Uzair Ahmed Zafar

Agents Driven Smart Sensors

Master's Thesis in Information Technology

October 31, 2017

University of Jyväskylä

Faculty of Information Technology

Author : Uzair Ahmed Zafar

Contact Information : uzair.az@gmail.com

Supervisor(s): Vagan Terziyan, Oleksiy Khriyenko, Michael Cochez

Title: Agent Driven Smart Sensors

Työn nimi: Agenttiohjautuva älykäs anturi

Project: Master's Thesis

Study line: Web Intelligence and Service Engineering, Faculty of Information Technology (Department of Mathematical Information Technology)

Page count: 34

Abstract:

Any physical area (like schools, home, hospitals etc.) that uses either mobile devices, sensors, embedded systems or computers to gather information from the users and the environment and eventually, adapt according to the new information gained. [1] Smart spaces compromises of heterogeneous hardware which often leads to the issues of interoperability. One way of reducing the heterogeneity between the sensors is to introduce the semantic interface as sensors default interface. Semantic Web provides a common interface and format for data representation so that one can decrease the heterogeneity of data and increase data reusability. [2] With the Smart Spaces, it is important to use only the hardware's that can achieve the tasks efficiently and are reliable and durable. When it comes to physical components of Internet of things, there are many different vendors that provide their products and cases often arises where a Smart Space will actually consist of products from different vendors. This is where the need of interoperability comes into story. It plays an important role in integrating the critical services, hardware and applications [3]

The aim of this thesis is to present a solution on how that will decrease the heterogeneity among various devices used in these smart spaces and at the same time, present some example of already existing different architectural workflows that currently exists in real world, that handles the (same) scenario in real world.

One of the concept in accomplishing this is presenting every resource with an agent [4]. We discuss on how we can collaborate agents (software/hardware) and sensors to work side by side. The open source JADE framework [5] provides distributed multi agent platform where agents can communicate and perform task. We aim on providing the agents with the ability to register sensors, retrieve the information from them and act accordingly if they need to. One framework that provides the platform for registering and using the sensors through the web is 52North [6]. It presents the sensors as a web services interface. This thesis discuss the implementation of a service, which follows semantic web, link and allow the agents in JADE platform to communicate and invoke tasks from the sensors registered in 52North.

Keywords: Agents, Smart sensors, Semantic Sensors, semantic web, heterogeneous hardware, interoperability, data reusability, distributed platforms

The Term List

AI Artificial Intelligence
FIPA Foundation for Intelligent Physical Agents
ICT Information and Communication Technology
IoT Internet of Things
OWL Web Ontology Language
RDF Resource Description Framework
RDFS Resource Description Framework Schema
UI User Interface
W3C World Wide Web Consortium
WWW World Wide Web
XML Extensible Mark-up Language

Contents

1. Introduction

Research Question:

"How can the concepts from the semantic web, be used to facilitate the communication between heterogeneous sensors and agents?"

"Semantic web is a web of data" [9]. It annotates the data available on the internet so that not only can it be in human readable form, but also the machine-readable form. It eventually promotes and supports common data formats and trustable data exchange protocols [9].

Any environment containing embedded sensors, appliances, computers or mobile devices that facilitates people to have remote access to their functionalities, forms a smart space. This can be in a home, offices, hospitals, schools, malls etc. Sensors are devices that detect/measure the conditions in the physical world. [10]

Agents are either hardware or software components, that can act autonomously to accomplish tasks on behalf of their users.

As talked briefly in the abstract, the trend of smart spaces is on the rise recently. With the integration of products manufactured by different vendors, in order to establish efficient communications, the protocols required for translations become complex and the translating hardware becomes costly. Besides the cost and integration of these translation gadgets, there are other limitations. For instance, like updating these hardware to accommodate newer agents (or sensors) after some interval of time.

There are several different agent platforms where multiple agents communicate and collaborate to achieve either single or different goals. Quite often, these agents must use the facilities of sensors to gather data from their surroundings. Normally, the agents have embedded sensors but at times, when they require the services of different sensors, they invoke the web services of those sensors. This is where constraints and limitations kick in. There are several different manufactures for sensors and agents, and hence not all of them are compatible with each other. In some other cases, users must pay extra to obtain interface that is compatible with theirs. Hence, there is a good amount of heterogeneity and less interoperability between these different manufactures. Due to this, the facilities exposed by these manufactures can't be utilized to their fullest.

In this work, we look at a service, that works on reducing the heterogeneity factor and increase the interoperability between devices of different brands. Utilizing the standards of semantic web, the service provides a semantic interface for communication between agents and sensors. For the domain of this thesis, service is only limited to one platform for both sensors and agents. These (open source) platform will be further explained under the sub-

heading of "Workflow". There will also be a facility where third party companies can register their semantic interface for their functionalities so that their exposed agents (or sensor) services can be utilized but the existing ones.

Example Scenario:

Let's take an example to further simplify the problem statement. Suppose there is advanced multinational auto manufacturing company. In a car manufacturing life cycle, there are several steps that are needed to be performed to manufacture one car. Some of these steps might include:

- Building engine
- Installing engine
- Installing hoods
- Installing chassis
- Customizing interior
- Installing interior
- Wheel adjustment/alignment

Now a day, most of the work done on the assembly line are done by robotic parts (robotic arms, etc.) and each of these parts have an agent assigned to it, to represent it in the network and provide an interface between it and the human and other agents in the factory.

These robotic parts (or hardware agents) are usually manufactured by different vendor who may expertise in producing parts for one specific assembly line part. The companies need to invest money in either hiring more staff to manually check and keep the assembly line active and consistently moving car from one step to another, or buying programs (or hiring developers to build and maintain software) to autonomously check and transfer the car to the next allocated step.

The problem with later part is when the company purchase a new part and then adding the interface for that to the existing software can be expensive. Similarly, same is the case if the factory replaces decides to replace some part, then they need to do recreate the interface for the new part.

To the above-mentioned solution, there might exist more but again, they don't prove to be cost efficient and expandable. Usually, the vendors supplying the parts tend to incorporate the factories into buying their provided official interface layers and services for their products so that they can charge them in the form of maintenance and services fees.

