
- In building: defines the object property that is used to associate a floor to a building
- Floor explain: description of the floor
- Floor name: defines a string based name of the floor
- Floor image uploads an image for that specific floor

When the save button is clicked the status for dividing the image into smaller portions is shows in the list. Figure 72 shows the indoor map management execution screen. It provides interface for demarcating room area on the floor map. Map Percent labels the current map size in percent, Map Size means current map total size, View Size means current map visible range, Division Map shows the number of map partitions inside the viewer, Other Information provides options for showing the map division grid and division information on the map, Pixel Value means the current

location of mouse pointer on the map, Map Move offers controls for map movement and adjustment, and Zoom Up/Down are controls for adjusting the map size (larger/smaller).

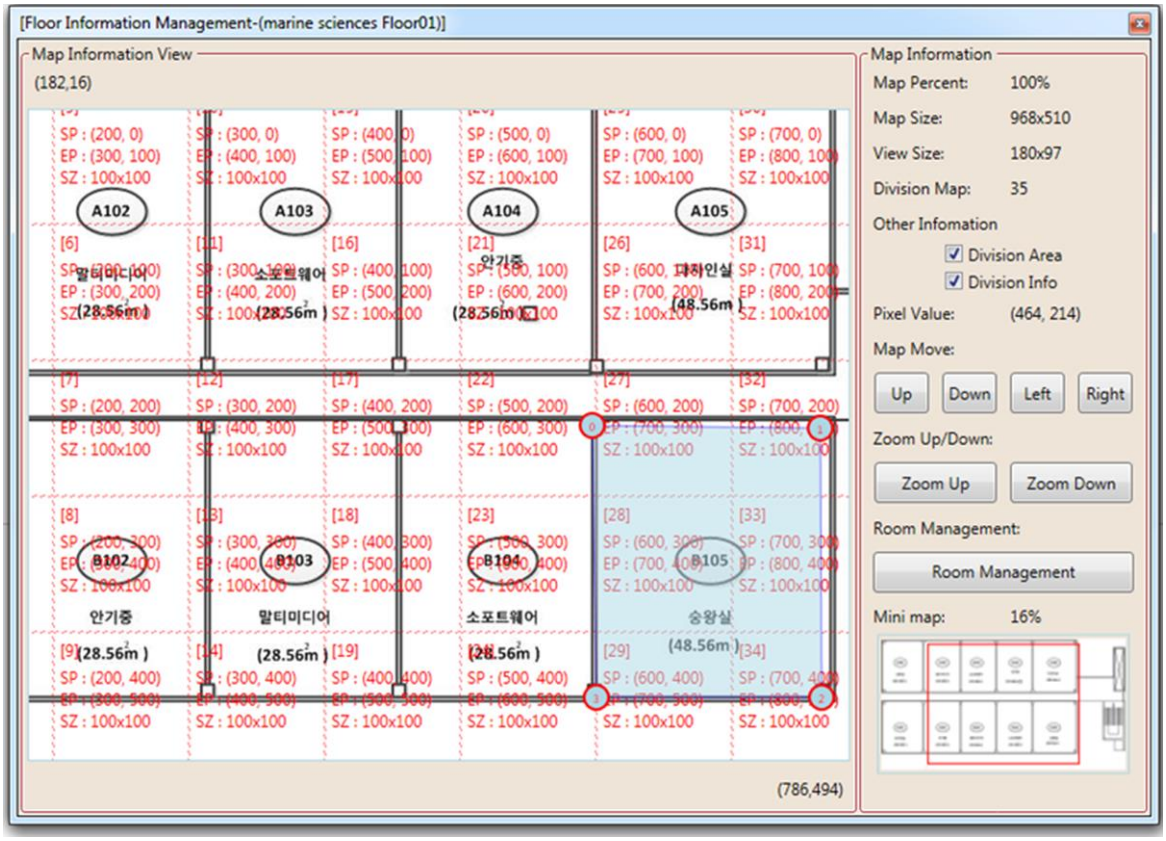

Figure 72 Indoor Map Management

Figure 73 shown displays the form for adding room information to the ontology. The information needed to add a room is:

- Room no: defines string based no of the room
- Room name: defines the room name
- Room mark shows the room marks selected by the user
- Room image loads an image for the room
- On floor associated the room with a floor
- Room instance defines the instance of the room class. On clicking the save button the room information is stored to the ontology.

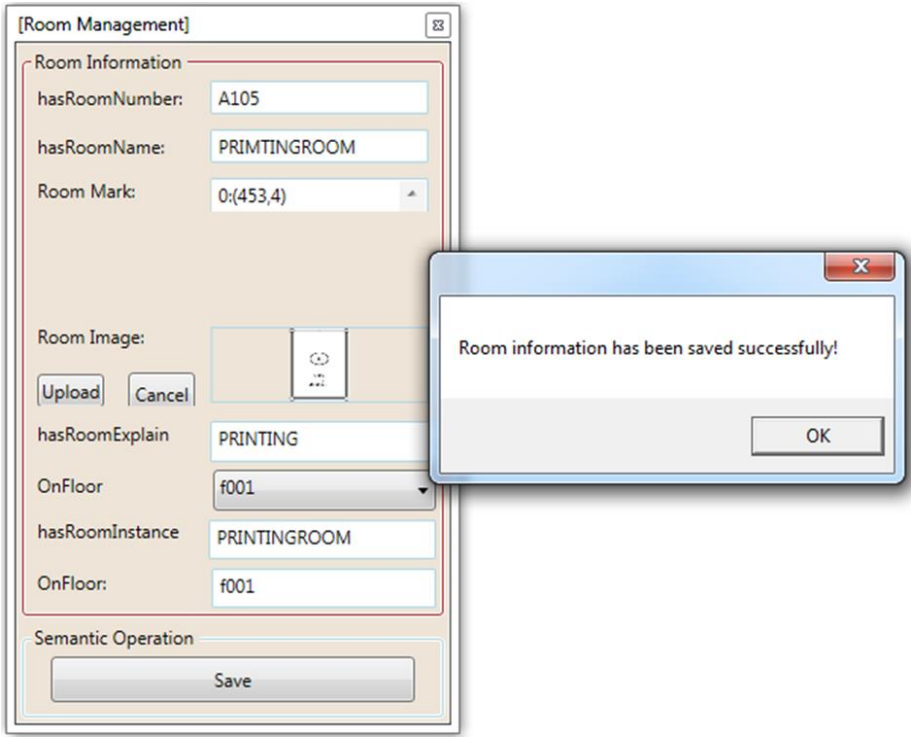

Figure 73 Semantic Room Information Management

5.3.2 Development and Reasoning of GIS Provider ontology

This section illustrates the development of an owl ontology for the GIS provider module that represents all the location information of the sensors and actuators registered in the system. Figure 74 shows the classes defined in the ontology which are Building Information that stores information about any building on the given map, Building Management, Floor Management, and Room Management classes represents the management performed by the GIS support toolbox module. Map Information class represents the buildings registered in a specific map stored.

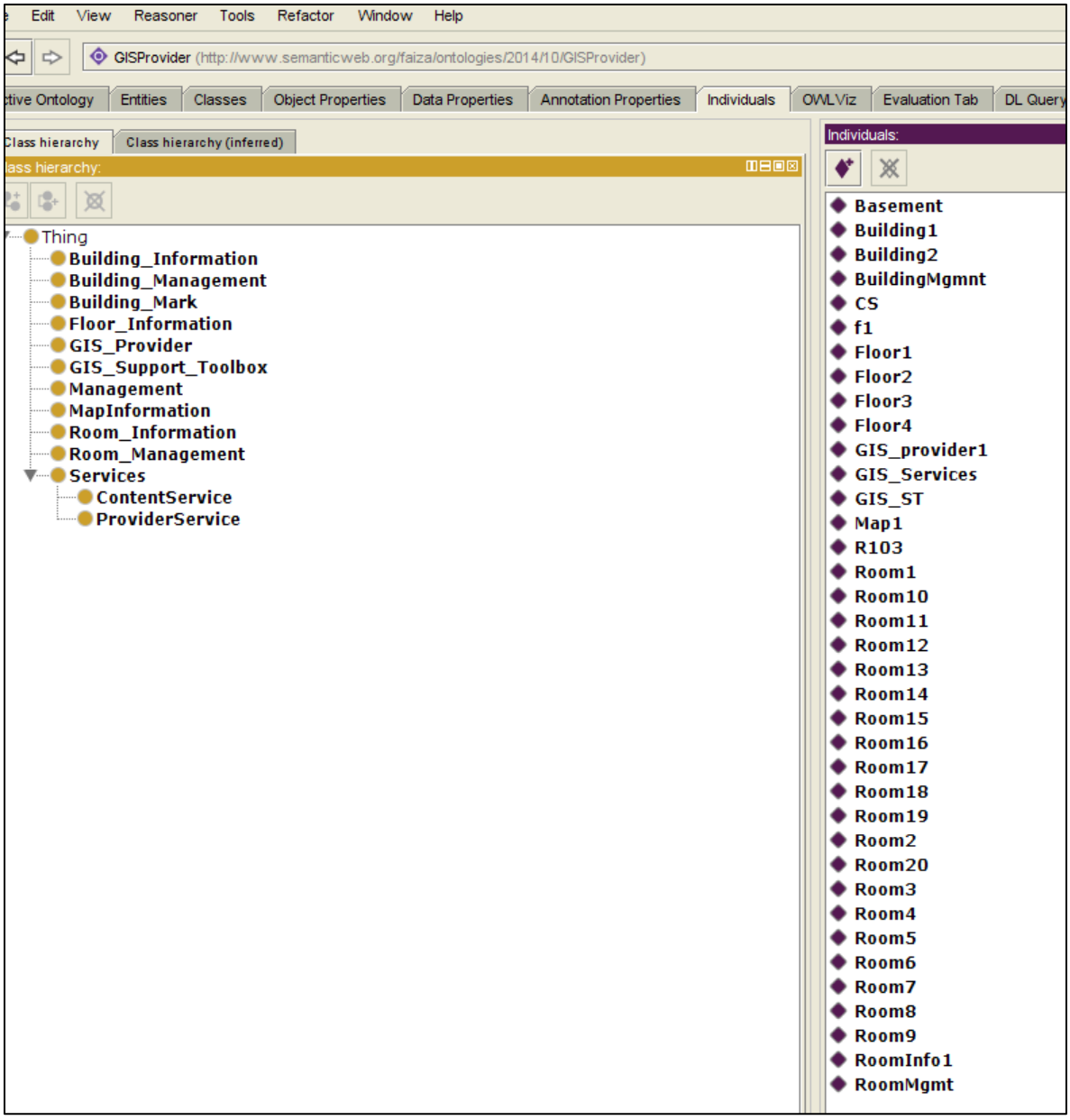

Figure 74 GIS Provider ontology

Floor Information class represents all the floors that are registered in a building whereas Room Information class represents all the rooms that are registered in a specific floor. Services class represents the services that are offered by the GIS Provider module. Figure 75 shows the reasoning performed on the GIS provider ontology. The object property hasFloor is inferred in the figure. This can help the user query for floors and discover new relations. The inferencing shown in this

study will be used after publishing the ontologies online which is included in future work of the thesis.

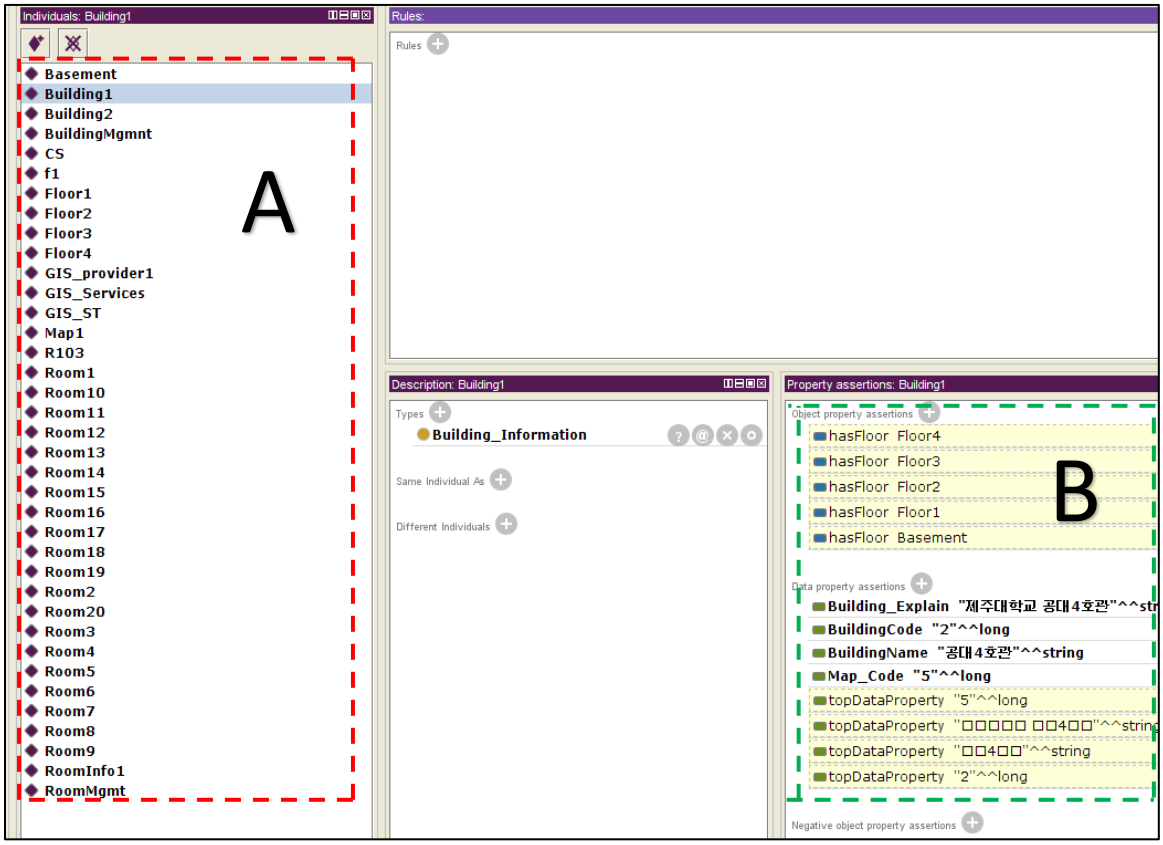

Figure 75 Reasoning on the GIS provider ontology

5.3.3 Performance Analysis of Semantic GIS Service Provider

This section provides the results of SPARQL and SQL query comparison performed for analyzing the semantic GIS service provider system performance. All the comparisons are based on the min, max and average time taken in milliseconds for 10 iterations of each query. The graph shown in Figure 76 displays SPARQL and SQL add query comparison based on the min, max and average time taken in milliseconds. These queries saves building information to the semantic GIS service provider repository. The min time taken in milliseconds by SPARQL add query is 41ms, the max time taken is 54ms and the average time taken is 47.6ms. Whereas the min, max, and average time taken by the SQL add query is 46ms, 972ms, and 459.8ms respectively.

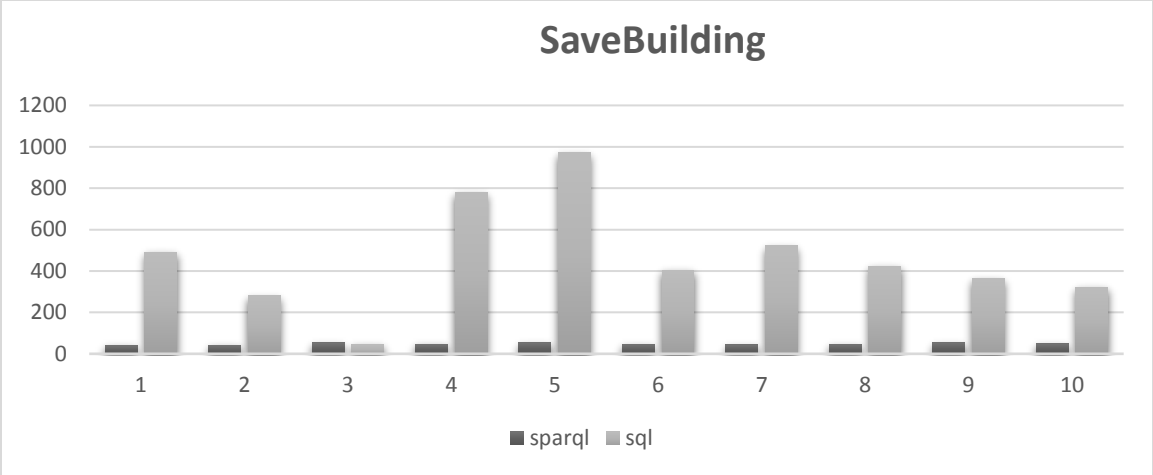

Figure 76 Query comparison for saving building information

Figure 77 shows a graph that describes the comparison results for SPARQL and SQL queries based on the min, max and average time taken in milliseconds. The graph shows the comparison results taken for 10 iterations of each query. The min time taken for a SPARQL query to add floor information to the semantic GIS service provider repository is 48ms, the max time taken is 73ms and the average time taken is 54.9ms. Whereas the min, max and average time taken for a SQL query to add floor information to the semantic GIS service provider repository is 30ms, 973ms and 419.6ms.