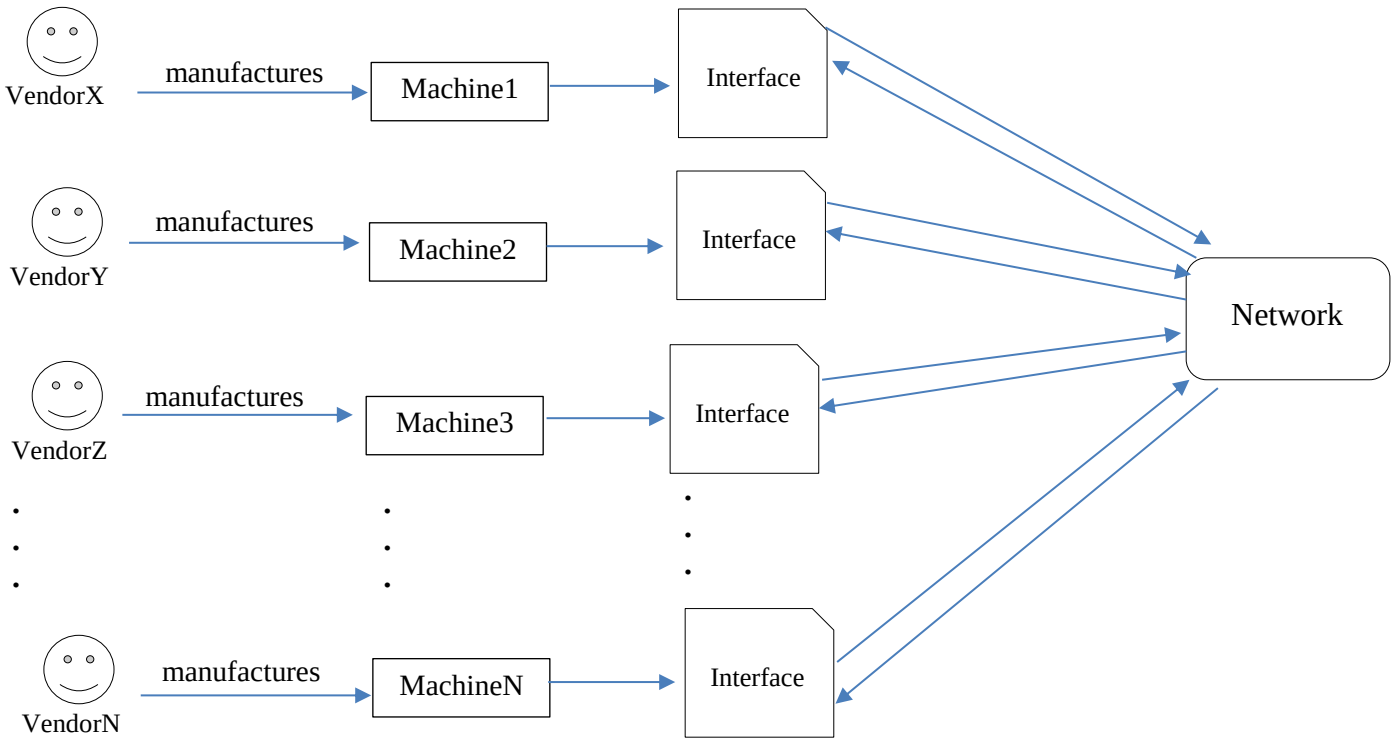

Fig 1 General layout example for existing model

This thesis discusses a cost effective, expandable and dynamic alternate to availing the interfaces provided by using the semantic standards. The goal is to provide one big interface layer that provides communication between the network and the different machines (agents/sensors) without the need for the network to know the interface protocol and the rules of the different machines. Instead, the machines just register (informs) all their provided functionality and they are automatically stored with the interface. When a network needs some specific functionality then they just use one same method, and the interface then makes sense of what method of which machine is to be called and what parameters are needed to be passed to it.

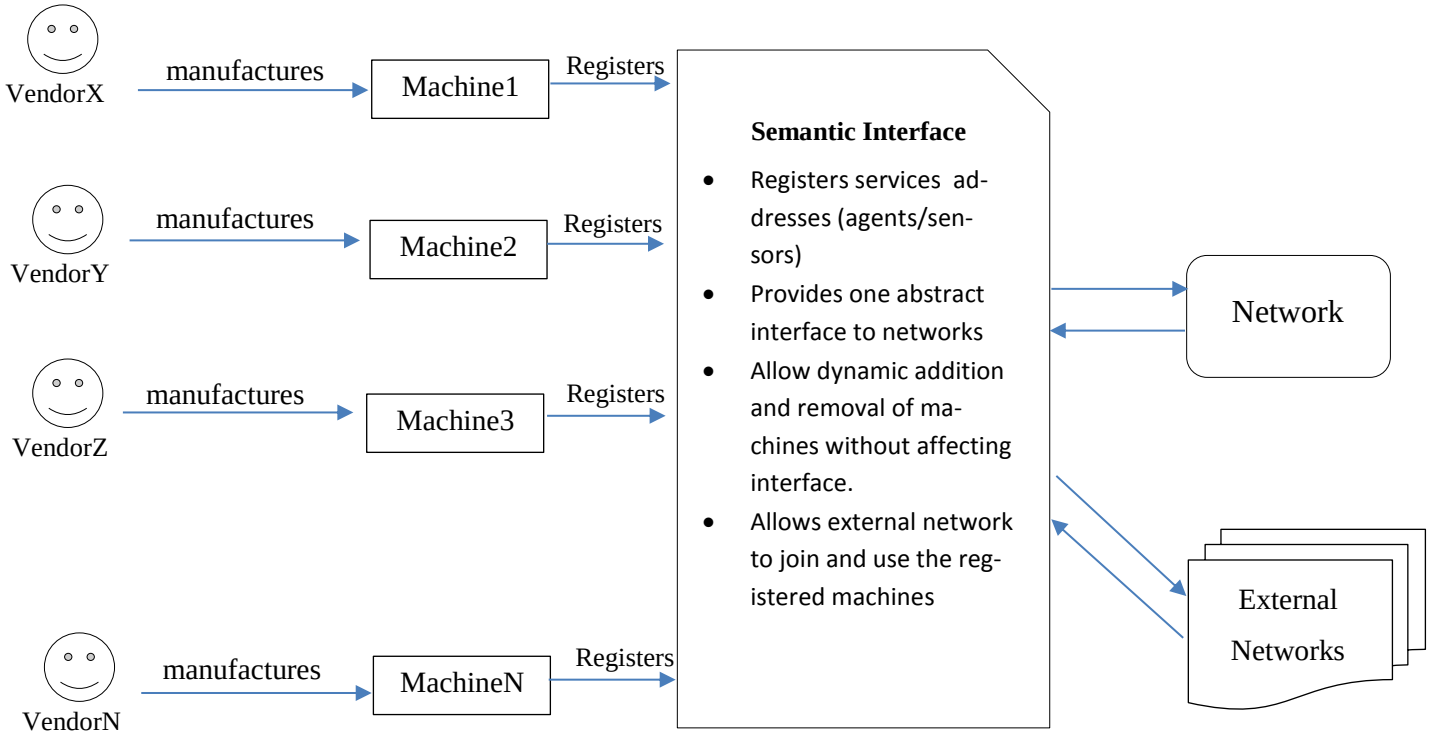

Fig 2 Model with the Proposed Semantic Interface

In this thesis we start discussing the problem domain by examining some architectural scenarios that are most commonly practiced and exist in real working environments. In order to obtain a better foundation and understanding to the basis of the problem, we go to the chapter two. Then, chapter three talks in detail first, about the proposed solution and then we look at it from semantic point of you. Chapter three also talks about the third party services that we avail to build up our interface. In chapter four, the paper briefly describes on how this proposed service can be integrated to the existing architectures to provide them a semantic interface.

2. Introduction to Existing Architectural Practices

Let's start by considering the example of a Hospital yard [fig 3] to compare the existing architectural practices that are in use in various SmartSpaces. Let's start by a basic scenario and assume that there exists a hospital yard where there are several patient beds. And to each patient, lies a series of small machines that are taking the crucial readings like patient blood pressure level or sugar levels in a patient. These machines senses these reading from the patient's body and then displays it somewhere for the nurse to observe and take desired actions if the readings are not in the normal range.

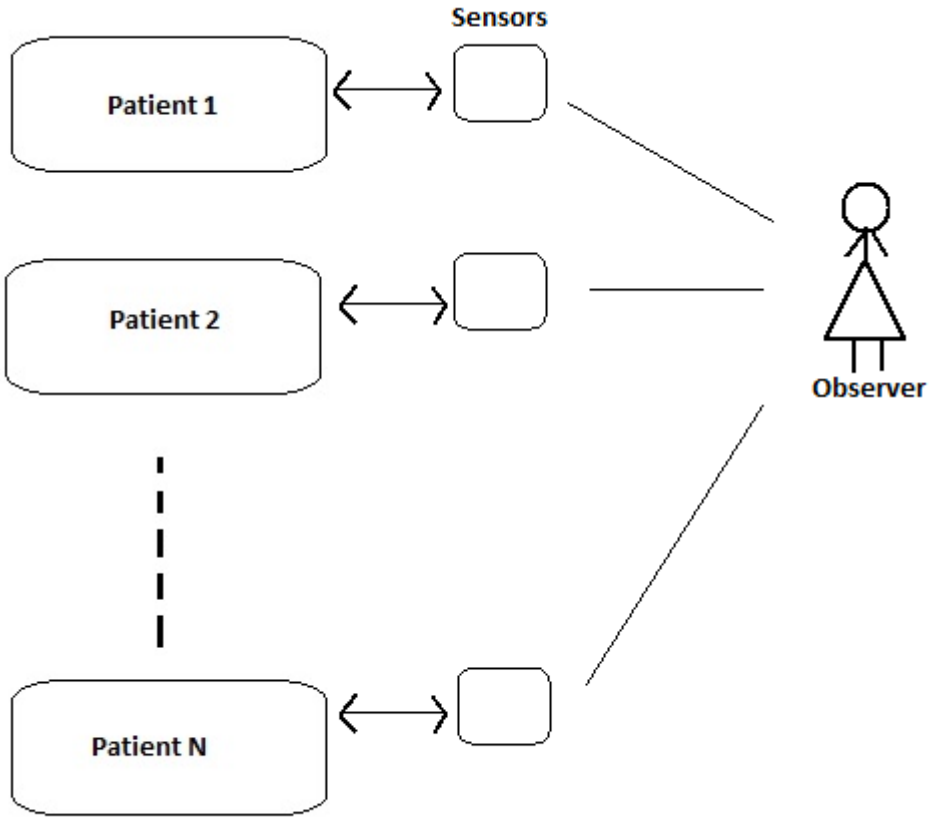

Fig 3 Hospital Yard, Nurse Attending to admitees

The scenario discussed just now, looks pretty good at the first glance but there can be several cons in it. For example, the cost for maintaining staff for twenty-four hours every day will be high. Also, there is always the possibility of a human error occurring that might result in wrong dosage to the wrong patient.