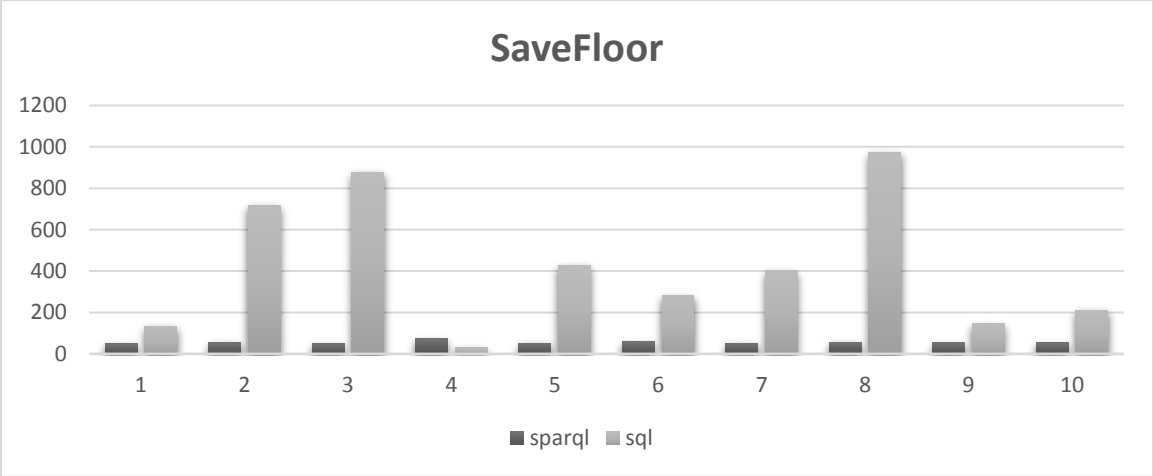

Figure 77 Query comparison for saving floor information

Figure 78 shows a graph that describes the comparison results for SPARQL and SQL queries based on the min, max and average time taken in milliseconds. The graph shows the comparison results taken for 10 iterations of each query. The min time taken for a SPARQL query to add room information to the semantic GIS service provider repository is 88ms, the max time taken is 292ms and the average time taken is 161.6ms. Whereas the min, max and average time taken for a SQL query to add room information to the semantic GIS service provider repository is 132ms, 990ms and 607.7ms.

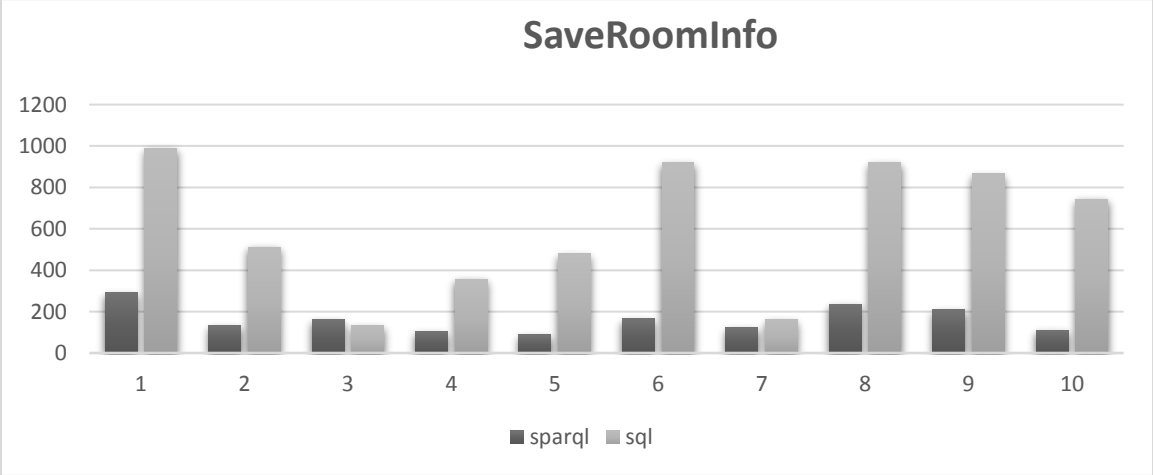

Figure 78 Query comparison for saving room information

5.4 Semantic Application Server

5.4.1 Implementation of Semantic Application Server

Application server is the top most module in semantic IoT system. Figure 79 shows the application server execution screen. It shows information saved in application server ontology such as object state. It controls the Provider Service, Content Service and Smart Control service. Time now shows the application server's execution environment real time. Sensor Count is the number of sensor object that bound in the ontology. Actuator Count is the number of actuator objects bound in the ontology. Work Time is the Normal operation time of service. Service State means the service operational state. Service Start and Service Stop buttons control the service on/off.

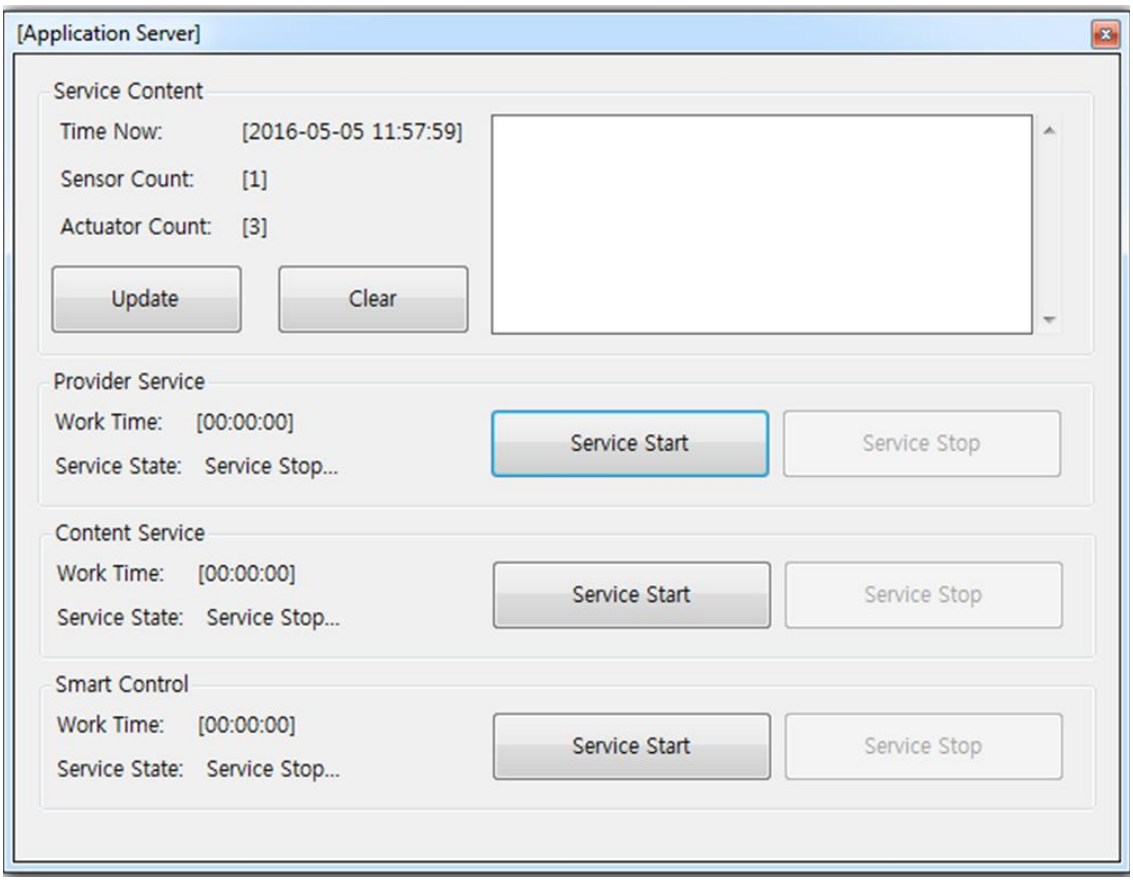

Figure 79 Semantic Application Server Module

Figure 80 shows the application server composite tool execution screen. It presents controls to user for performing map binding, semantic object binding, and service publishing. By clicking each button user can access the forms for manipulating the data stored in the ontology. Through the Map Binding button, go to map service binding management. Press Semantic Object Binding button to bind sensor or actuator data to its location data. Press service register button to show service information management window. It shows the no of the services registered in the application server ontology. It shows the binding status of the services. Object count shows the total no of objects registered in the application server ontology.

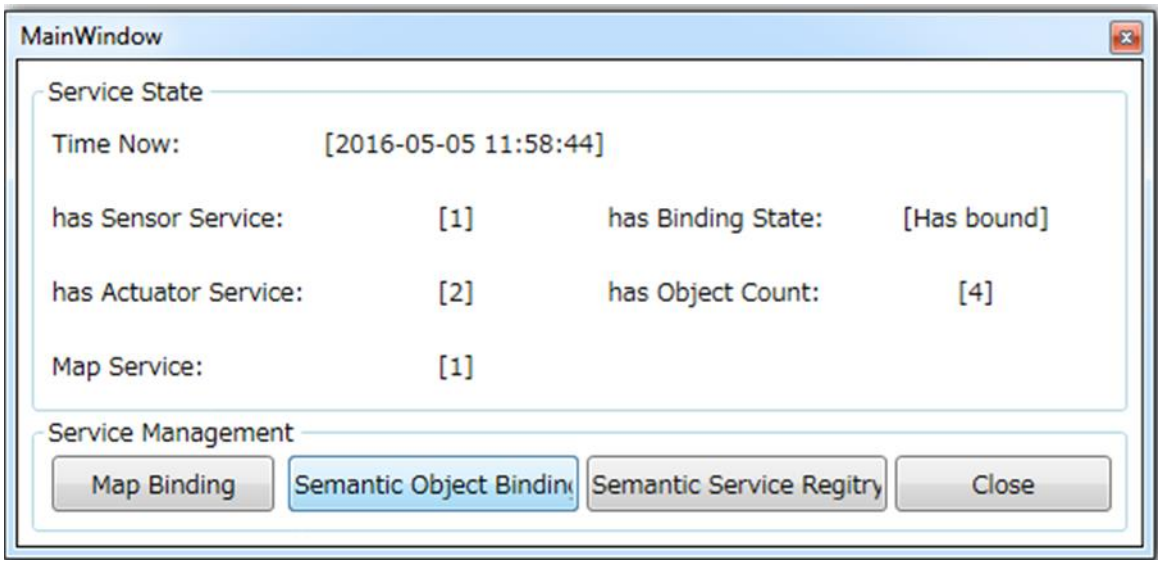

Figure 80 App Server Support Toolbox

Figure 81 shows the map service binding management execution screen. This interface is used for searching map service and binding selected map service. Service Name is GIS service name. Service Uri means GIS service access address. Key Word is GIS service search keyword. Service Refresh button performs the GIS service search again and refresh the list. Map Binding button binds GIS service with a map. Binding Cancel deletes GIS service information that is already bound.

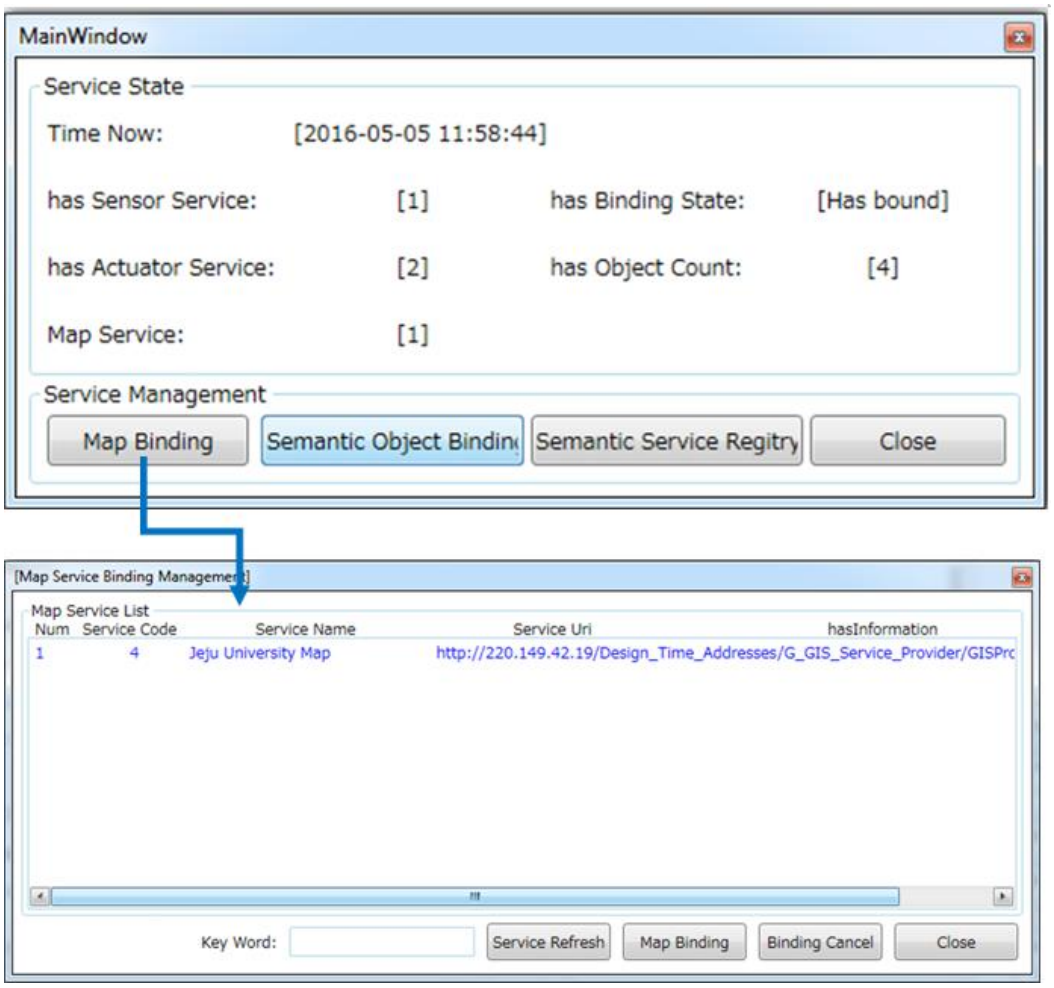

Figure 81 Map Service Binding

Figure 82 shows the outdoor map viewer execution screen which is displayed by clicking. Outdoor map viewer offers visualization of outside map and building information on the map. The buildings registered in the ontology are shows colored with mouse hovering. Selecting a registered building retrieves the floors that are registered with that specific building. The floor name is displayed in a menu item as shown in the figure. The floor menu item consists of the floor name's that are associated with the selected building. On clicking the floor name the form shown in the next figure is displayed.