To cut costs, and built a more reliable system, the information that is obtained from the sensors and displayed for the hospital staff to take action, can actually be sent to some other machine that knows the meaning of the information it gets, and can operate accordingly by taking the desired action. In this case, for example, giving the patient the desired dosage of some medicine [fig 4].

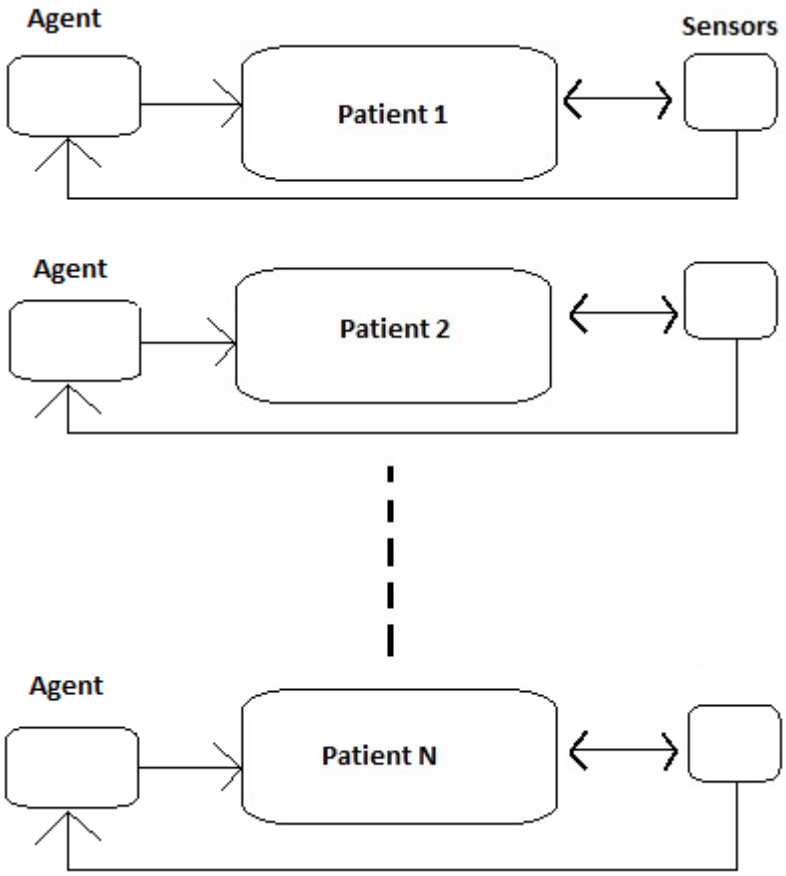

Fig 4 A simple Agent sensor based system.

For example, if there are two patients, who have different disease. Let us assume that patient 1 has diabetes and patient 2 has cancer. Now the sensor with patient 1, will be measuring the

blood sugar level of the patient and then these observations will be passed to the agent machine that will be assigned to patient 1. Depending on the reading, the agent can interpret and decide if the drip attached to the patient needs to start injecting the medicine into the patient's body. Similarly, the agent will observe the next few reading and when the blood sugar level are normal, it can stop injecting the medicine.

Similarly, with the patient 2, the same process will follow but the sensor will be different than the sensor for the patient 1. This sensor will be designed specifically to take reading necessary for chemotherapy. The agent, in the same way, will be different than the agent of patient 1. This agent will have the knowledge on how to interpret the reading obtained from the chemotherapy sensors, and then make decisions accordingly.

So, the agent and sensor might be different depending on the conditions of the patients and their requirements. But they will all follow the same protocols and life cycle.

Now, the scenario discussed later, if compared to the one discussed before that, has its advantages and disadvantages. It is more reliable as compared to scenario one and it cuts cost by reducing the number of staff. But there also exists architectural structures that are further efficient than the one discussed before.

The agents in the above case, needs to have the knowledge on processing the information being passed to them to take the desired actions. The costs of building an intelligent agent machine to interpret and act according to the data it receives, can be high. It also happens that the different agent machines (or sensors) may not be able to interoperate with the sensors (or agents) of different manufactures or the new models of the same vendors. The agent machines might have limited support for updates and with time, one may need to purchase the newer model of the agent machine as well only so that it can work along the sensor. One other option possible is to buy the updates for the agents to make them interoperable with the new model of the sensor.

One way to cut costs in this case, is to not to manufacture the agents to interpret information and act accordingly, but built them in such a way that they just need to act based on the instruction they receive. This is where cloud computing plays a role. The data collected by the sensors, can be sent to a central server. The agent, in the meanwhile, will know the ID of the sensor whose information it needs to act. So, the sensor will just invoke the service on the cloud and the service can obtain the information from the server, process it and just return the action that the agent should perform. [fig 5]

In this design, even if one of the agent machine or the sensor, is changed, there would not be any use of buying the other component that supports the new component. But simply, the service hosted on cloud will be provided information about the newer component and it can then be updated to interpret the data from the newer component. The cost of maintaining and updating the service is significantly less than the cost of updating and maintaining the whole machine. So, in this way, we can further reduce the running costs.

So, for example, keeping in the context of the hospital yard scenario, in this case, the agents (drips/infusion pumps/inhibitors) just register themselves to the service with an ID and the information on what data they are waiting for to operate on (sensor ID). The sensors, in the meantime, uploads data streams on the cloud. The service in return, checks the data streams that are coming to the cloud from the various sensors and retrieves the data for sensor IDs required. After that, the service simply forwards it to the respective agents which then interprets this data and perform actions accordingly.

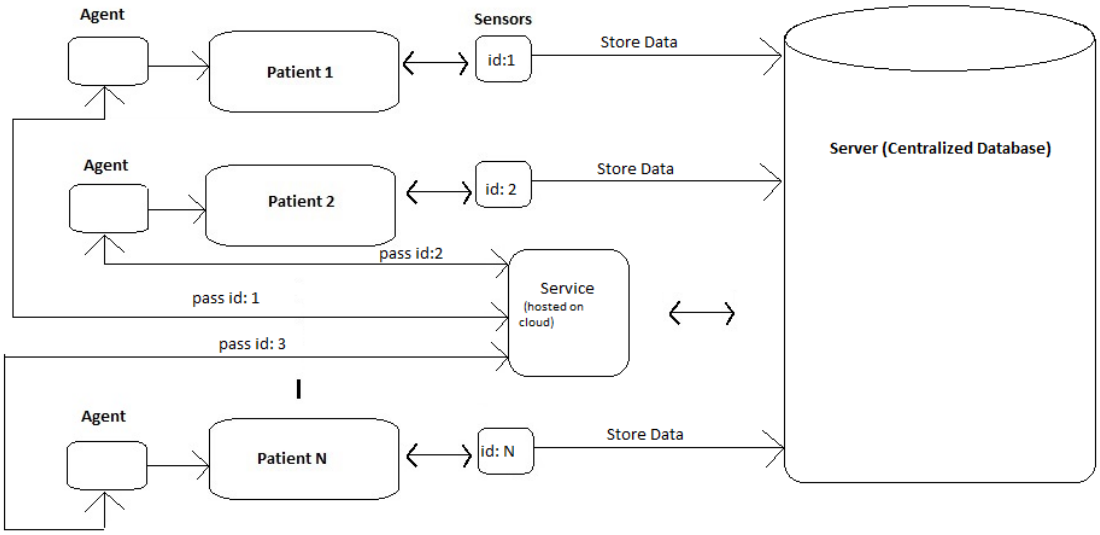

Fig 5 Usage of Cloud to receive instructions and data

3. Introduction to Proposed Solution

3.1 Proposed Structure

The latest workflow structures discussed in the previous chapter looks feasible and much more cost effective than the rest, but as the other two scenarios, it also has a few bottlenecks. For example, the method requires a constant access to the internet for the sensors to send the data to the Server and for the agent machines to access the service and get instructions. Also, there will be hidden costs for the maintenance of service on cloud and to keep it active and running for twenty-four hours every day. And since, our example scenario concerns with the very delicate situation, that is the hospital yard with various patients in either normal of critical condition, looking after the patients and making sure they are getting their dosages at the right time, we need to be sure that our system is 100% reliable.