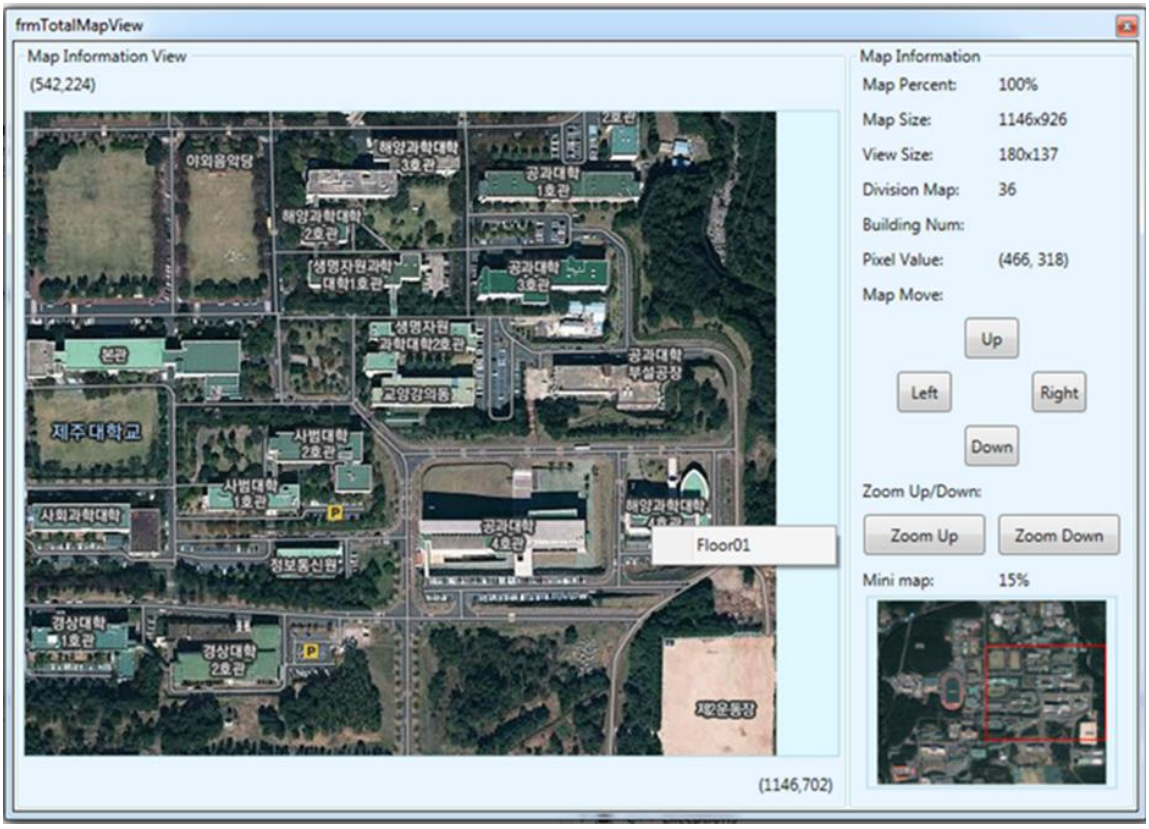

Figure 82 Outdoor Map Viewer

The form shown in the Figure 83 displays the rooms that are registered with the particular floor selected from the previous image. The user is now able to semantically bind objects in these rooms. The title of the floor displays the building name and the floor number. On clicking a room the menu item for binding a sensor or an actuator in the room is displayed. Other information displayed on the map is: Map Percent labels the current map size in percent. Map Size means current map total size. View Size means current map visible range. Division Map shows the number of map partitions inside the viewer. Pixel Value means the current location of mouse pointer on the map. Map Move offers controls for map movement and adjustment. Zoom Up/Down are controls for adjusting the map size (larger/smaller).

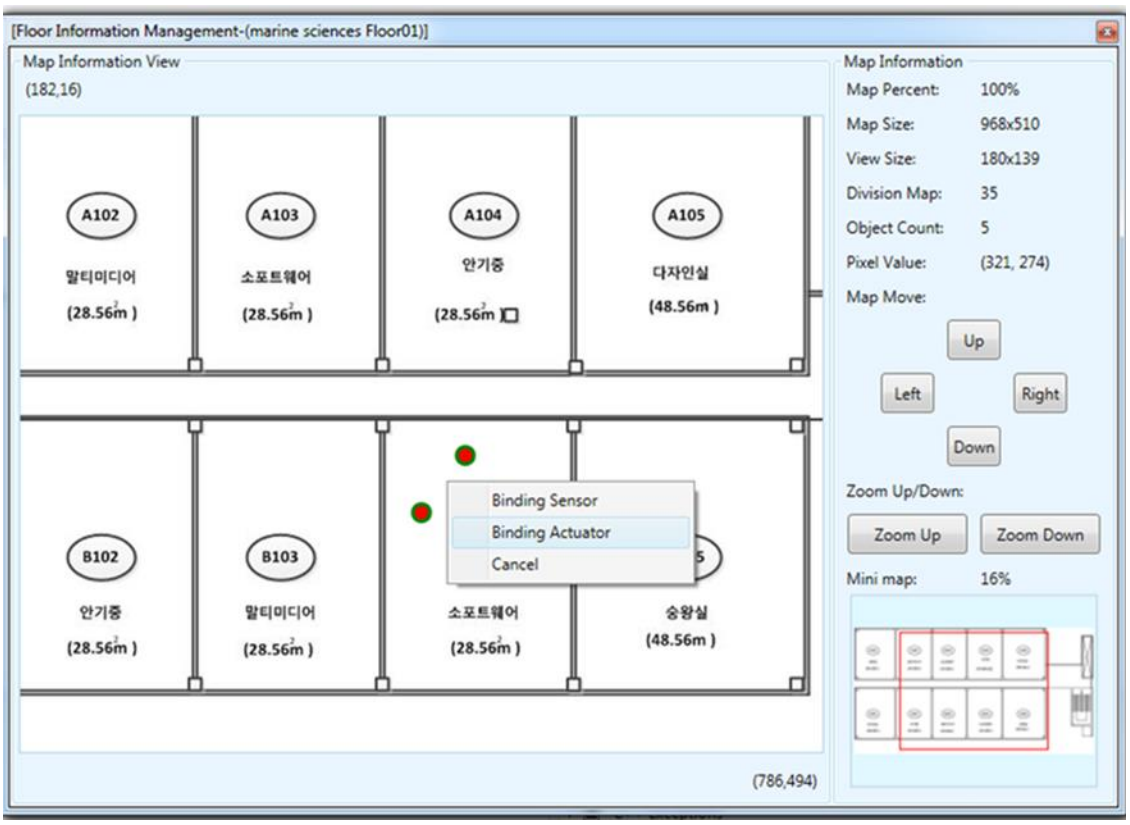

Figure 83 Semantic Object Binding

Query1:

```
INSERT DATA
 {app:Actuator1 rdf:type app:Object_Information
 app:ObjectID '#DS00005'^^xsd:string;
 app:ObjectType 'Actuator'^^xsd:string;
 app:BuildingCode '2'^^xsd:long;
 app:FloorCode '18'^^xsd:long;
 app:RoomCode '7'^^xsd:long;
 app:ObjectCode '10'^^xsd:long;
 app:ProviderUri
 '<http://220.149.42.19/G_Actuatorweb_Service_Provider/clsProviderService/>'^^xsd:string;
app:hasService app:MapService1}.
```

Query 1 shows the SPARQL query for registering/binding an object to the ontology. The first triple pattern selects the type of the instance by using the type property from RDF vocabulary. The second triple pattern inserts the specified id of the object using the data property ObjectID, the third triple pattern inserts the type of the object using the data property ObjectType, the third triple

pattern associated the object with a building by inserting the building code, the fourth triple pattern inserts the floor on which the object is, the fifth triple pattern inserts the room where the object physically is, the sixth triple pattern inserts the Uri of the object's provider i.e if it is an actuator this Uri will have the value of semantic actuator service provider and if it is a sensor then this Uri will have the value of semantic sensor service provider.

And the last triple pattern associated the map service for the object. The map service provides an Uri that is used to visualize the indoor map area. Figure 84 shows the results of query 1. It displays a screen shot from the application server ontology that illustrates the binding of an object to its location information using object and data properties.

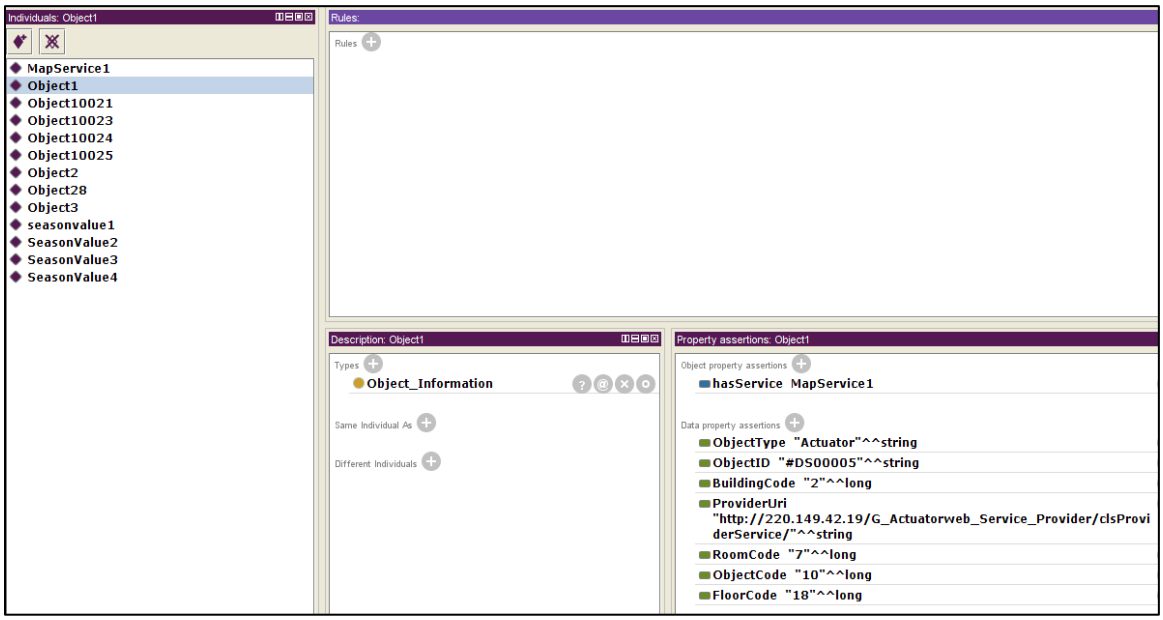

Figure 84 Query 1 result

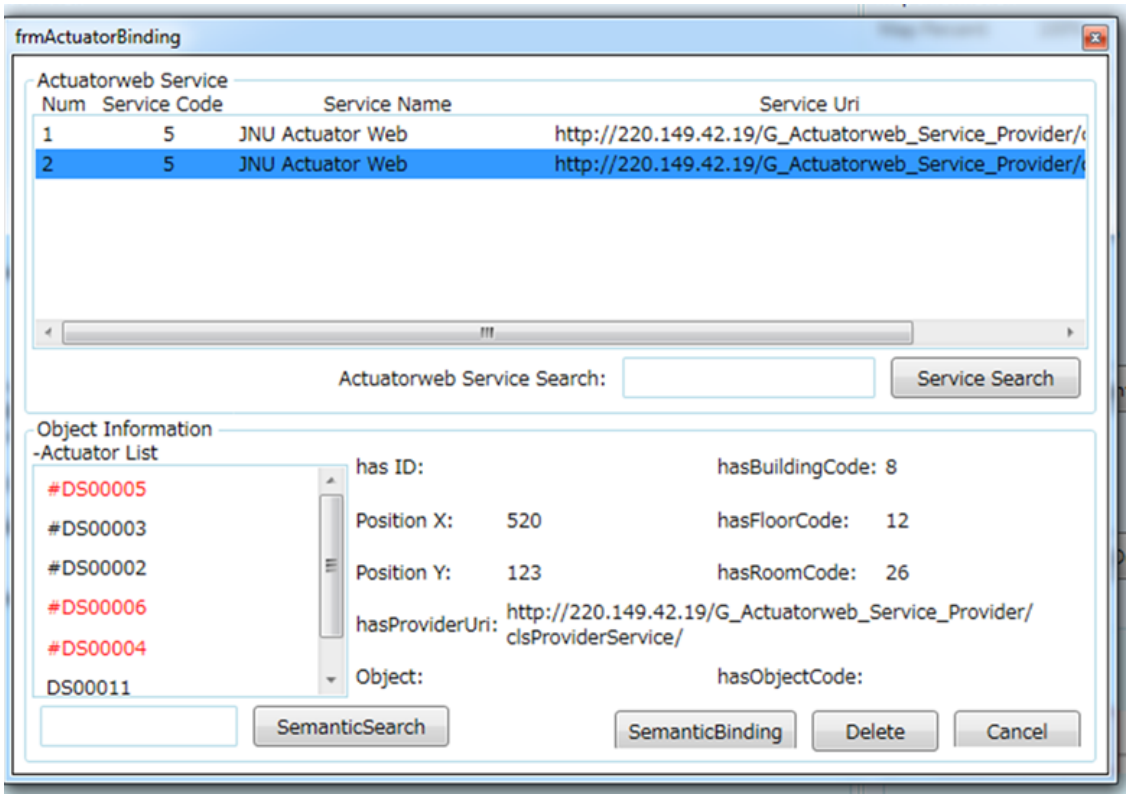

Figure 85 Semantic Actuator Binding

Figure 85 displays a form for binding an actuator to a room. It consists of two groups semantic actuator service and object information. Semantic actuator services shows list of services for the actuators that are registered in the ontology. The information displayed about a service consists of Service code, Service name, and Service Uri. Object information shows an actuator list that consists of actuator id's retrieved from the ontology. The object information displayed is

- Has Id: associates an id with the actuator
- Position x: shows the x position of the actuator in the room
- Position y: shows the y position of the actuator in the room
- Building code: shows the code of the building where the room is
- Floor code: shows the code of the floor on which the room is
- Room code: shows the code of the room
- Object: shows the instance of the object information class
- Object code: shows the code of the object

Figure 86 displays a form for binding a sensor to a room. It consists of two groups semantic sensor service and object information. Semantic actuator services shows list of services for the

sensors that are registered in the ontology. The information displayed about a service consists of Service code, Service name, and Service Uri. Object information shows a sensor list that consists of sensor ids retrieved from the ontology.

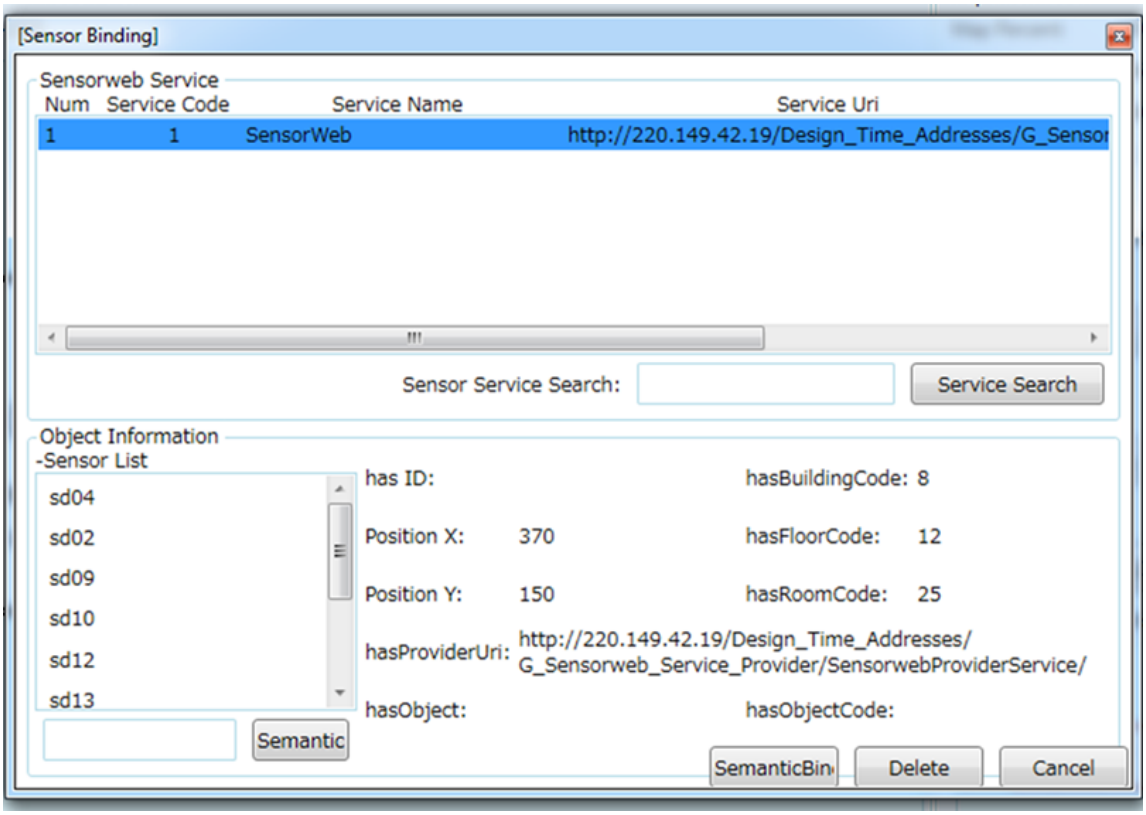

Figure 86 Semantic Sensor Binding

The object information displayed is:

- Has Id: associates an id with the sensor
- Position x: shows the x position of the sensor in the room
- Position y: shows the y position of the sensor in the room
- Building code: shows the code of the building where the room is
- Floor code: shows the code of the floor on which the room is
- Room code: shows the code of the room
- Object: shows the instance of the object information class
- Object code: shows the code of the object

5.4.2 Smart Control Implementation

The comfort index calculation method used in this thesis was developed by Fanger (1972) and adapted in ISO standard 7730. It is based on the determination of the PMV index (Predicted Mean Vote) calculated from an equation of thermal balance for human body shown below.

$$S = M - W \pm R \pm C \pm K - E \pm Res$$

The variables used in the equation are defined below:

- M: Metabolic heat production of a human body
- W: Mechanical work
- R: Radiation exchanges with surroundings
- C: Convection exchanges with air layers
- K: Conduction to or from clothing
- E: Evaporation losses in sweating, and
- Res: wet and dry heat exchanges in respiration.