One proposal of minimizing internet usage is to change the design of the agent machines in such a way, that instead of using a service to obtain instruction, they can be directed to use another service that can send the rules and set of instructions to the agent machine and it can then communicate directly with the sensors.

This also eliminates the need for the server to which the sensors send the information. Instead, what is happening is:

1. A new sensor S, is installed in the current system, for let's assume, patient P
2. The agent A, is already associated with patient P, the first message that sensor S will be send agent A will be the identification information.
3. The agent A can pass this information, to the service to retrieve the proper set of instructions on how to deal with the data it will be getting from the sensor later.
4. Upon receiving the instructions, the agent A can program itself to be ready to deal with the data that it is about to receive.
5. Sensor S and agent A will now communicate with each other. Eliminating the need of a cloud storage.

These steps are labelled in the diagram below to explain the workflow better. [Fig 6]

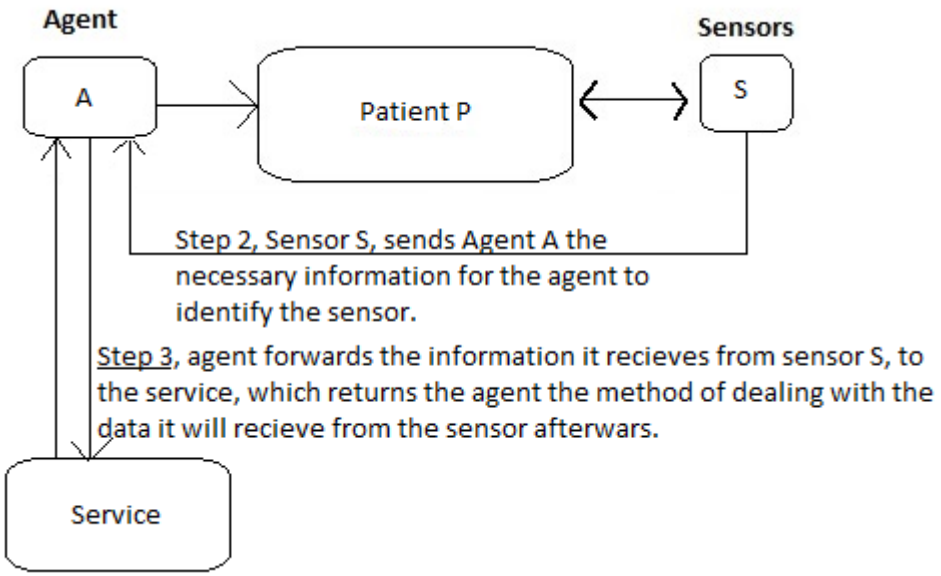

Fig 6 Server Connection Steps

Step 2, 3 and 4 and the initialization phase and the only time we need the internet connection will be in step 3. After that, till we get a new agent into the system, or a new sensor, we don't really need the internet connection.

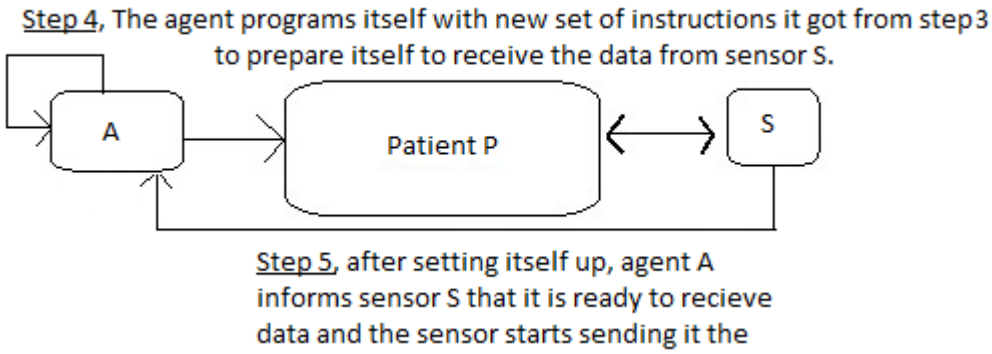

Fig 7 Communication Example

This structure, even though being reliable than the structural patterns discussed in last chapter, but in case we need to determine the change in readings for a long period of time to determine patterns and make predictions based upon those patterns, then one might need to integrate the cloud storage structure, as the servers have ability to store data (that it retrieves from the sensors) over a longer period. In this case there might be limited or no storage functionality [Fig 7]. And implementing this feature in the agents (or sensor) might increase the costs and would also might require maintenance costs.

Though this architecture's will have uncanny resembles to the last structure discussed in previous chapter, its communication with the internet (to retrieve data from the storage) will be eliminated. And instead, the agent (or the sensor) can be given enhanced features to store the data for some interval of time.

The DF performs all its original functionalities and it will also support in helping sensors the desired agent that they need to communicate with. For example, if a sensor is registered to a platform and is assigned to the patient, it will forward a request to the DF and ask to send it back the ID for the agent that is associated with the same patient. The service searches from its directory and forwards the information to the sensor who then starts to communicate directly with the agent. And as discussed earlier, the first message the sensor needs to share with the agent is its identification code. The agent will use this code to retrieve the set of instruction on how to act on the readings (data) it will get from the sensor. This is where another additional for DF comes in. It checks its local repository to check if it has the set of instructions required to help agent process and interpret the data it obtains from the sensor. If it doesn't find it, then the DF needs to check the main database on the internet and retrieve the instructions from there and store it locally. Then then forward it to the agent.

Similarly, the service can check at regular interval for the instructions that have not been accessed for over a period and can remove them from local storage to keep free space for other modules. This will keep the database clean of old and obsolete versions of sensor instructions.

Once the agent configures itself with instructions to process the data from the sensors, the agent and the sensor communicate directly with each other over local network. Hence that eliminates the need to constantly obtain the data from the internet which was the case in the previous scenario.

3.2 Introduction to Service Workflow:

In order to answer the research question, this thesis studies a service that connects the agent platform, JADE, to the sensor's platform, known as 52° North Sensor Observation Service [8]. This service has an interface based upon the concepts of semantic web and will use the SSWAP protocol for message passing.

Since both open-source platforms are developed in Java, the intermediate service will also be developed in the same language and will provide a web interface as well as a RESTFUL web service interface for the other applications to use it.

The service will be communicating with JADE platform's Directory Facilitator (DF) service as well as its Agent Management System (AMS) service at all times.

("The DF (Directory Facilitator) provides a directory which announces which agents are available on the platform". [7])

("The AMS (Agent Management System) controls the platform. Is the only one who can create and destroy other agents, destroy containers and stop the platform." [7])

The new JADE operations that will be available to the user will be as follows:

A1. Add/Remove Agents (preexisting method provided by JADE, will need some modifications).
A2. Register/Un-register Sensors for an agent (new method – communication with 52° North).
A3. Get observations from the sensor (new method - communication with 52° North).

In addition to that, the service will also have a bi-directional communication with 52° North platform, for communicating with the registered sensors. Hence, the new methods can be identified as:

B1. Add/Remove sensor(s) (will be invoked by A2).
B2. Return-Observations (will return either current reading or the readings over some time interval based upon the parameters passed. This method will be invoked by A3).

The service includes serialization method(s) to transform the output from the functions, to the SSWAP format and then pass them on to the interface. In the opposite direction, the service will also have a de-serialization method(s) for extracting relevant information from the incoming messages in the SSWAP format, extracting the relevant parameters and then invoking the desired method(s).