Table 2 comfort index scale

PMV Index	ET Index	Comfort State
2.5~3	35~41	Very Hot
1.5~2.5	29~35	Warm
0.5~1.5	23~29	Slightly Warm
-0.5~0.5	18~23	Normal
-1.5~-0.5	13~18	Slightly Cool
-2.5~-1.5	8~13	Cool
-3~-2.5	4~8	Very Cold

Fanger established a model of correlation between the subjective human perception, expressed through the vote of comfort on a scale ranging from -3 (very cold) to +3 (very hot) shown in table 2, and the difference between the heat generated and the heat released by the human body, which corresponds to the following equation [30]:

PMV = (0,303e-2,100*M + 0,028)*[(M-W) - H - Ec - Cres - Eres]
(1)

Where the different terms represent, respectively:

- M - the metabolic rate, in Watt per square meter (W/m2);
- W - the effective mechanical power, in Watt per square meter (W/m2);
- H - the sensitive heat losses;
- Ec - the heat exchange by evaporation on the skin;
- Cres - heat exchange by convection in breathing;
- Eres - the evaporative heat exchange in breathing.

In equation 1, the terms H, Ec, Cres, and Hres, correspond to the heat exchange between the body and the surrounding environment and are calculated from the following equations:

H = 3,96*10-8*fcl*[(tcl+273)4 - (tr+273)4] - fcl*hc*(tcl-ta) (2)

Ec = 3,05*10-3*[5733 – 6,99*(M-W)-pa]-0,42*[(M-W)-58,15] (3)

Cres = 0,0014*M*(34-ta) (4)

Eres = 1,7*10-5*M*(5867-pa) (5)

Where:

- Icl is the clothing insulation, in square meters Kelvin per watt (m2 K/W);
- fcl is the clothing surface area factor;
- ta is the air temperature, in degrees Celsius (°C);
- tr is the mean radiant temperature, in degrees Celsius (°C);
- var is the relative air velocity, in meters per second (m/s);
- pa is the water vapor partial pressure, in Pascal (Pa);
- tcl is the clothing surface temperature, in degrees Celsius (°C).

The other index proposed in ISO Standard 7730 is PPD (Predicted Percentage of Dissatisfied) that quantifies the expected percentage of dissatisfied people in a given thermal environment shown in the equation 6 [30]. Thermal comfort zones (A, B and C classes) are defined in by the ranges of PMV values from -0.2 to 0.2, -0.5 to 0.5 and -0.7 to 0.7, which correspond respectively to PPD values below 6, 10 and 15%.

PPD =100 −95· e−(0.03353·PMV 4 +0.2179·PMV 2) (6)

$$ET = DBT - 0.4 * (DBT - 10) * (1 - RH/100) \text{ in C} \quad (7)$$

Using the normalization formula and applying to each PMV and ET index [31], you can convert the index range from 0-1. Using these three formula comfort index can be easily calculated. To easily let the user know the total comfort index, there should be a total comfort state rule. So averaging the normalized PMV and ET ranges the total comfort state rules can be achieved as shown in table 3.

Table 3 combined values

Comfort State	PMV Value	ET Value	Combine Value
Hot	0.916~1	0.838~1	0.877~1
Warm	0.75~0.916	0.676~0.838	0.713~0.877
Slightly Warm	0.583~0.75	0.514~0.676	0.5485~0.713
Normal	0.416~0.583	0.378~0.514	0.397~0.5485
Slightly Cool	0.25~0.416	0.243~0.378	0.2465~0.397
Cool	0.083~0.25	0.108~0.243	0.0955~0.2465
Cold	0~0.083	0~0.108	0~0.0955

Table 4 shows the fuzzy rules to optimize indoor temperature. For example, some indoor space PMV environment state is cold and ET environment state is Slightly Cold and the combination comfort state is cold. Thus for same PMV State and ET State the Comfort State is the same. So for Comfort State Cold, raise indoor temperature and when hot, drop the indoor temperature. Here the Smart Control Algorithm is presented which collects environment information and based on this information it controls the appliances in two ways. The following flow chart describes each plan.

Table 4 comfort index fuzzy rules

PMV State	ET State	Comfort State	PMV State	ET State	Comfort State
Cold	Cold	Cold	Slightly Warm	Normal	Normal
Cold	Cool	Cold	Normal	Normal	Normal
Cold	Slightly Cool	Cold	Hot	Hot	Hot
Cool	Cool	Cold	Hot	Warm	Hot
Cool	Cold	Cold	Hot	Slightly Warm	Hot

Cool	Slightly Cool	Cold	Warm	Warm	Hot
Slightly Cool	Cold	Cold	Warm	Hot	Hot
Slightly Cool	Cool	Cold	Warm	Slightly Warm	Hot
Slightly Cool	Slightly Cool	Cold	Slightly Warm	Hot	Hot
Normal	Slightly Cool	Normal	Slightly Warm	Warm	Hot
Normal	Slightly Warm	Normal	Slightly Warm	Slightly Warm	Hot
Slightly Cool	Normal	Normal			

Figure 87 shows the message routing process from application server to an object (sensor, actuator). According to the figure assume that application server sends a message to an actuator with id=DS00012. First, from the application server mapping table it gets the corresponding provider service (semantic actuator service provider in this case) address and forward the message on that address. The service provider receives the message and finds the corresponding middleware address from the service provider's mapping table.

Finally, middleware receives the message and finds the object IP address from the middleware mapping table and send the message to the object. For response message the process is reversed. The object sends message through saved middleware IP. Middleware sends message through the service provider Uri. Service provider receives the message, processes it and saves it in the DB. At some point application server request data from service provider and provider responds with the data from the ontology.

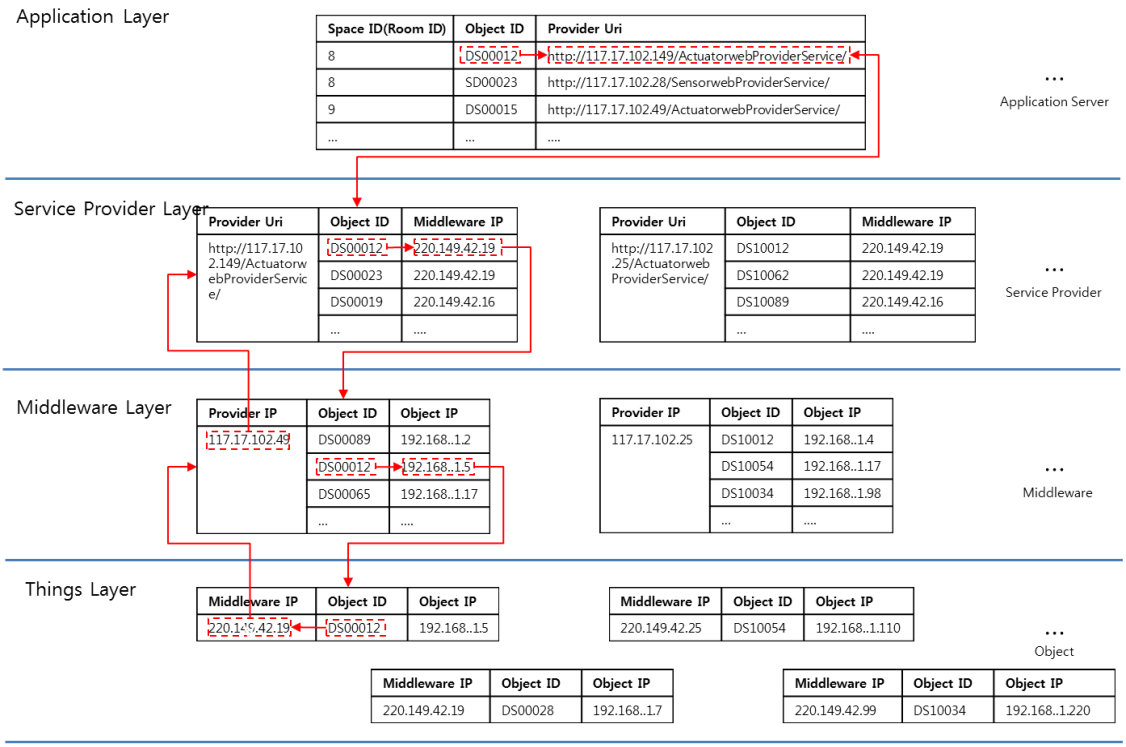

Figure 87 Object routing process

If sensor is connected to the sensor network and started, it will automatically create sensing data and send to the service provider. Sensor provider process receives sensing data and saves it in its DB. So for application server to receive some sensor node sensing data, it just need to request sensing data which is saved in Sensor provider using sensor ID and the service provider address. Actuator control process is little complicated. For actuator control, application server sends control command to specific actuator. However, in the actual system architecture, Actuator provider acts as a server and the actuator acts as client. Figure 88 shows sensor and actuator connection routing plan. Application server smart control module connects the sensor and actuator through area information (Room ID). Actually, it is not connected directly to the sensor node but in fact it requests the desired information from Sensor Web provider's ontology. Using the actuator ID, it gets the target actuator from mapping table and sends the message to it.

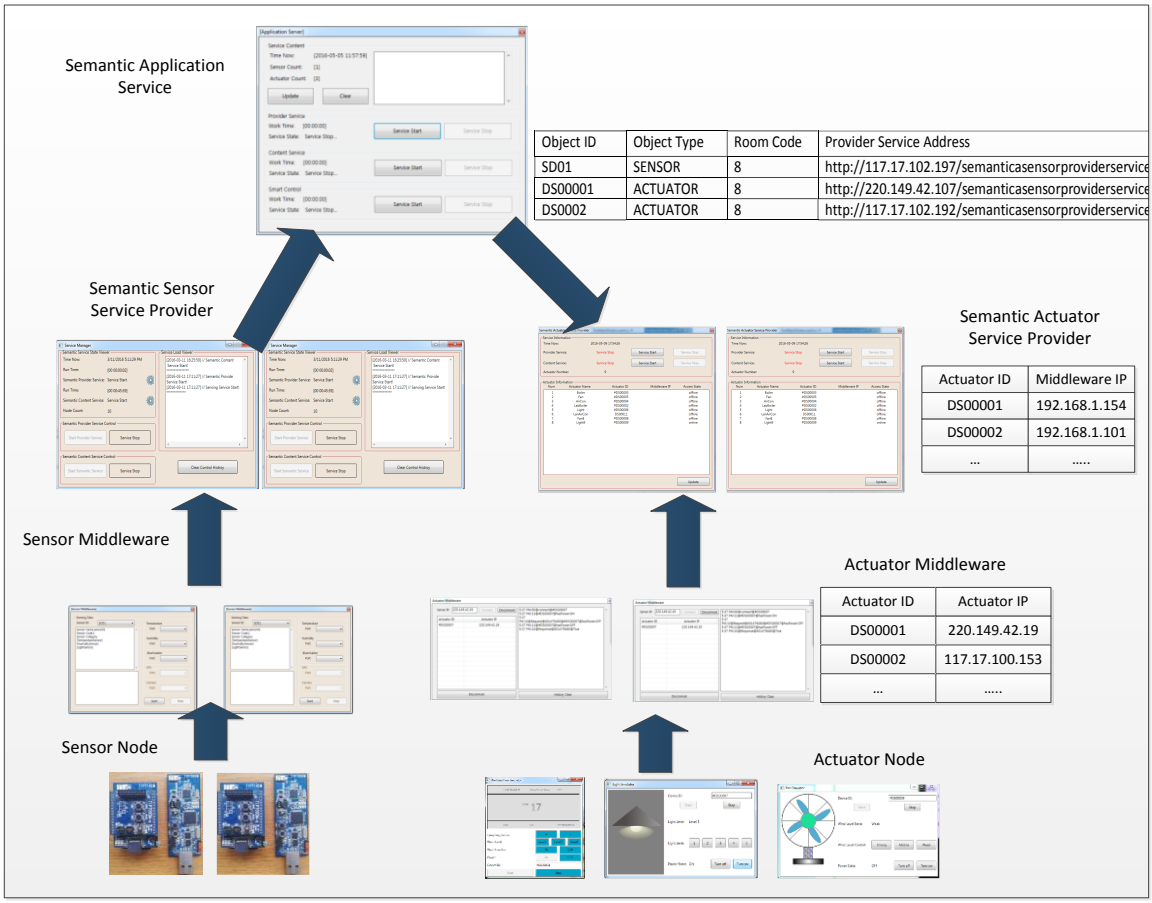

Figure 88 Sensor and actuator control routing plan

5.4.3 Actuator Emulator Control

This section uses figures to describe the implementation of the actuator control performed by the actuator emulator, actuator middleware and semantic actuator service provider and application server. In the IPV6 era, technologies such as smart home are evolving rapidly. In the future all appliances will be equipped with sensor chips and a user can connect remotely to it using network/internet. Here, instead of discussing the device hardware architecture, an emulator is implemented to imitate real-world appliances. Figure 89 shows the device emulators that are connected to the system.

Figure 89 Device emulators

Figure 90 shows the process of fan control from the application server module. The control process is based on the evaluation of comfort index using PMV, PPD and ET values. Area 1 indicates the control messages exchanged between the application server and the fan emulator through the actuator middleware. Area 2 shows the fan emulator screen. It offers functions like airflow adjustment and power control. Wind Level State show fan Current wind speed rating. Wind Level Control can adjust to 3 values of wind speed i.e. Strong, Middle and Weak. Power

State can adjust power ON/OFF. The comfort index calculated based on these values is less than 0.4585 which according to table 4 indicates normal comfort state. The message exchange shown in area 2 of the figure illustrates the actuator control based on this comfort index value. As the user starts the fan by clicking the turn on button, the middleware receives message from the application server to switch off the fan, the actuator middleware turns the fan off and sends an acknowledgement message to the application. Figure 90 shows the process of controlling the light from the application server. The light emulator offers functions to control illumination and power. Light Level show light appliance Current illumination level.

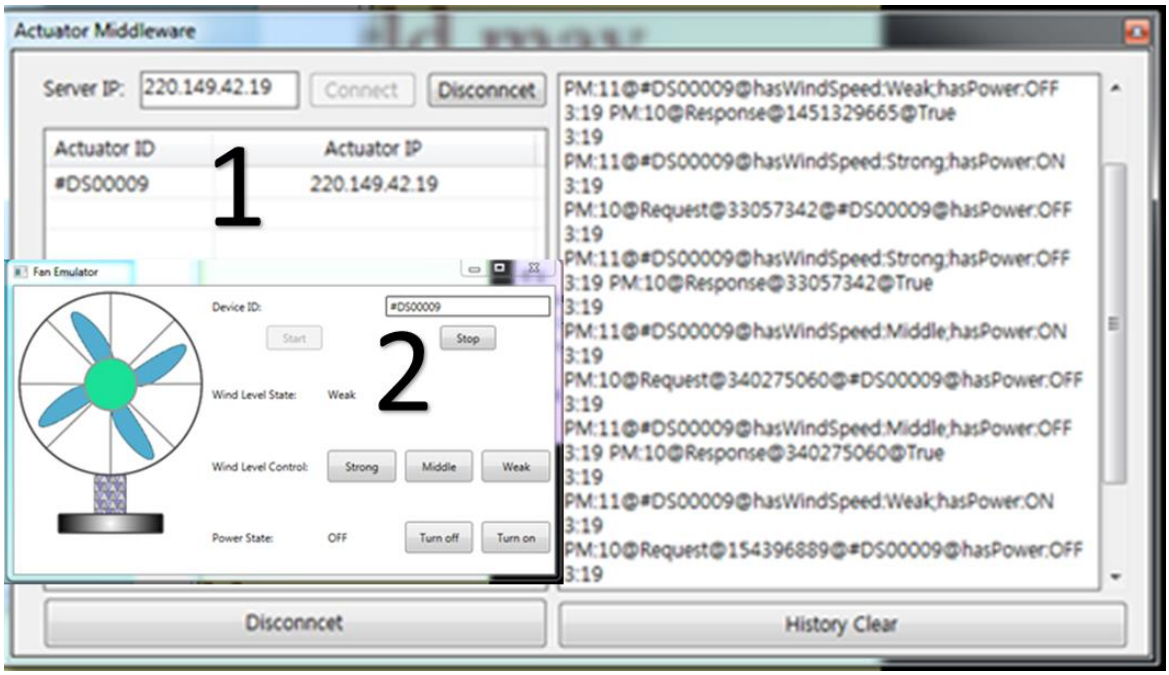

Figure 90 Fan control

Light Level Control can adjust light appliance illumination from 1 to 5. Power State can adjust power ON/OFF. Screen 1 shows the state of the light is ON with level 3. Screen 2 shows the command message generated by the application server based on the data collected from the illumination sensor. The control message asks the middleware to switch off the light. Screen 3 shows that the state of the light is turned off according to the control message sent by the application server. Comfortable indoor environment for normal human illumination requires maintaining the lux value 1000lx. If lux value is < 1000 the application server sends control message to the actuator middleware to switch on the light, whereas if the lux value is > or = 1000, the application server sends a control message to the actuator middleware to switch off the light.

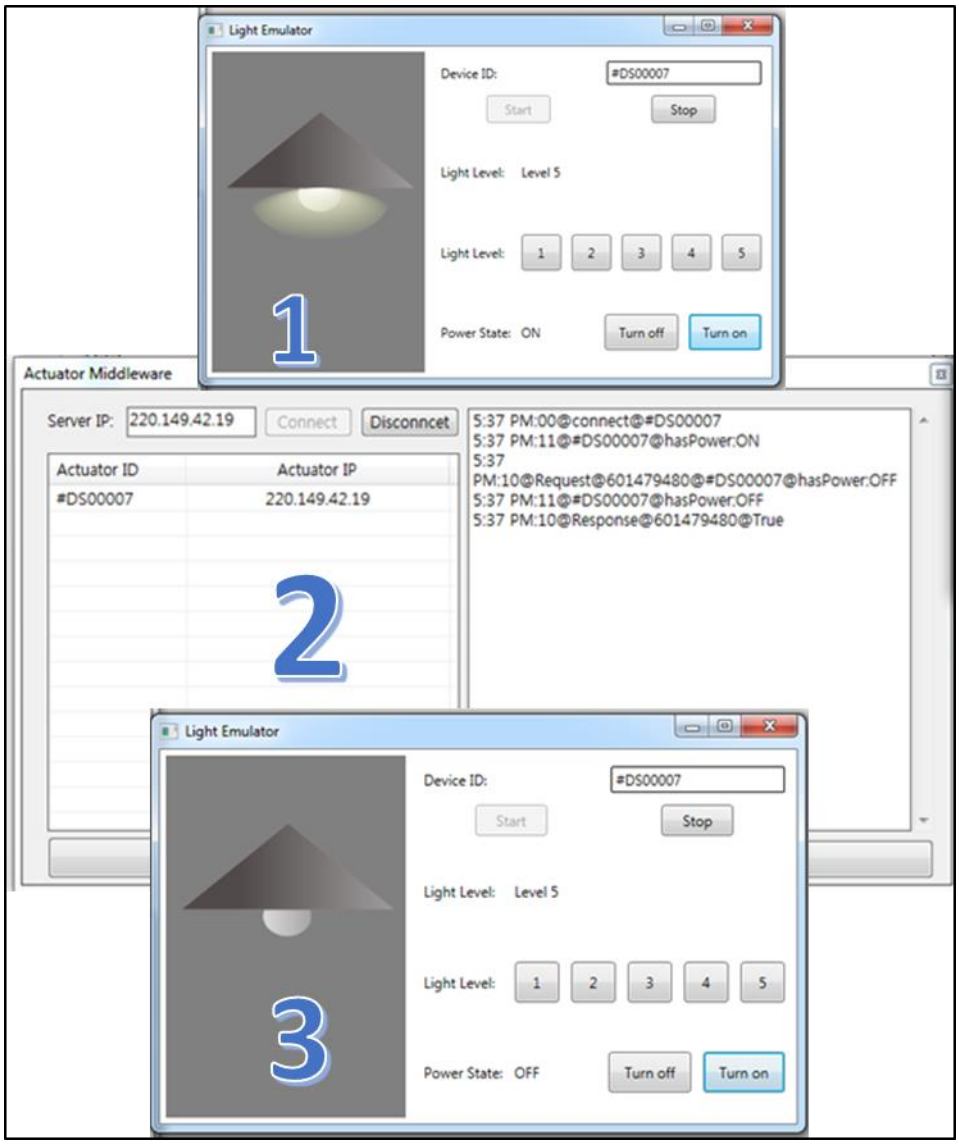

Figure 91 Light control

Figure 92 shows the control of air condition through the application server. The air con emulator offers functions for temperature adjustment, air flow adjustment, wind direction adjustment, ventilation adjustment, conservation arrangement and power adjustment. The upper part of the figure shows Air conditioning state, lower part shows the control area. Temp Regulation offers temperature adjustment and Wind Level offers Wind strength adjustment. Wind Direction offers Wind mode operation. Air Renewal offers ventilation mode. Open Order and Close Order specifies the scheduled time for air conditioning function. Power offers Air conditioning power

control. Based on the comfort index of the room shown in the previous figure, the comfort state is normal which means there is no need for air condition.

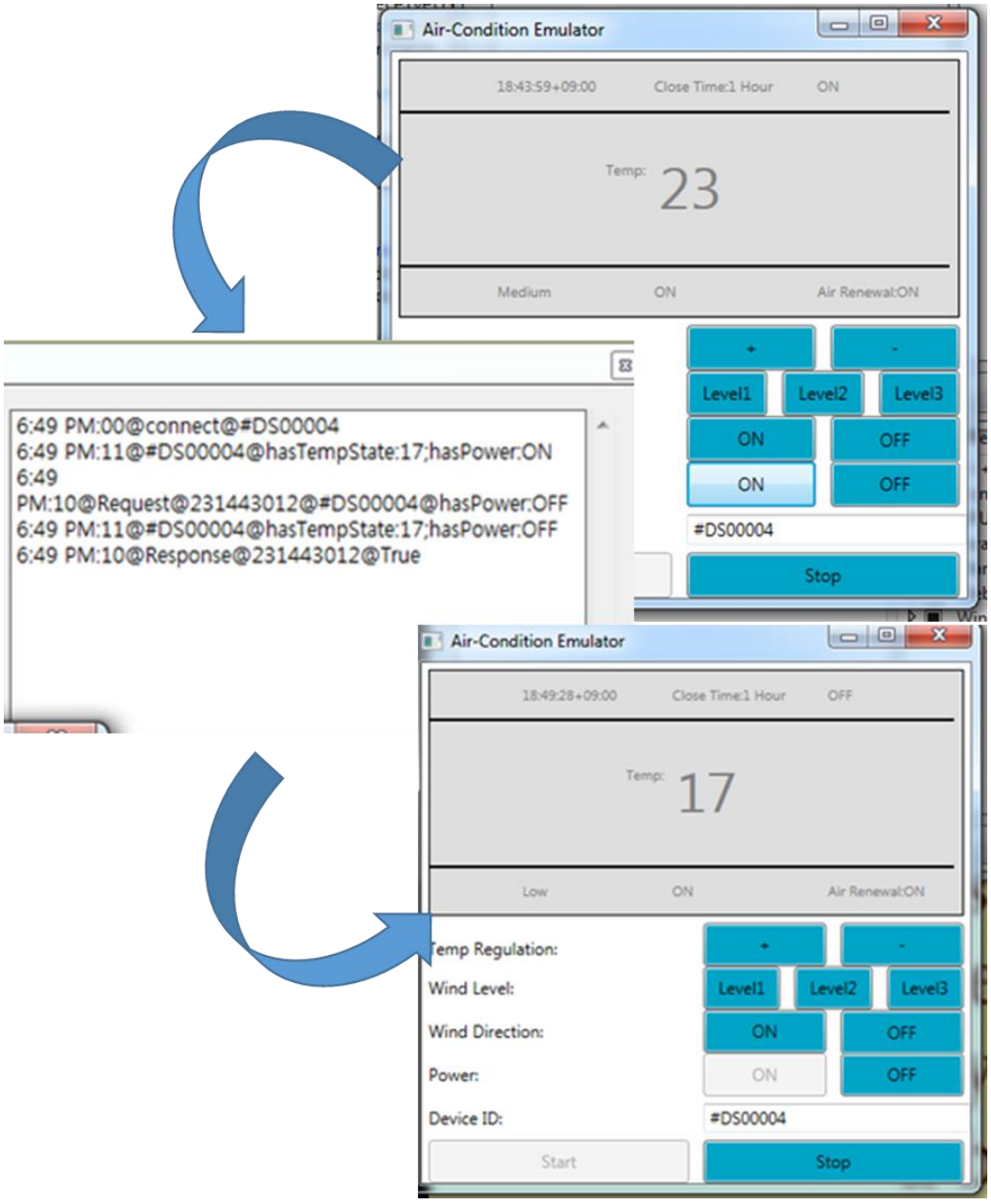

Figure 92 Aircon control

As the user connects the air con to the actuator middleware and sets the power state to ON, the application server sends a control command to the actuator middleware to set the power state to OFF. This process is shown in the screen shots here. Figure 93 shows the control of the boiler

through the application server. The boiler emulator offers functions like temperature adjustment, boiler mode adjustment and power control. Temperature can be adjusted from 20 to 80, the mode selection functions can specify the operation mode of boiler as water heater or heater, and adjusts power ON/OFF state. Based on the comfort index of the room shown in the previous figure, the comfort state is normal which means there is no need for switching on the boiler.

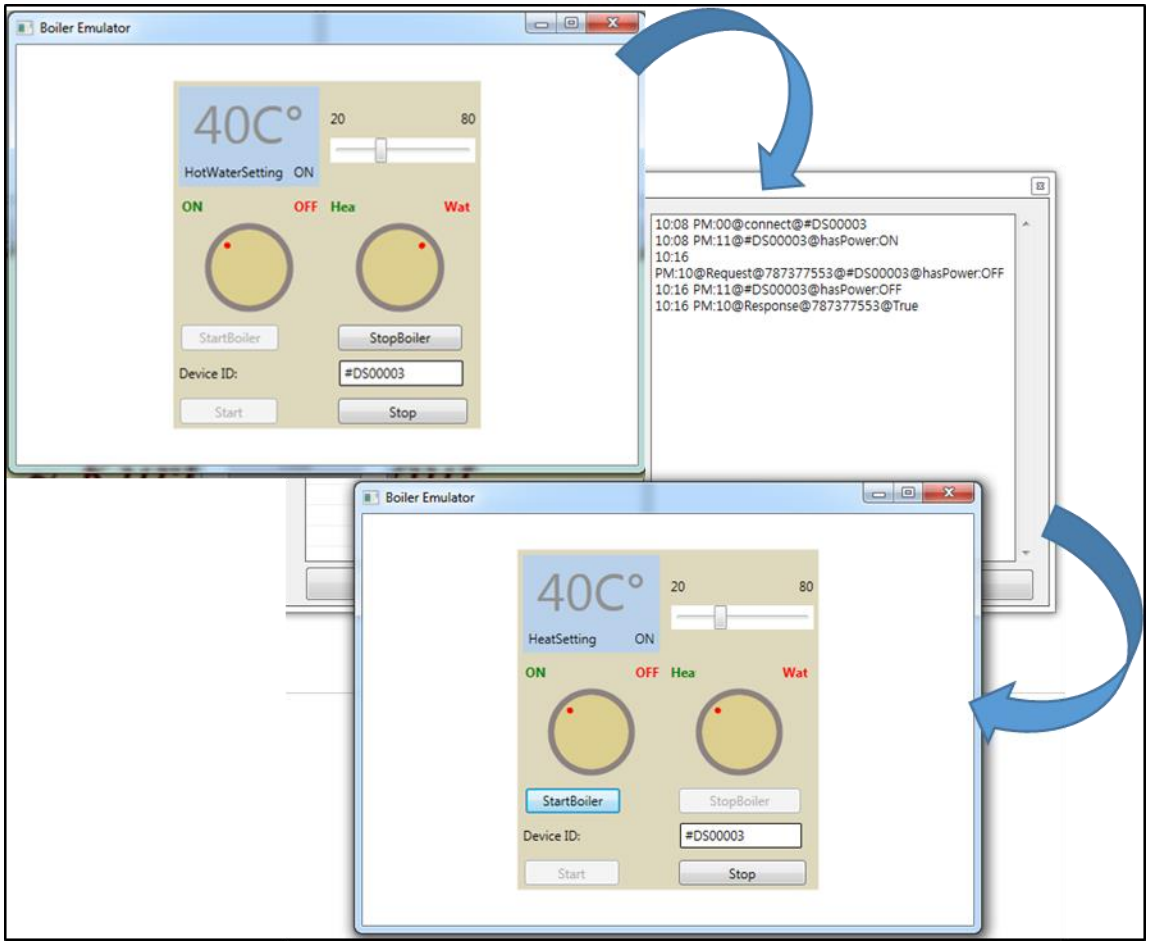

Figure 93 Boiler control

As the user connects the boiler to the actuator middleware and clicks the start button, the application server sends a command control to the actuator middleware to stop the boiler. And the boiler is set to stop through the actuator middleware. This process is shown in the screen shots here.

5.4.4 Development and Reasoning of Application server ontology

This section describes the development of the application server ontology as well as the reasoning performed on the ontology. Figure 94 shows the screen shot from protégé illustrating the classes and the individuals defined in the ontology. Map Service class represents the Uri and the code of the map that is registered in the GIS provider service provider. An Object in the ontology represents any device that is registered in the system along with its location information. Object Information class represents the objects registered in the application server ontology. Rule information class represents

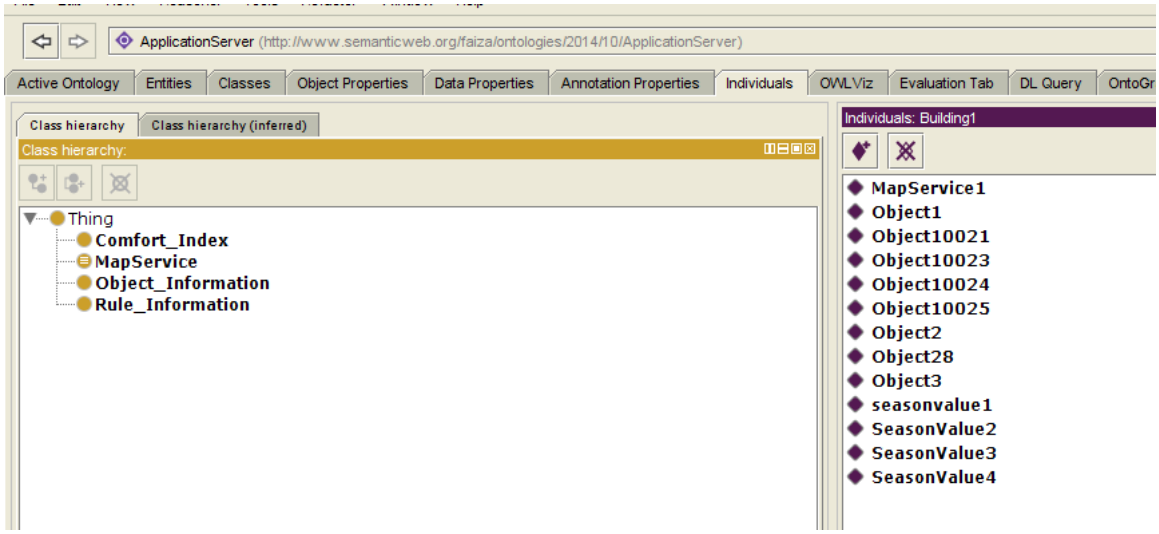

Figure 94 Application Server ontology

To calculate the required comfort index of a room, the application server first calculates the season at that time. The application server ontology calculated the season value based on the data given in table 5. It is calculated using Icl, M, and W value. Icl means the clothing insulation. It is the thermal insulation provided by clothing. M means the metabolic heat production. W means the effective mechanical work. In the ontology these are stored as attributes of the Rule Information class, and each season is represented by an instance of the Rule Information class. Table 5 shows the season name based on the values of these three factors.

Table 5 Season Data Calculation

Season Value	M_Value	W_Value	Icl_Value
Spring	1.2f	0.0f	1.0f
Summer	1.0f	0.0f	1.5f
Autumn	1.1f	0.5f	1.1f
Winter	1.4f	0.4f	1.0f

Figure 95 shows a screen shot from protégé that describes the inferencing results performed on the ontology. As shown in the figure the object property hasInformation is inferred by the reasoner, relating the objects in the ontology to their map service. This information can be queried by the user to inquire about the objects registered based on the map services in the ontology.