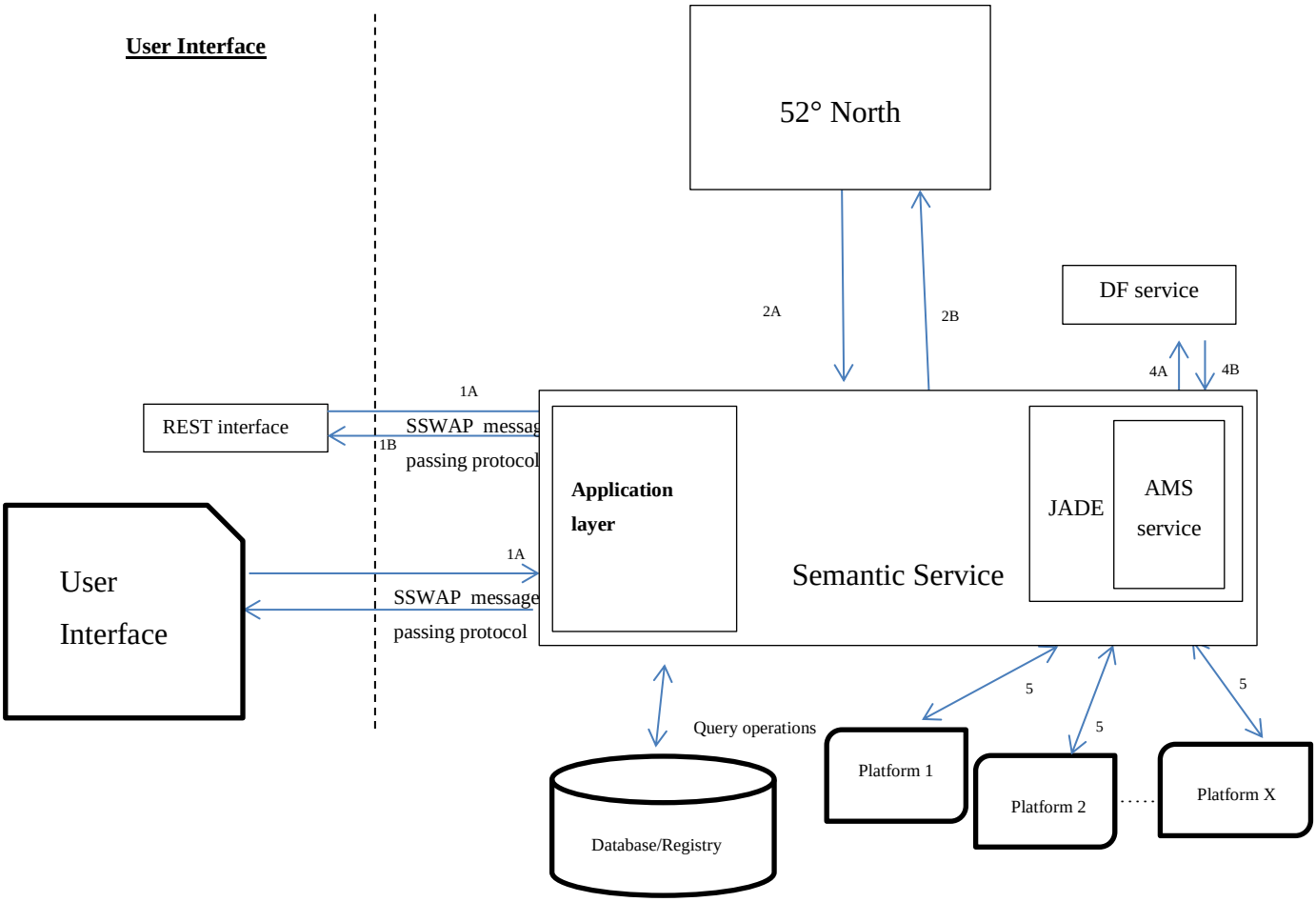

Fig 8 Workflow diagram

3.3 Applicability of Proposed Structure

With the help of the workflow diagram shown above [fig 8], a common workflow scenario will be explained. Operation requests can be initiated by the either the user or autonomously. For example, the agent may realize that it does not require services of sensor X that has been registered in its database. The agent can then invoke the method to unregister the sensor X, from its registry (database), automatically.

Another frequent scenario can be a user requesting for retrieval of all the data collected by some sensor, say X, belonging to the Agent Y.

1. The message passed will come through step 1a and the format of the message will be SSWAP. The service will process and de-serialize the request to extract the meaning and will invoke the corresponding method of Jade Platform with correct parameters (Step 3A).
2. Jade platform will then first use the directory Facilitator service (an agent dedicated to access the service whose sole purpose is to provice the features exposed by every other agents in the system. This is further explained later in the thesis) to find out in which platform the agent is deployed using Step 4A and the address to the agent will be returned by the service using Step 4B.
3. After getting the address, the agent platform will use the AMS service to retrieve information from the agent (Step 5).
4. After this, the main service will use this information to generate a query to retrieve the list of registered sensors to this specific agent and then using Step 2A, it will ask those sensors to return the readings. The readings will be returned using Step 2B.
5. Finally, the results, which are encoded in XML format, will be converted to RIG (SSWAP format) by using serialization method and then they will be sent back to the user (Step 1B).

3.4 Java Agent Development Platform

3.4.1 Background

As mentioned in the earlier section, JADE is a multi-agent platform that follows FIPA's specifications. FIPA stands for "Foundation for Intelligent Physical Agents". Founded in 1996 [19], the standards defined by this organization are being followed by several academic institutes and companies including HP, IBM and Fujitsu. JADE follows the FIPA's standards for Agent Management and for Agent Communication Language (ACL/FIPA-ACL).

We can define JADE by stating that it is a Java based framework (that follows FIPA specification [11]) that can be used to develop agent based system. JADE is an open source software that is developed by Telecom Italia [12]. It consists of a runtime environment (Containers), which hosts the agents and provides a location to execute. It also consists of various libraries that developers can attain to develop different functionalities for the agents. JADE also provides a set of APIs that can help in creating a distributed multi agent systems. A

"yellow page" service supports subscribe discovery mechanism for distributed multi agent systems as well [12]. To administrator and debug the running agents, we can use the graphical tool that are included in the framework.

JADE follows the standards defined by FIPA [13]. It puts two mandatory conditions for a (distributed) agent platform. They are as follows:

a. Each host should provide a runtime environment for agents to perform their tasks such as message parsing, lifecycle stage management etc.
b. There needs to be at least one "main" host which contains a directory of all the other hosts in the platform. This directory will be used for agent discovery.

3.4.2 Structure

As JADE follows the standards defined by FIPA, its architecture follows the baselines of FIPA agent platform (as shown in the Fig 9)

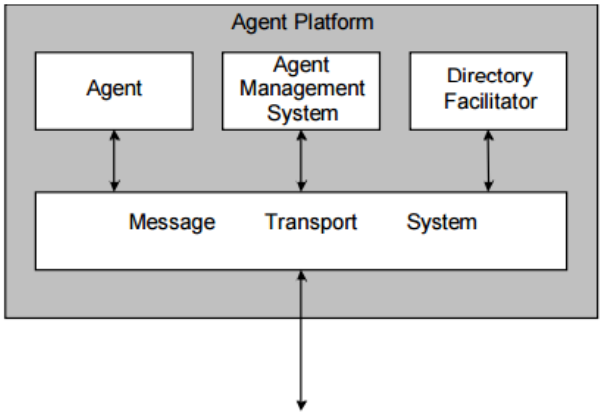

Fig 9 Reference Architecture 1 [14]

Agent Management System (referred in the future as AMS) and Directory Facilitator (referred in the future as DF) are nothing but agents with special features. AMS is unique for every platform but one can create several instances of DFs. One instance of a JADE environments is known as a CONTAINER [18] and one platform can have several containers.

But each platform has one container at least, known as MAIN CONTAINER, which is usually the first Container initialized in the platform. It holds the AMS and DF, and the rest of the containers which gets initialized later on, registers to the MAIN CONTAINER [18].

All the agents in the platform need to register with the AMS. AMS acts like a supervisor and its tasks include to keep track of all the agents in the platform and the state they are in. Upon registering, every agent is assigned an Agent ID (referred in the future as AID). A directory is maintained in AMS with the AIDs of all the agents in the platform and their current state. A JADE agent have 5 states in total and an agent's circulation between these states define an Agent Lifecycle. [Fig 10]

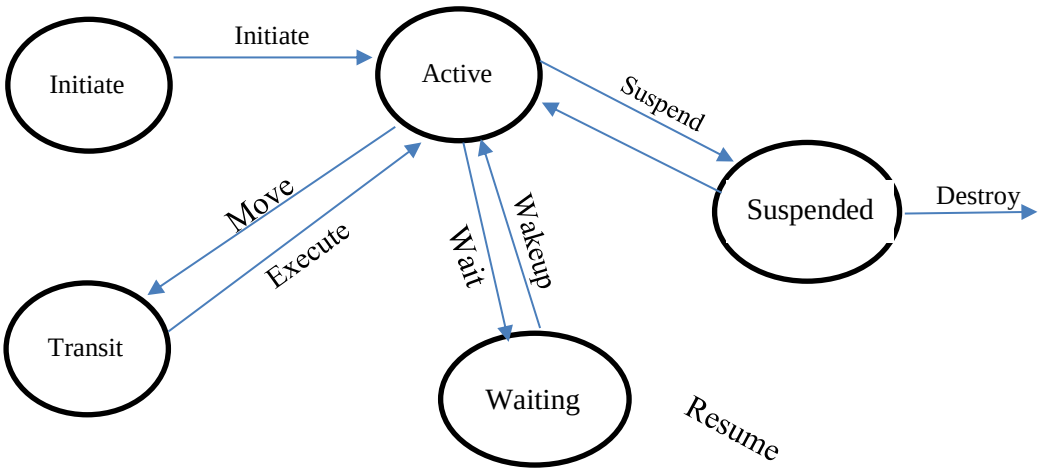

Fig 10 A JADE Agent Life-Cycle

The agent is said to be INITIATED when its instance is built. As soon as the instance or the agent is registered to the AMS and is assigned an AID, it comes in the ACTIVE state. Once in this stated, the agent can access all the features provided by the platform. The agent is in TRANSIT state when it is moving from one location to the other. When the agent is WAITING state, it is waiting for some operation to be completed and it can't do anything else. Once the condition is met, the agent can "wake up" and be active again. An agent will go to "SUSPENDED" state when it is idle. And Finally, Agent(s) can be deleted by removing their AID from the AMS. Agent Class provides the methods to navigate between states. Naming

convention of these methods are made simple by following the states the agent is going it. e.g. doMove() will change the state of the agent to Transit and so on.