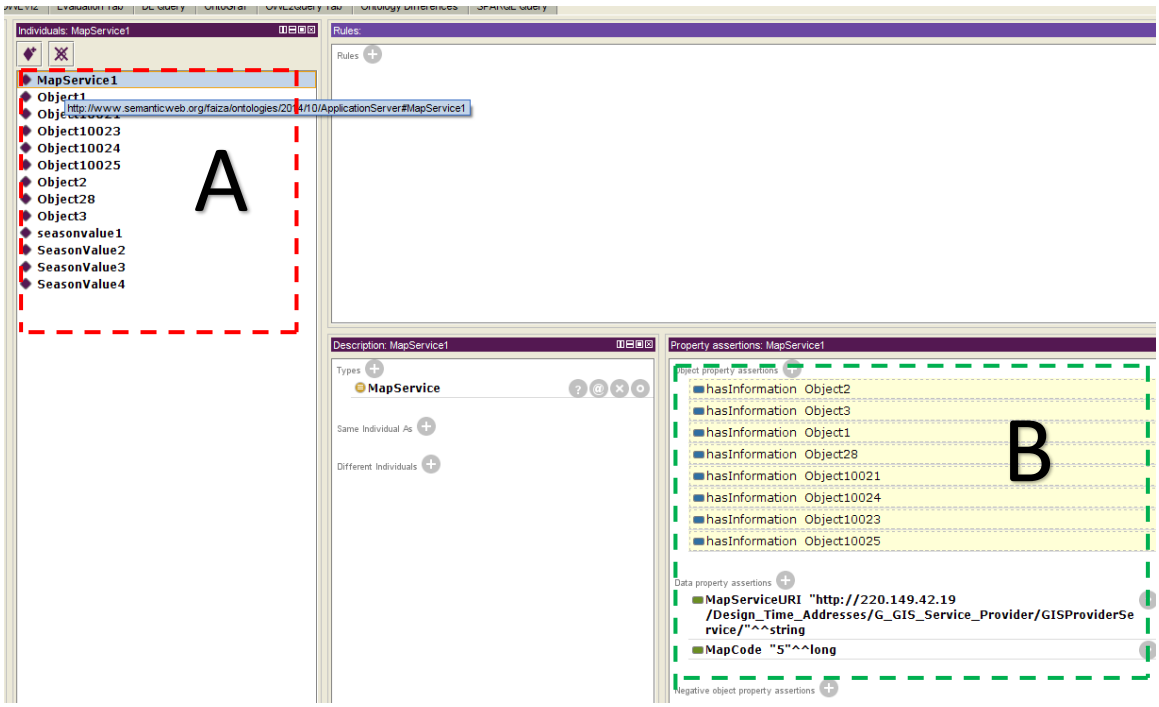

Figure 95 reasoning on application server ontology

5.5 Semantic Application Client

5.5.1 Implementation of Semantic Application Client

Figure 96 shows the execution screen for the client which is a web based application and acts as a simple visualization tools. It displays two controls to the user a textbox and a search button. It searches for available services in the semantic service registry ontology. Service registry returns the following service information Service name, IS WSDL, and Service Uri. The client displays the above information to the user in the form of a list. By clicking the access button the client displays the map data to the user. It displays the outdoor map view of the building registered.

The building that is registered in the ontology turns red with mouse hover. Once the user right clicks the building, a user menu item is displayed showing the floors that are registered in the building. When user selects a floor, the floor map is displayed in a separate window. It displays zoom in, zoom out, move right, move left, move up and move down controls Floor map displays the rooms that are registered in the floor. The objects registered in a room are displayed using ellipses. The blue ellipse indicated a sensor object whereas the yellow ellipse indicate an actuator object.

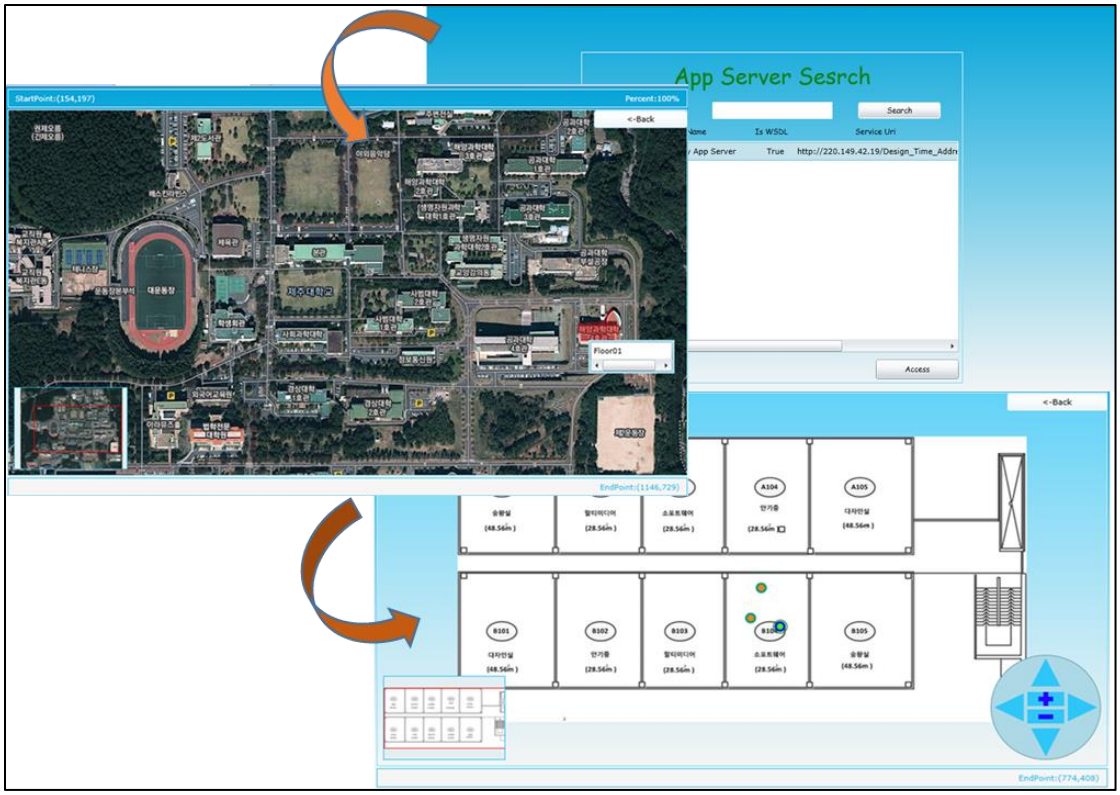

Figure 96 Application Client

Figure 97 shows the sensor data retrieved by the client. When the user clicks on the sensor object, a data view is displayed that shows:

- The state of the sensor. If the sensor is connected to the application it shows the state ON
- Current date time value
- The latest temperature value detected by the sensor
- The latest humidity and illumination value detected by the sensor
- The sensor name
- The sensor id
- And the sensor category list

Figure 97 Sensor Data View

The SPARQL queries used to retrieve the above data are discussed below. Figure 98 shows the SPARQL query for getting the sensor state from the sensor ontology. The figure displays the SARQL query tab from protégé which allows users to execute SPARQL queries on the ontologies developed in it.

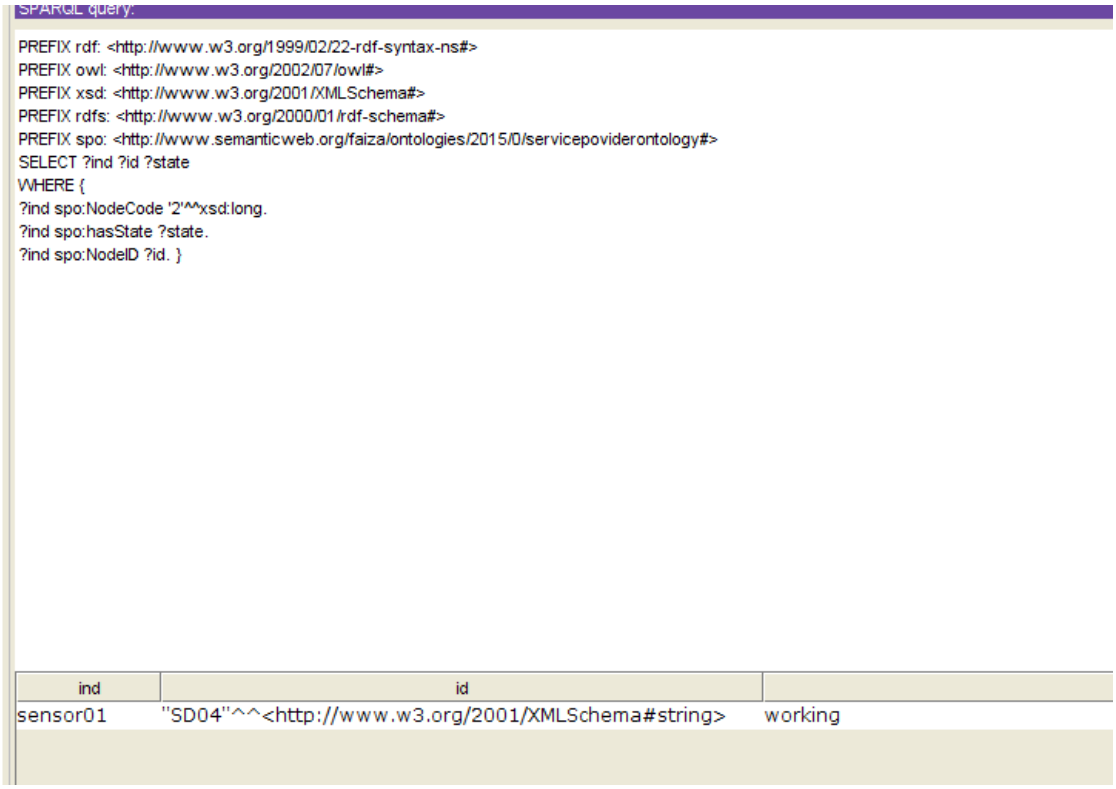

Figure 98 Get sensor state query

The first triple pattern in the query ?ind spo:NodeCode '2'^^xsd:long selects the sensor with the code value = 2. The second triple pattern *?ind spo:hasState ?state* selects sensor state based on the code of the sensor stored as a object property hasState in the ontology. The third triple pattern ?*ind spo:NodeID ?id* selects the id of sensor based on the code of the sensor. The table displays the results based on the query. It returns the instance, state and id of the sensor based on the sensor code given in the query.

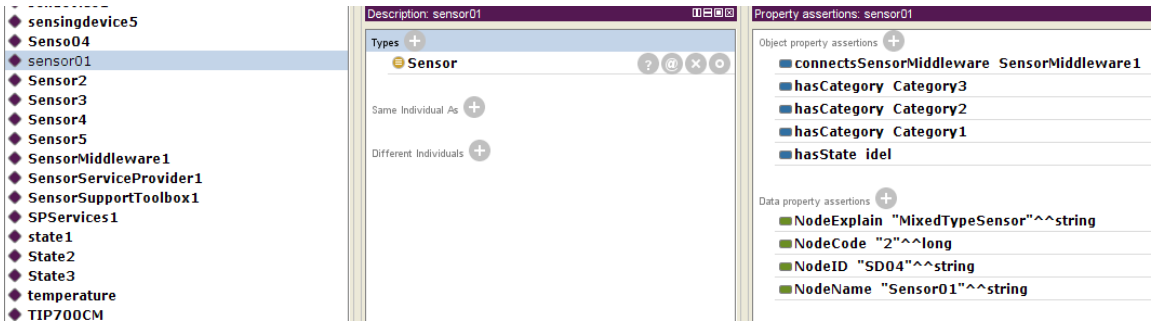

Figure 99 Sensor state before update query

Figure 99 shows a screen shot from protégé that illustrates the results before and after an update query shown in Figure 100. State of a sensor is stored in the ontology using an object property hasState. To update the state of the sensor the query shown in Figure 100 is run.

```
DELETE {
<http://www.semanticweb.org/faiza/ontologies/2015/0/serviceproviderontology/sensor01>
spo:hasState ?st.}
INSERT {
<http://www.semanticweb.org/faiza/ontologies/2015/0/serviceproviderontology/sensor01>
spo:hasState 'working'.}
WHERE {
<http://www.semanticweb.org/faiza/ontologies/2015/0/serviceproviderontology/sensor01>
spo:hasState ?st.}
```

Figure 100 Sensor state update query

Figure 100 shows a query from Sparl 1.1 Update query, which is an update language for RDF graphs. It uses a syntax derived from the SPARQL Query Language for RDF. Update operations are performed on a collection of graphs in a Graph Store.

Figure 101 Update query results

Operations are provided to update, create, and remove RDF graphs in a Graph Store. This query updates the sensor state in the sensor ontology graph. The first triple pattern deletes the hasState property associated with the specific sensor given in the Uri. The second triple pattern assigns a given value to the hasState property of the specified sensor.