The function of the DF agent is to provide a service called “yellow pages”. It is a service that allows all the agents to publish the functionalities they offer and also to search and make use of the functionalities offered by other agents in the platform. [Fig 11] Agents who publish their services need to provide their AID, the interface, and the (list of) published service(s). And for each service to be published by and agent, a service description, type, name, languages required to use it (by other agents) is required.

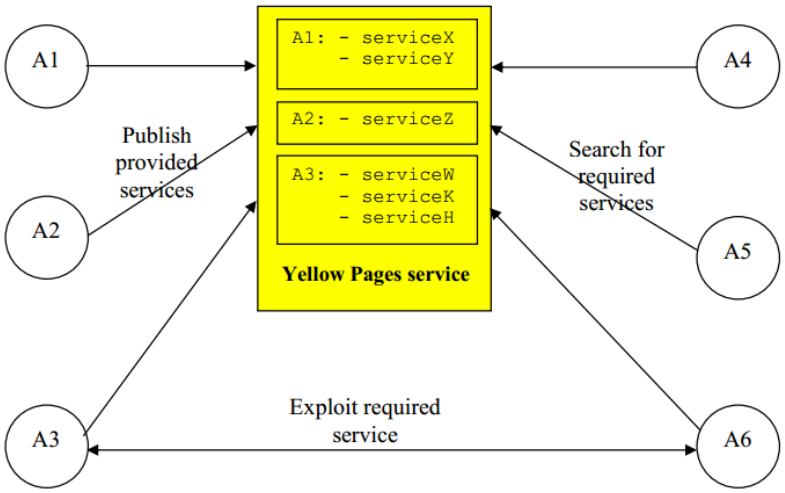

Fig 11: Yellow Page Service. [16]

The agents wishing to search from the list of available services need to provide DF a template of service description and the DF service will search and return the list of matching services from the yellow pages. The services are identified to be a match when all the parameters in the search query matches with the parameters in the service description of the service(s) registered in the yellow page service.

The DF agents have four basic functions that covers all the functionalities [17]. They are as follows:

- Register: Registering Service to the yellow page.
- Deregister: taking the service off from the yellow page.

- Modify: to modify some part of the service description of already existing service in the yellow page.
- Search: to look for a service from the list of services registered in the yellow pages.

To provide communication between agents, AMS and DF(s), we use Message Transport System, commonly known by the name of Agent Communication Channel (referred to in the future as ACC). It controls the communication taking place inside the platform as well as with the other platforms.

3.5 Semantic Sensor Observation Service (SemSOS)

3.5.1 Background

Sensor Observation Service (SOS) is a web service that is working in standardizing and increasing the interoperability of heterogeneous sensor data and the sensors. The service follows the standards set by the Open Geospatial Consortium (OGC) Sensor Web Enablement (SWE) [15]. The applications that use the sensor data generated are often not very well equipped with process the raw form of the data generated by the sensors. To deal with this, the sensor data is made more meaningful by adding a suite of ontologies, semantic annotation and using ontology models for reasoning. This extra layer of applied semantics to the sensor data gives us a SemSOS platform [15] [fig 12]. Some of the benefits of integrating both semantic web and sensor web are listed as follows:

- Independently accessible entities by using Uniform Resource Identifiers.
- N3 notations can make datasets integrations easy.
- Reasoning by using OWL.

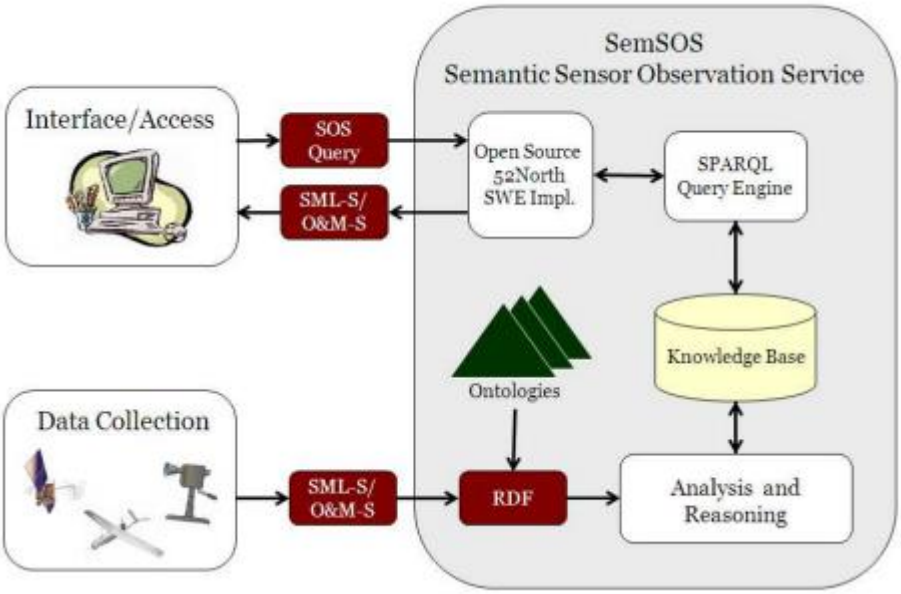

Fig 12 Architecture of SemSOS [15]

3.5.2 Overview

Semantic SOS is a result of combining sensor web with semantic web. Nowadays, there is a lot of data available on the internet that has been captured by sensors. In fact, the term "Internet of things" uses this data (or observations) to achieve various tasks, depending on the situation/scenario.

The sensor data obtained from different sensors that is available on the internet, follows encoding methods defined by Sensor Model Language (SensorML) to store the information about its sensor [21] and the values are encoded in Observation and Measurement(O&M) format [22]. O&M is an international standardized encoding schema for observations (from the sensor). Both SensorML and O&M encoding format have been standardized by OGC (Open Geospatial Consortium) and is a commonly followed standard among various organizations worldwide [2l].

SensorML is nothing more than an XML file that defines the sensor's features like its location, sensor description, how to process observations (data) and etc. With Sensor Observation Service, we can query either real time data or data obtained in a certain period. Sensor observation service uses the SensorML file to identify if the data it needs is provided by the sensor it called and if so, then through what method can it decode the data returned in O&M format.

The three main core operations in any SOS are as follows [23]:

- **GetCapabilities**: This method return the service description, period in which the data is obtained, exposed endpoints, sensor description (e.g. name, location, etc.)
- **GetObservation**: Returns the observations in O&M format.
- **DescribeSensor**: Returns sensor's description. It basically returns the contents of SensorML

Besides these core operations, there are some enhanced and transactional methods available as well. These features come as separate profiles to handle access operations. E.g. for handling transactional features, there exists a transactional profile which has the following two methods:

- **RegisterSensor**: By using the sensorML, the sensor manufactures (or data providers) can register their sensors with the SOS.
- **InsertObservation**: is used to register the new observations for the sensor. E.g. if a new component it added to the registered sensor, and now it has two set of observations then we use this function to register the new observation.

Then there are some other methods grouped together to form Enhanced profile. The methods it consists of are:

- **GetResult**: This method is used by clients to get data from the sensors periodically without sending multiple requests.
- **GetObservationByID**: This returns one single observation with the specific ID.
- **GetFeatureOfInterest:** This method returns the features of the sensors. They can be a single feature or multiple.
- **GetFeatureOfInterestTime:** It returns the time data for the feature of interest. E.g. one feature of interest for a particular sensor could be it's location. We can obtain the places where the sensor was during specific time by using this method.

- **DescribeFeatureType:** This method returns the details about the feature in a XML format.
- **DescribeResultModel:** Returns O&M XML for the observation. It is helpful to decode and process data from complex sensor models.

3.6 Integrating Semantic Interface with JADE

3.6.1 Introduction

With the help of Semantic web and its protocol, it can be possible to build a semantic layer upon JADE to allow components registered with SemSOS, to interact and use the services provided by the JADE platform and expose their information to the platform so that they can be used by agents present in the platform. This will open door to two-way communication between both SemSOS and the JADE platform, without the dire need of understanding the protocols and standards defined at the backend of the respective applications. Both platforms will follow semantic protocol and SSWAP message passing standards to communicate with the semantic layer to communicate (or obtain information from the specific component) of the other platform.

Similarly, the ontology files will store the interface definitions for every method and as a part of future development, further components can be defined as well in these ontologies files to make their functionality available for common use. And at the end, there can even be a possibility of adding other platforms given that they can provide their ontologies.

The main advantage of having a semantic layer for intra platform communication can be summarized as follow:

1) No need to define new set of standards and protocol as semantic web comes with W3C approved standards and its own protocols.
2) A two-way communication will be established between JADE and SemSOS that will allow the features of both the platform to get accessed by the other without knowing the interface of the method (as that information will be stored in RDF files in semantic layer)
3) SemSOS already has semantic interface for its methods which can be easily accessed via this layer
4) Semantic layer will provide transparency to both the platforms from the other methods and protocols.
5) Methods exposed by both the platforms in this semantic layer can be made accessible to the other platforms that registers itself with this service.

3.6.2 Architecture

We start defining the architecture of the proposed service by first, going through the schema of metadata that we require to hold the information for both the methods that are exposed to be utilized from JADE and SemSOS platform.

To explain the service and the methods it possesses, we require one OWL file that will hold the schema of our service and the methods.

Web ontology language (OWL) is a modelling language which includes information about the data. In our case, we will use OWL file to expose the interface of our service by defining our exposed methods, objects and data properties for the external applications to understand and use the service.

Alternatively, we will be maintaining extra RDF files to keep track of all our data and their relationship. These files will hold information related to only one domain at a time. That means we will have two files, each one corresponding to each platform

The ontology includes a collection of all the methods (classified as object properties) from JADE platform that can be accessed from SemSOS platform and vice versa, i.e. the methods exposed by SemSOS to be used by JADE. In addition to this, the information about the parameters for the functions, their data type and the return type will also be saved here.

We have already discussed what essentials are required in building an agent platform in JADE. Each platform consists of two essential modules, namely Agent Management System (AMS) and Directory Facilitator (DF). Whenever a new agent JADE agent platform is registered with the service, it will provide its location to its AMS and DF services. These paths will be added to our ontology along with extra information describing the purpose of the platform.

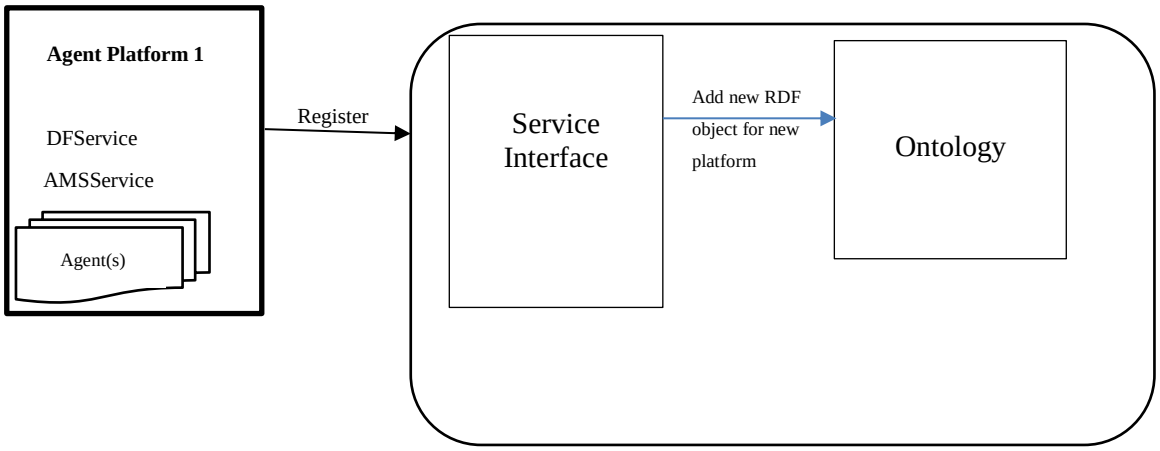

Fig 13 Service Register Workflow

In the ontology, the class that registers the new JADE platform is called “Register-JADEObject”

```
:RegisterJADEObject rdf:type owl:Class .
```

The dataproperties that this class posses are as follows, these will hold basic information about the platform and the URLs to access its DF and AMS

```
###  http://localhost:8888/SemService#AMSServicePath

:AMSServicePath rdf:type owl:DatatypeProperty ;

                rdfs:range xsd:anyURI .


###  http://localhost:8888/SemService#DFServicePath

:DFServicePath rdf:type owl:DatatypeProperty ;

               rdfs:domain :RegisterJADEObject ;

               rdfs:range xsd:anyURI .


###  http://localhost:8888/SemService#description

:description rdf:type owl:DatatypeProperty ;

             rdfs:domain :RegisterJADEObject ;

             rdfs:range xsd:string .


###  http://localhost:8888/SemService#platformUID

:platformUID rdf:type owl:DatatypeProperty ,
                      owl:FunctionalProperty ;

             rdfs:domain :RegisterJADEObject ;

             rdfs:range xsd:integer .
```

Fig 14 Data properties for RegisterJADEObject class

Suppose we have a new agent platform that we have to register, let's for example, take a platform called "FactoryXAgentPlatform", for this, the storage in the RDF will look like as below:

```
###  http://localhost:8888/SemService#FactoryXAgentPlatformIndividual

:FactoryXAgentPlatformIndividual rdf:type owl:NamedIndividual ;

                          :platformUID 19291 ;

                          :description "This platform contains agents present in Factory X, providing autonomus functionalies
                                        for data collection and automation process"^^xsd:string ;

                          :AMSServicePath "services.factoryX.fi/amsservice.asmx"^^xsd:anyURI ;

                          :DFServicePath "services.factoryX.fi/dfservice.asmx"^^xsd:anyURI .
```

Fig 15 Instance of registering a new Jade Platform

Similarly, for other states in the agent life cycle, we will have separate representation of them as OWL classes. Once an agent is invoked, it goes to Active state. The class representing that states is called “ActiveJadeObject”.

```
:ActveJadeObject rdf:type owl:Class .
```

An agent becomes a member of ActiveJadeObject class after being invoked from Register class. Suppose, after registering “FactoryXAgentPlatform” we want to invoke it to become active. In order to call the invoke method, we have our “InvokeAgent” object property as follows:

```
:InvokeAgent rdf:type owl:FunctionalProperty ,
                      owl:ObjectProperty ;

             rdfs:range :ActveJadeObject ;

             rdfs:domain :RegisterJADEObject .
```

So when we invoke our example, the following object property will be added to the instance of registration object:

```
:FactoryXAgentPlatformIndividual rdf:type :RegisterJADEObject ,
                                          owl:NamedIndividual ;

                    :InvokeAgent :FactoryXAgentActive.
```

Where FactoryXAgentActive is a member of ActiveJadeObject class.

```
:FactoryXAgentActive rdf:type :ActveJadeObject ,
                               owl:NamedIndividual ;


###  http://localhost:8080/SemService#FactoryXAgentActive
```

The instances of this class will have access to all the features provided by JADE and they can only execute their actions (or behavior) while in active state.

Similarly, to how AciveJadeObject represents the Active instance of the Jade lifecycle, we have classes that define the instances of the other classes. For representing each Jade lifecycle instance, we have the following declarations:

Suspend:

```
###  http://localhost:8080/SemService#SuspendedJadeObject

:SuspendedJadeObject rdf:type owl:Class .
```

An agent object becomes an instance of this class after it has fulfilled its tasks and have provided its functionalities to achieve an Active object. Being in this state, the agent can be asked to resume its functionality after a certain period. And hence it will move back to the Active class. A common example of a agent switching between these state can be a temperature gauge sending reading of a specific place after a regular interval. When an agent is in Suspend state, it is using less resources from the server and hence in the end it will eventually help in maintaining the resource allotments.