SPARQL query:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX spo: <http://www.semanticweb.org/faiza/ontologies/2015/0/servicepoviderontology#>
SELECT ?category ?catgcode
WHERE {
?ind spo:NodeCode '2'^^xsd:long.
?ind spo:hasCategory ?category.
?category spo:CategoryCode ?catgcode. }
```

category	
Category2	"HUM00XXX"^^<http://www.w3.org/2001/XMLSchema#string>
Category3	"LUX00XXX"^^<http://www.w3.org/2001/XMLSchema#string>
Category1	"TMP00XXX"^^<http://www.w3.org/2001/XMLSchema#string>

Figure 102 Select Query

The WHERE clause identifies data in existing graphs, and creates bindings to be used by the template. Figure 100 shows the state of the sensor after executing the update query. As shown the state is changed from idle to working. Figure 102 shows the SPARQL query for retrieving the category name and the category code of a sensor. The first triple pattern selects a sensor based on its code whose value is specified as 2. The second triple pattern selects the hasCategory object property of a sensor selected in the previous triple. The third triple pattern selects the category code of the category instance selected in the second triple pattern. The results of this query are shown in the same figure.

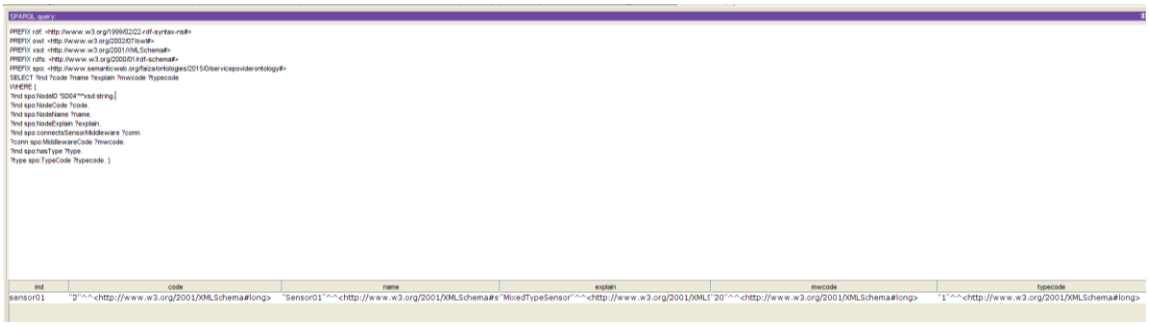

Figure 103 Select query

The SPARQL query for retrieving the sensor information is shown in Figure 103. The first triple pattern selects a sensor based on its id whose value is specified as SD01. The second triple pattern selects the code of a sensor, the third triple selects the name of a sensor, the fourth triple pattern selects the explain of a sensor, the fifth triple pattern selects the connectssensormiddleware property of a sensor, the sixth triple pattern selects the middleware code based on the previous triple. The seventh triple pattern selects the type of a sensor and the last triple pattern selects the code of the type selected in the previous triple. Selected in the previous triple.

Figure 104 shows the indoor comfort index and environment state for a selected (right-click1ed) room. This process is carried out by the control system in the application server. The comfort index calculation method is described in detail in the previous section. The figure displays the PMV value, the PMD and the ET value. Mamdani Fuzzy system based on Mamdani fuzzy concept is used to implement the fuzzy system. Based on the final value of the comfort index a control message is created by the control system to send to the actuator.

Figure 104 Room Comfort Index

Figure 105 shows the fan data retrieved by the client. When the user right clicks on an actuator object it requests the service provider to retrieve that data from the actuator ontology. The figure

shows the data for a fan that is registered in the ontology. It shows the Actuator ID, Actuator Name, Actuator Type, Actuator Model, and the power consumption of the actuator. In a separate list it displays the latest state of the actuator, it consists of the date time when the state is changed, the power state, and the wind speed at the time.

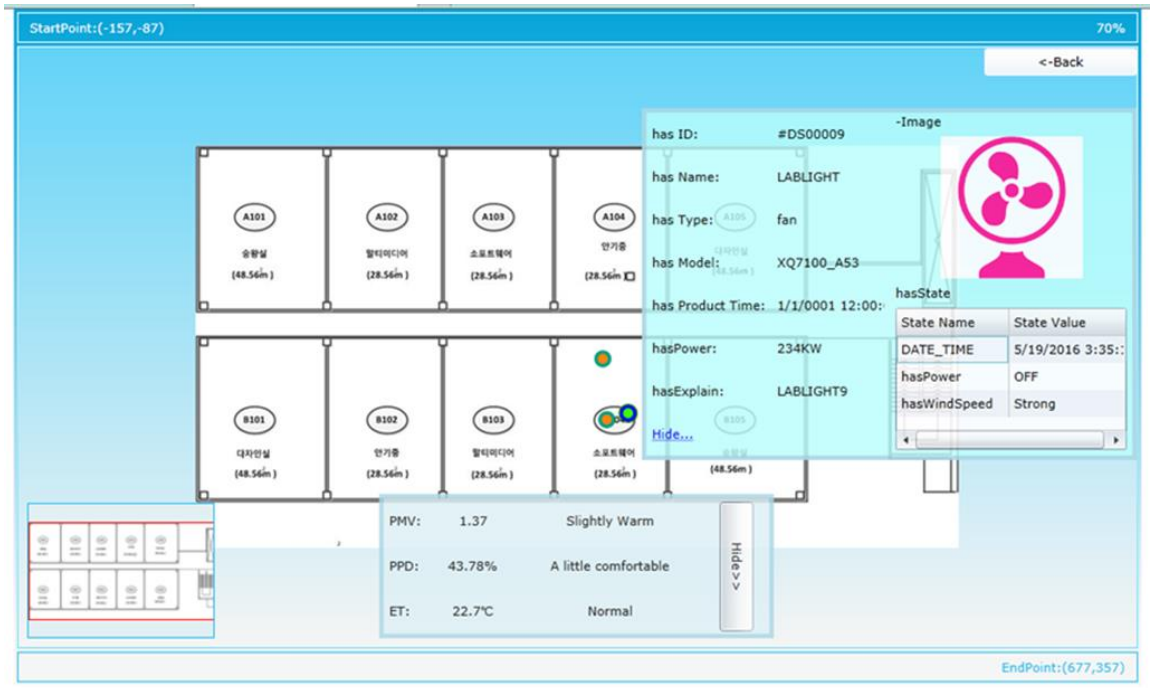

Figure 105 Fan data view

Figure 106 shows the light data retrieved by the client. When the user right clicks on an actuator object it requests the service provider to retrieve that data from the actuator ontology. The figure shows the data for the light that is registered in the ontology. It shows the Actuator ID, Actuator Name, Actuator Type, Actuator Model, and the power consumption of the actuator. In a separate list it displays the latest state of the actuator, it consists of the date time when the state is changed, the power state, and the wind speed at the time.

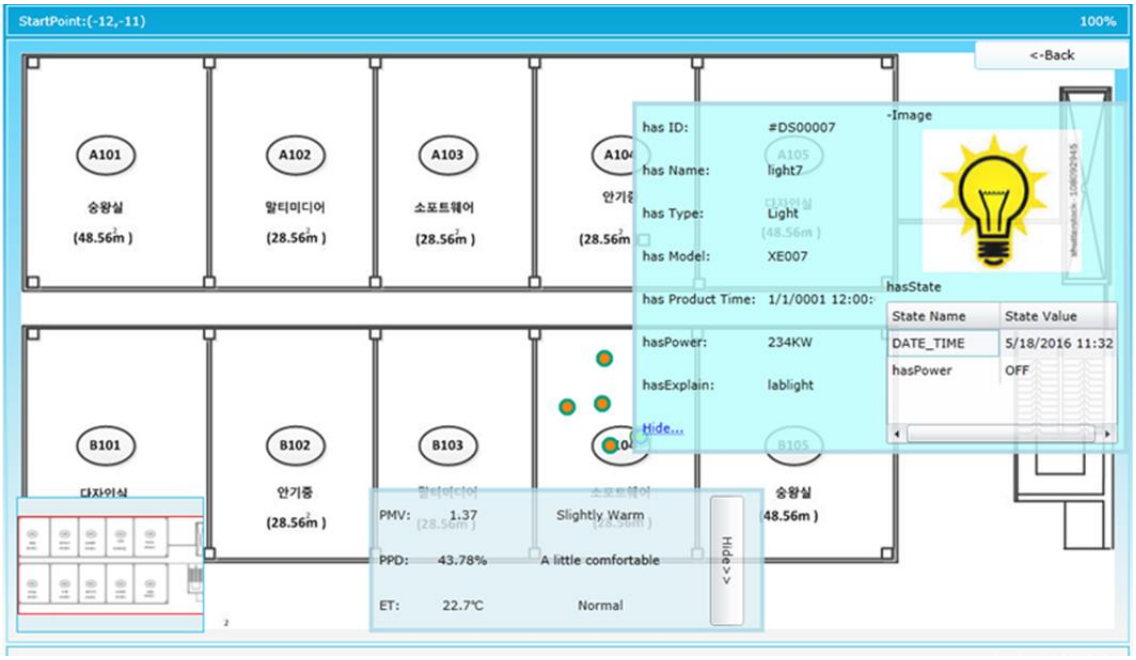

Figure 106 Light data view

Figure 107 shows the air conditioner data retrieved by the client. When the user right clicks on an actuator object it requests the service provider to retrieve that data from the actuator ontology. The figure shows the data for the air conditioner that is registered in the ontology. It shows the Actuator ID, Actuator Name, Actuator Type, Actuator Model, and the power consumption of the actuator. In a separate list it displays the latest state of the actuator, it consists of the date time when the state is changed, the power state, and the wind speed at the time.

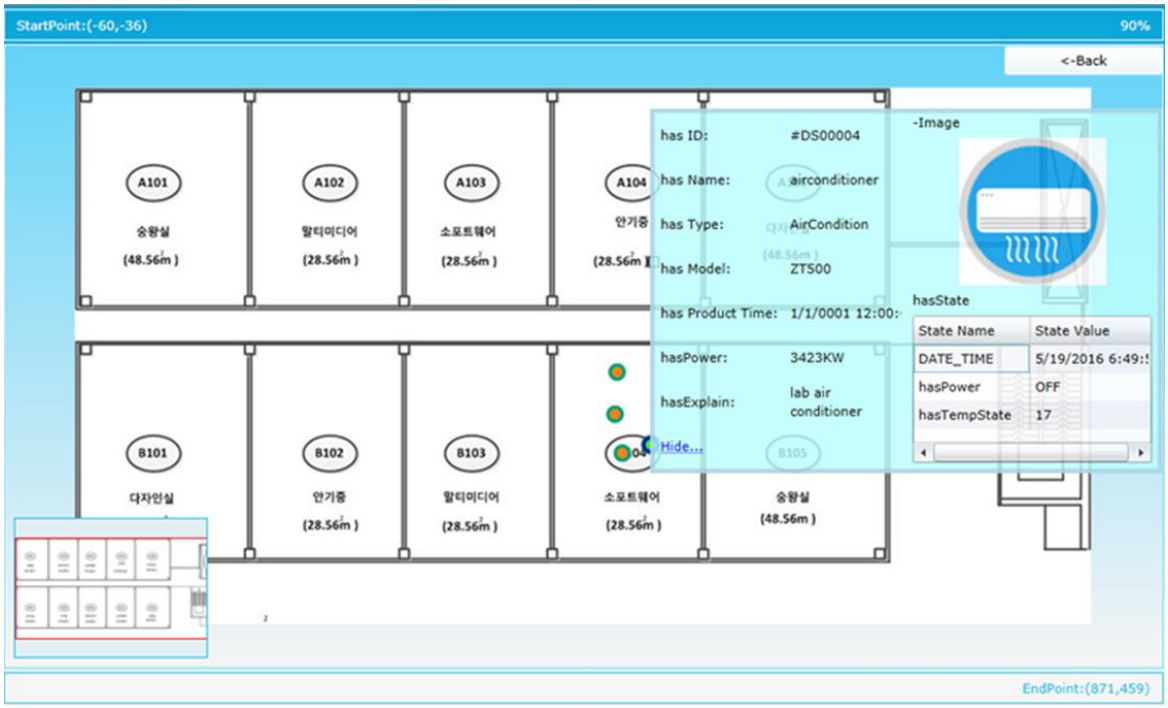

Figure 107 Aircon data view

Figure 108 shows the boiler data retrieved by the client. When the user right clicks on an actuator object it requests the service provider to retrieve that data from the actuator ontology. The figure shows the data for a boiler that is registered in the ontology. It shows the Actuator ID, Actuator Name, Actuator Type, Actuator Model, and the power consumption of the actuator. In a separate list it displays the latest state of the actuator, it consists of the date time when the state is changed, the power state, and the wind speed at the time.

Figure 108 Boiler data view

Query shown in Figure 109 a query from Sparql 1.1 Update query, which is an update language for RDF graphs. It uses a syntax derived from the SPARQL Query Language for RDF. Update operations are performed on a collection of graphs in a Graph Store. Operations are provided to update, create, and remove RDF graphs in a Graph Store. This query updates the actuator state in the sensor ontology graph. The first triple pattern deletes the hasState property associated with the specific actuator given in the Uri. The second triple pattern assigns a given value to the hasState property of the specified actuator. The WHERE clause identifies data in existing graphs, and creates bindings to be used by the template. Figure 110 displays a screenshot that shows the state of an actuator after the update query is executed. As shown the state is changed from offline to online as explained before.

```
DELETE {
<http://www.semanticweb.org/faiza/ontologies/2016/3/SemanticActuator/Actuator1>
ap:hasState ?st.}
INSERT {
<http://www.semanticweb.org/faiza/ontologies/2016/3/SemanticActuator/Actuator1>
ap:hasState ap:'online'.}
WHERE {
<http://www.semanticweb.org/faiza/ontologies/2016/3/SemanticActuator/Actuator1>
ap:hasState ?st.}
```

Figure 109 Actuator state update query

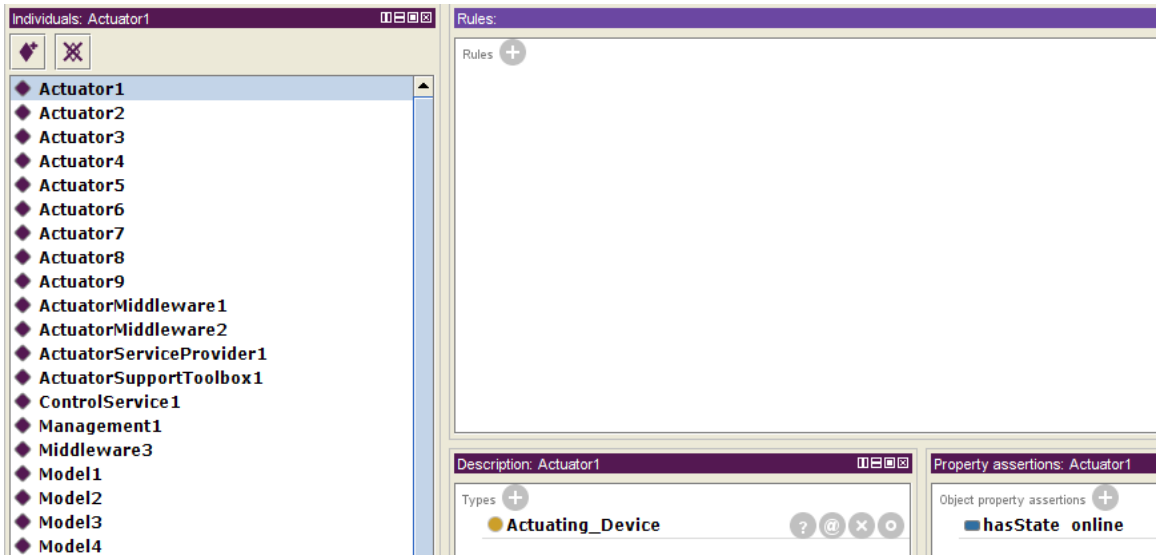

Figure 110 Update query results

Figure 111 shows a simple sparql select query to show the state of an actuator in the ontology. The state is stored as an object property hasState. The first triple pattern illustrates the variable for the value that is returned by the query. The second triple pattern states the id of the actuator whose state is needed, and the third triple pattern mentions the hasState property related to the actuator. The results of the query are also shown in the figure.

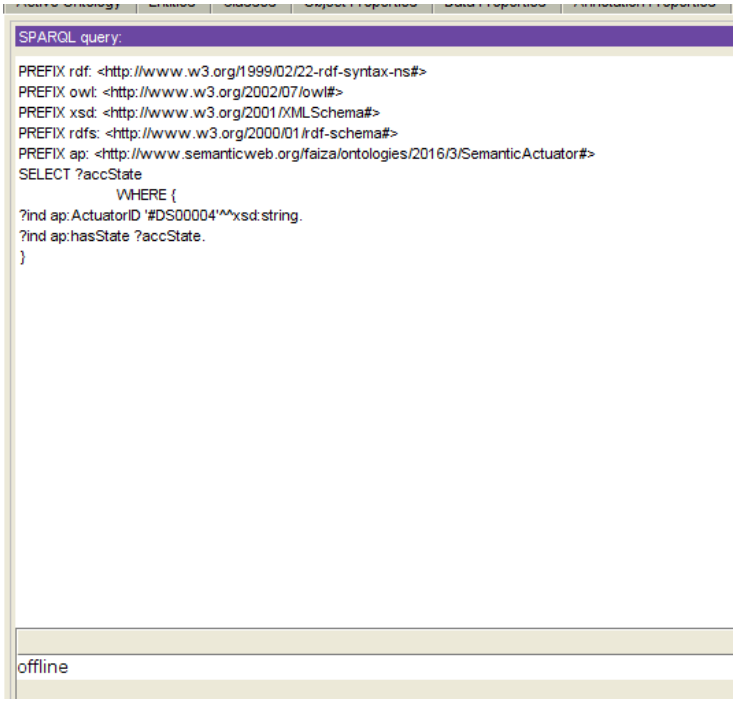

Figure 111 Select actuator state query

5.5.2 Performance Analysis of Sensor Provision based on Application Client

The following graphs shows the comparisons of SPARQL and SQL queries for retrieving sensor information and sensor state from the sensor ontology through the semantic sensor service provider module. The queries are compared for 10 iterations of retrieval. The results show that SPARQL queries take less time to retrieve and display the results to the client than the SQL query.

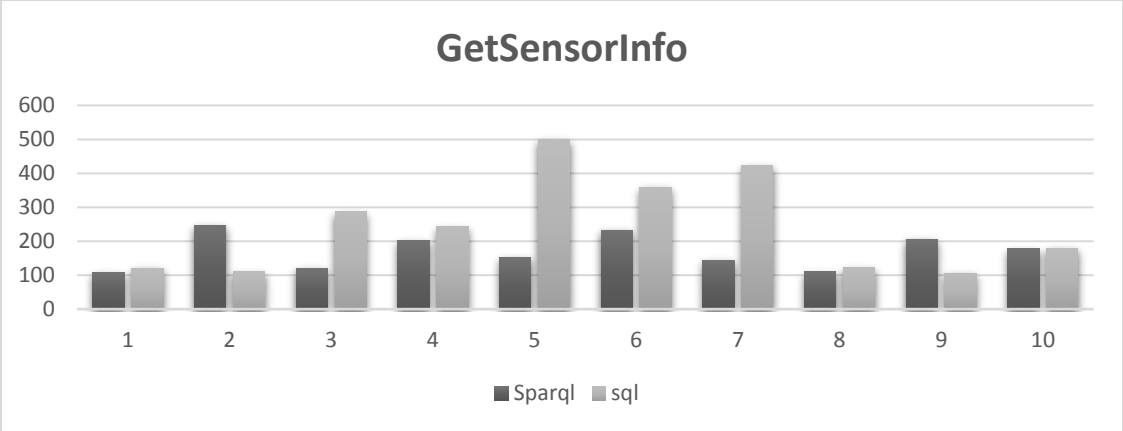

Figure 112 Query comparison for getting sensor info

Figure 112 shows the graph for comparing the SPARQL and SQL queries for retrieving sensor information. Application client displays the registered objects (sensors and actuators) to the user. When the user clicks an object, the client connects to the object's service provider and retrieves the information. The graph shows the time taken in milliseconds by SPARQL and SQL queries. The min time taken by SPARQL query is 106.5ms, the max time is 246.5, and the average time is 169.78. Whereas the mix, max and average time taken by the SQL query is 104.9, 498.3, and 244.42. The graph shown in Figure 113 illustrates the SPARQL and SQL query comparison for retrieving sensor state from the semantic sensor service provider repository. The comparison is based on the min, max, and average time taken in milliseconds to retrieve the sensor state. The time taken by SPARQL query is min 46.1ms, max 201.43, and average 86.29ms. The time taken by SQL query to retrieve sensor state is min 48.92ms, 225.16ms and average 114.92ms.

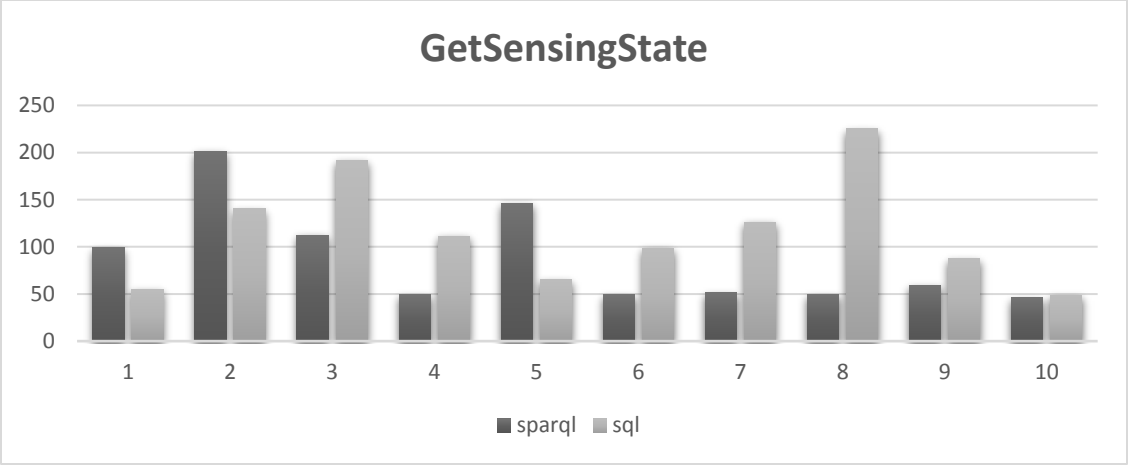

Figure 113 Query comparison for retrieving sensor state

5.5.3 Performance Analysis of Actuator Information Provision based on Semantic Application Client

The following graph shows the comparison results of SPARQL and SQL queries based on min, max and average time taken in milliseconds. The queries compared here are for retrieving actuator information and actuator state. Figure 114 displays the graph for comparing SPARQL and SQL queries for retrieving actuator data. The comparison is based on the min, max and average time taken in milliseconds for each query to retrieve the information. The min time taken by SPARQL query is 105.6ms, the max time taken is 225.9ms, and the average time taken is 162.9ms.

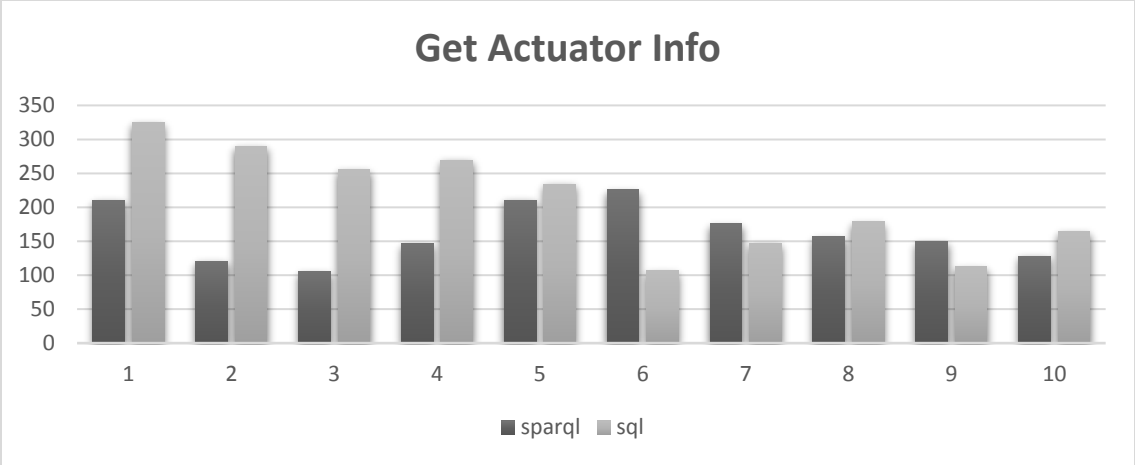

Figure 114 Query comparison for getting actuator information

Whereas the min, max and average time taken by the SQL query is 106.8ms, 324.2ms, and 208.1ms.The graph shown in Figure 115 displays the time taken in milliseconds for each query to retrieve the actuator state. The min time taken by SPARQL query is 88.23ms, the max time taken is 251.57ms, and the average time taken is 142.17ms, Whereas the min, max, and average time taken by SQL query is 102.3ms, 341.5ms, and 197.5ms.

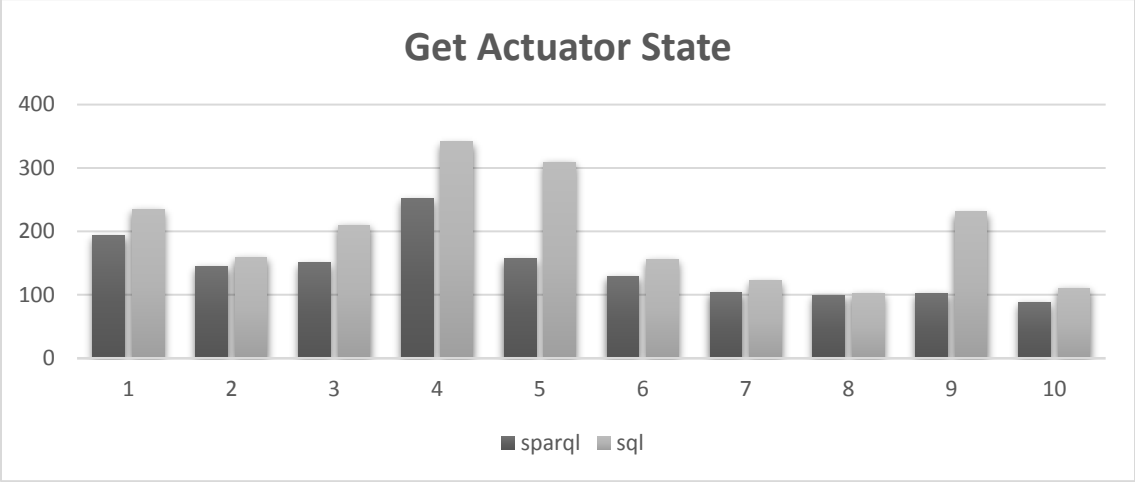

Figure 115 Query comparison for retrieving actuator state

Figure 116 displays the graph for comparing the time taken in milliseconds by the application client to retrieve the building information. 10 iteration have been compared for each query in terms of min, max, and average time taken. The min time taken by SPARQL query is 90ms, the max time

taken is 165ms, and the average time taken is 140.47ms, Whereas the min, max, and average time taken by SQL query is 142ms, 325ms, and 194.6ms.

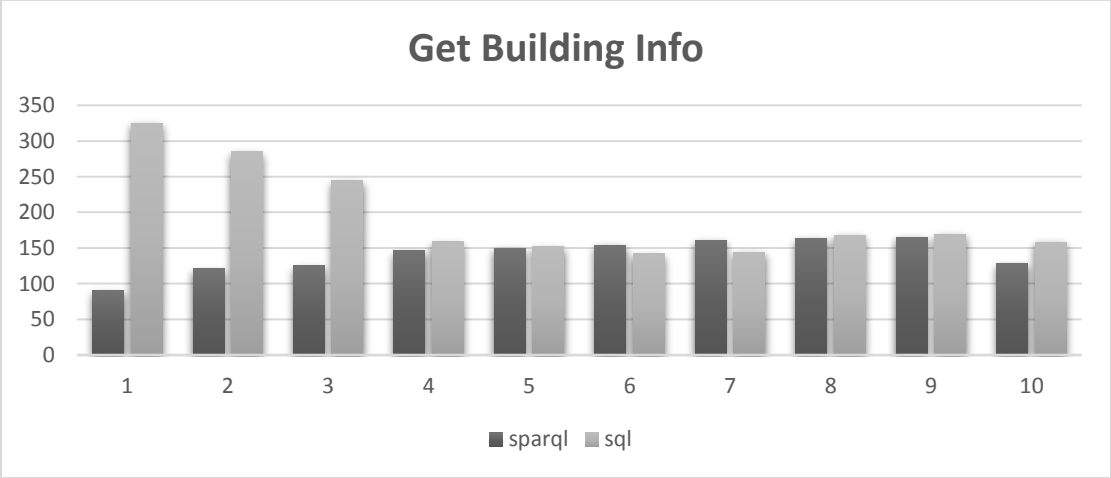

Figure 116 Query Comparison for getting building information

6 Conclusion

The problem discussed in this thesis is perhaps one of the most challenging task for IoT application developers. In this thesis we have proposed a solution for providing interoperability caused by the heterogeneity of IoT devices. We have used semantic technologies to overcome the issue of interoperability. We have developed a Semantic IoT system based on Support toolbox, that uses both semantic and database technologies to collect, store and provider environmental context information. It is built upon various service modules that collects data from sensors and actuators bind their location data and provider this information to the users.

We have taken a modular approach by building an ontology model for each module to represent its data. In the Semantic sensor module we have reused the SSN ontology. Reusing existing ontologies increases application interoperability both on syntactic and semantic level. Stakeholders using the same ontology are assumed to agree on the concepts used in the ontology. We have used SSN ontology to define basic definition of sensor, its properties and its observations. We have extended this ontology by adding additional attributes related to our system.

We did performance analysis of the system based on the SPARQL and SQL queries. The main queries that effect the system performance are adding a new resource to the ontologies, updating or deleting the resource, and to retrieve the resource information from the client layer. The time taken in milliseconds for these queries to execute has been calculated for 10 iterations of each query. The results of the analysis shows that the overall response of the SPARQL queries is better than the SQL queries.

Future work includes linking the RDF data by publishing the ontologies online. By publishing the ontology online it can be interlinked to existing ontologies and become more useful through semantic queries. Publishing an ontology means making it an accessible resource, both human and machine readable, with documentation with examples and with its license specified. Linked Data lies at the heart of what Semantic Web is all about: large scale integration of, and reasoning on,

data on the Web. It can help a person or machine can explore the web of data. With linked data, when you have some of it, you can find other, related, data. Our eventual goal is to publish the ontology online and link it to other similar ontologies. This can make the ontologies useful for people developing similar applications.

References

[1] S. De, P. Barnaghi, M. Bauer, and S. Meissner, "Service modelling for the Internet of Things," *2011 Fed. Conf. Comput. Sci. Inf. Syst.*, pp. 949–955, 2011.

[2] L. F. Sikos, "Introduction to the Semantic Web," in *Mastering Structured Data on the Semantic Web*, 2015, pp. 1–11.

[3] V. H. La and A. Cavalli, "Security Attacks and Solutions in V Ehicular a D Hoc N Etworks : a S Urvey," vol. 4, no. 2, pp. 1–20, 2014.

[4] E. Koivunen, M. R., & Miller, "W3C Semantic Web Activity," *W3C. Semantic Web Kick-Off in Finland*, 2001. [Online]. Available: http://www.w3.org/2001/12/semweb-fin/w3csw.

[5] E. Bash, *No Title No Title*, vol. 1. 2015.

[6] "Semantic Web - XML2000, slide 10." W3C.

[7] W3C, *World Wide Web Consortium (W3C) About the Consortium*. 2009.

[8] "XML and Semantic Web W3C Standards Timeline." 2012.

[9] M. Compton, P. Barnaghi, and L. Bermudez, "The SSN Ontology of the Semantic Sensor Networks Incubator Group," *J. Web …*, pp. 1–6, 2011.

[10] M. Botts, G. Percivall, C. Reed, and J. Davidson, "OGC (R) Sensor Web Enablement: Overview and High Level Architecture," *Lect. Notes Comput. Sci.*, vol. 4540, no. December, pp. 175–190, 2007.

[11] A. Katasonov and M. Palviainen, "Towards ontology-driven development of applications for smart environments," *2010 8th IEEE Int. Conf. Pervasive Comput. Commun. Work. (PERCOM Work.*, pp. 696–701, 2010.

[12] P. Barnaghi, W. Wang, C. Henson, and K. Taylor, "Semantics for the Internet of Things: early progress and back to the future," *Int. J. Semant. Web Inf. Syst.*, vol. 8, pp. 1–21, 2012.

[13] L. Atzori, A. Iera, and G. Morabito, "The Internet of Things : A survey," *Comput. Networks*, vol. 54, no. 15, pp. 2787–2805, 2010.

[14] C. Reed, M. Botts, J. Davidson, and G. Percivall, "Open Geospatial Consortium Sensor Web Enablement: Overview and High Level Architecture.," *IEEE Autotestcon*, pp. 372–380, 2007.

[15] M. Botts and A. Robin, "OpenGIS ® Sensor Model Language (SensorML) Implementation Specification," *Design*, p. 180, 2007.

[16] S. Avancha, C. Patel, and A. Joshi, "Ontology-driven adaptive sensor networks," in *Proceedings of MOBIQUITOUS 2004 - 1st Annual International Conference on Mobile and Ubiquitous Systems: Networking and Services*, 2004, pp. 194–202.

[17] C. A. Henson, J. K. Pschorr, A. P. Sheth, and K. Thirunarayan, "SemSOS: Semantic sensor observation service," in *2009 International Symposium on Collaborative Technologies and Systems, CTS 2009*, 2009, pp. 44–53.

[18] M. Compton, C. Henson, L. Lefort, H. Neuhaus, and A. Sheth, “A survey of the semantic specification of sensors,” in *CEUR Workshop Proceedings*, 2009, vol. 522, pp. 17–32.

[19] E. Saddikt, “A Universal Ontology for Sensor Networks Data 1,” *Work*, no. June, pp. 27–29, 2007.

[20] C. Villalonga, M. Bauer, V. Huang, J. Bernat, and P. Barnaghill, “Modeling of sensor data and context for the real world internet,” in *2010 8th IEEE International Conference on Pervasive Computing and Communications Workshops, PERCOM Workshops 2010*, 2010, pp. 1–6.

[21] A. Gyrard, “A Machine-to-Machine Architecture to Merge Semantic Sensor Measurements,” pp. 371–375.

[22] N. Shah, K. M. Chao, T. Zlamaniec, and A. Matei, “Ontology for home energy management domain,” in *Communications in Computer and Information Science*, 2011, vol. 167 CCIS, no. PART 2, pp. 337–347.

[23] A. Pease, I. Niles, and J. Li, “The Suggested Upper Merged Ontology: A Large Ontology for the Semantic Web and its Applications,” *Imagine*, vol. 28, pp. 7–10, 2002.

[24] M. Ryu, J. Kim, and J. Yun, “Integrated semantics service platform for the Internet of Things: a case study of a smart office.,” *Sensors (Basel).*, vol. 15, no. 1, pp. 2137–60, Jan. 2015.

[25] H. Patni, C. Henson, and A. Sheth, “Linked sensor data,” in *2010 International Symposium on Collaborative Technologies and Systems, CTS 2010*, 2010, pp. 362–370.

[26] K. Center.org, “Project Descriptions,” *Environment*, no. August. 2005.

[27] A. Gómez-Pérez, “SemsorGrid4Env FP7-ICT-223913.” .

[28] W3C, “Semantic Sensor Network Ontology.” 2014.

[29] A. Bröring, J. Echterhoff, S. Jirka, I. Simonis, T. Everding, C. Stasch, S. Liang, and R. Lemmens, *New generation Sensor Web Enablement*, vol. 11, no. 3. 2011.

[30] Open Geospatial Consortium, “Sensor Web Enablement DWG.” pp. 1–2, 2015.

[31] Marek Obitko, Faculty of Electrical Engineering, Czech Technical University in Prague, “Philosophical Roots - Introduction to ontologies and semantic web - tutorial.” .

[32] E. Sirin, B. Parsia, and B. Cuenca, “Pellet : A Practical OWL-DL Reasoner.”