Transit:

```
###  http://localhost:8080/SemService#TransitJadeObject

:TransitJadeObject rdf:type owl:Class .
```

An agent takes the instance of Transit class when it's been ordered to change its physical location. As an instance of this class, the agent focuses on migrating to the new location as efficiently as possible and dedicate (and make use of relevant) resources required in achieving the transit task proficiently.

Waiting:

```
###    http://localhost:8080/SemService##WaitingJadeObject

:WaitingJadeObject rdf:type owl:Class .
```

As the name of this class suggests, the instances of this class are in some way waiting for either further instructions or resources to become available so that they can perform their task.

Object Properties

In order to switch states, we need to have a proper channel to link the instances from one class to the instance of other class so that they can be placed at the correct state during their lifecycle. We have different object properties that are solely responsible for ensuring the smooth transition of instances from one state to another.

One such property that we looked at earlier was InvokeAgent. This connects the agent's register state to the agent's active state. So now when the Jade method "DoInvoke" is raised, the agent will know the exact instance of ActiveJadeObject to move to.

Similarly, we have other object properties that are used to link these states. For changing the state to WaitingJadeObject class from active, we have a property called "AgentWait". Its RDF looks like as follows:

```
### http://localhost:8080/SemService##AgentWait

:AgentWait rdf:type owl:ObjectProperty ;

           rdfs:domain :ActveJadeObject ;

           rdfs:range :WaitingJadeObject ;

           owl:inverseOf :WakeUpAgent .
```

As you can observe, the last line in the above picture, the object property AgentWait is the inverse of another object property named WakeUpAgent. WakeUpAgent moves the state from Waiting to Acive.

```
###  http://localhost:8080/SemService#WakeUpAgent

:WakeUpAgent rdf:type owl:ObjectProperty ;

             rdfs:range :ActveJadeObject ;

             rdfs:domain :WaitingJadeObject .
```

In order to move between Active and transit state and vice versa, we have two object properties called “MoveAgent” and “ExecuteAgent” respectively. Jade corresponding methods for them are “DoMove” and “DoExecute”.

```
###  http://localhost:8080/SemService#MoveAgent

:MoveAgent rdf:type owl:ObjectProperty ;

           rdfs:domain :ActveJadeObject ;

           rdfs:range :TransitJadeObject .

###  http://localhost:8080/SemService#ExecuteAgent

:ExecuteAgent rdf:type owl:ObjectProperty ;

              rdfs:range :ActveJadeObject ;

              owl:inverseOf :MoveAgent ;

              rdfs:domain :TransitJadeObject .
```

4. Integration into Existing Workflows

We shall now discuss the possible ways we can modify and enhance our service application to fit the different architectures discussed in chapter two.

To follow the standards and architecture proposed in this the first example we discussed [fig 4], we need to take into consideration that there is a local network of SemSOS and JADE operating within the premises of the local network system because in this example architecture, the internet is not being used, hence decreasing the risks of communication failure.

For that case, we can have two servers set up in our environment, each one dedicated to JADE and SemSOS respectively. The rest of the architecture remains pretty much the same but instead of the sensor and the agent are communicating directly and the agents will be fed instructions beforehand on what kind of information they will be receiving and how they can process it and take actions based on the processed information. Though this approach might seem reliable as there is a need for only local network to be set up, but at the same time, it increases the effort for configuring the agents so that they know how to interpret the information it receives and act accordingly.

Similarly, if there is a requirement to update or change either the agent or the sensor involved, then the other component will have to be updated accordingly so that it can work along the newly installed component and this will have to be done manually. Hence the updates process will not be autonomous and will be cost effective to configure the components.

The last example this thesis talked about, for the architecture, where agents and the serviceflow communicates with the cloud storage [fig 5], reduces the work load significantly and increases automation. The agents in that environment will have access to the yellow page service that could be hosted either locally or on the internet (recommended). From there they can extract the exact information that they require from the database. For instance, if a new entity (patient) requires a new assistant (agent), they will be provided with one and that assistant will be provided basic information, for example, what actions does it need to take

(or how to process the data it is provided with) and to what patient it is assigned to (patient ID).

Similarly, the sensors, when they are assigned to a patient, will also be provided with the ID of the patient, along with the address of the database where they need to send the readings (data) to.

The main role will be played by the (yellow page/Directory facilitator) service. It will hold information about all the different sensors and at what intervals do it has to retrieve the reading for specific intervals. Other than this, the service is also responsible for querying and passing the reading from the database to the correct agent (a match can be made on PatientIDs) so that the desired action can be taken by agents, after the process the data. Similarly, a few more purposes that the service can serve is to receive updates for the agents that are in the environment and then pass the update to all the related agents.

5. Related Work

In a dissertation presented by JUSSI KILJANDER, ”SEMANTIC INTEROPERABILITY FRAMEWORK FOR SMART SPACES” [24], the semantic level interoperability of devices is discussed. This work is more related and focused on developing Semantic interface service for increasing operability between agents and sensor devices, with heavy focus on making use of two platforms SemSOS and JADE. Where as the related work focuses on using semantic protocal on distirbuted systems and mapping physical entiities to virtual entities.

6. Conclusions

This thesis discussed several different architectural scenarios in which the agents and sensors are configured to work together (Chapter 1 and Chapter 4). Following a use case of a hospital scenario, the work compared pros and cons for every one of them. We observe that there exists already a lot of services following these architectures. But with especial focus on using the principal of Semantic web in establishing the communication protocol between agents and the sensor platform.

This thesis introduces two third party services, JADE and SemSOS, and explains on how they can be interconnected and communicate with each other using the concepts from semantic web. This work also includes the RDF structural of classes that will be used in doing so. Later on, the thesis explain on how the proposed architecture can be used in the different scenarios that were discussed earlier.

The whole concept that this thesis focuses on is whether if it is feasible and actually smart to follow semantic protocol when integrating different agents and sensor platforms. The advantages were touched and talked upon in this thesis. Hence this thesis looked at and sketched a skeleton on how the semantic web and RDF language can be used to create in interface layer to establish communication between agents and sensors belonging to different vendors (Discussed in detail in Chapter 3). This RDF document is saved as OWL notation and can be retrieved and enhanced by anyone who wish to work further expanding this topic.

This thesis also looked at these concepts with the perspective of using JADE and SemSOS. In chapter 4, the possibility of integration of the service in to some of the scenarios that already exists have also been discussed.

7. References

[1] Xiaohang Wang et al. (2004) Semantic Space: An Infrastructure for Smart Spaces

[2] http://www.w3.org/2001/sw/

[3] Michael Blackstock and Rodger Lea (2014) IoT Interoperability: A Hub-based Approach

[4] http://www.cs.jyu.fi/ai/OntoGroup/UBIWARE_details.htm

[5] http://jade.tilab.com/documentation/tutorials-guides/introduction-to-jade/

[6] http://52north.org/communities/sensorweb/sos/

[7] http://en.wikipedia.org/wiki/Java_Agent_Development_Framework

[8] http://52north.org/communities/sensorweb/sos/

[9] http://www.w3.org/standards/semanticweb/

[10] Art Botterell (2012) A Pragmatic Approach to Smart Workspaces for Crisis Management

[11] http://www.fipa.org/

[12] http://jade.tilab.com/

[13] http://www.fipa.org/

[14] Fabio Bellifemine, Giovanni Caire, Tiziana, Giovanni Rimassa (2010) JADE Programmer's Guide

[15] http://link.springer.com/chapter/10.1007/978-3-540-79996-2_10#page-2

[16] Giovanni Caire (TILAB, formerly CSELT) , Jade Tutorial Jade Programming for Beginners, Figure 4, pg.21

[17] Fabio Bellifemine, Giovanni Caire, Tiziana, Giovanni Rimassa (2010) JADE Programmer's Guide , pg 9.

[18] Giovanni Caire (TILAB, formerly CSELT) , Jade Tutorial Jade Programming for Beginners, pg. 4

[19] https://en.wikipedia.org/wiki/FIPA

[21] http://www.ogcnetwork.net/SensorML

[22] http://www.iso.org/iso/catalogue_detail.htm?csnumber=32574

[2l] http://www.opengeospatial.org/ogc/historylong

[23] https://en.wikipedia.org/wiki/Sensor_Observation_Service

[24] JUSSI KILJANDER, SEMANTIC INTEROPERABILITY FRAMEWORK FOR SMART SPACES,