Journal of Network and Innovative Computing
ISSN 2160-2174 Volume 1 (2013) pp. 260-269
© MIR Labs, www.mirlabs.net/jnic/index.html

JADE Agents to Wireless Sensors: Easy Wireless Sensor Network Management

Jakub Zak[1], Frantisek Zboril jr.[2], Jan Horacek[3] and Frantisek Zboril[4]

[1]Faculty of Information Technology, Brno University of Technology
Bozetechova 1/2, 612 66 Brno, Czech Republic
izakjakub@fit.vutbr.cz

[2]Faculty of Information Technology, Brno University of Technology
Bozetechova 1/2, 612 66 Brno, Czech Republic
zborilf@fit.vutbr.cz

[3]Faculty of Information Technology, Brno University of Technology
Bozetechova 1/2, 612 66 Brno, Czech Republic
ihoracek@fit.vutbr.cz

[4]Faculty of Information Technology, Brno University of Technology
Bozetechova 1/2, 612 66 Brno, Czech Republic
zboril@fit.vutbr.cz

***Abstract*—This particular paper describes our most recent work on our system called JAWS. This system is intended to control and monitor Wireless Sensor Network. In principle, JAWS is group of agents implemented on JADE platform. It consists of several agents that are able to communicate with Wireless Sensor Nodes and obtain values from particular sensor nodes called motes. We are also able to inject mobile code to each mote so we can change behaviour of that mote and in extension of the whole network. JAWS system uses services as a natural and most viable concept that helps us to control and monitor Wireless Sensor Network. We will briefly describe basic concepts of our system and describe services used for controlling the network. Concept of services is tightly bound to ability of the system to expose and use services. To expose and use services we created set of protocols. In this paper will be also described the most important protocols of our system. In the end of the paper we will describe some of the problems that we experienced using our system and we also describe some counter actions that we performed to troubleshoot some problems.**

***Keywords*-JAWS; JADE; Wireless Sensor Network; Service; Artificial Agent; Protocol.**

I. Introduction

Wireless Sensor Networks (further WSN) are in the scope of researchers in the last years. It is because of their ability to measure various physical quantities in physical environment. The second reason is that their motes (nodes of network) are also able to communicate with each other via radio channel. By this two abilities, WSNs are mostly used for monitoring purposes in some environment. This is useful when we need to measure data with high density of measurements because WSNs are able to run with thousands of motes that are distributed within measured environment. For data gathering is used special node called *Base Station* (further BS) that can be connected to computer and that can deliver data sent by motes from network. WNSs are also likely used in environments that are hostile for human or where human presence is not suitable. One example for all is experiment on monitoring habitats on *Great Duck Island*[1]. In this experiment, the ducks were monitored on island and presence of humans could change animal behaviour and thus devalue whole experiment.

There are various ways to control mote in WSN. Since our research is in area of *artificial agents* our systems are agent based. This approach has been proved viable in other agent based systems for controlling WSN (e.g. *Agilla*[2]). In our case we developed system called *WSageNt*[3] that is in principle similar to Agilla. It is agent based and it is a platform that runs on each mote. Agent is then understood as a small piece of code that is interpreted on platform. That is where the similarity ends. Agilla agent code is similar to assembler but our agents are written in *Agent Low Level Language*[4]. Since our ALLL agents can move among motes, ALLL language is designed to be minimalistic. It means that less bytes have to be sent via radio channel, which the is biggest consumer of the battery of a mote. As postulated in [5] sending 1 bit of information is equal interpreting about 1000 of instructions. We briefly describe WSageNt system in section II.

We explained the system running on each mote. This system allows each mote in WSN to sense environment (via mote's sensors) and it is able to talk to other motes in network. The only other thing we need now is BS that will collect data sent from motes. Motes can send data at any time so it seems useful to have a system that will gather this data from Base Station and that will process them in some, possibly intelligent, manner. Since "intelligent" agent runs on each mote, data can be processed directly on mote. For example sums of data or average values can be precomputed so less bytes will be transmitted over radio and some battery will be saved.

Even though data can be preprocessed some of it arrives to the Base Station and in extension to some system that will process them further to obtain more complex information than data itself. And JAWS is exactly such a system. It can obtain data from WSN on the one side and it can send data or even ALLL agents to network on the other side. Since

WSageNt is based on agent oriented paradigm it seems elegant to stick with the concept of agents when creating a system that will interoperate with WSN. So we decided to create pure agent based system and that's why JAWS is based on artificial agents. Basically, there are some agents that run on JADE platform and that create core of JAWS. These agents "live" on personal computer that has undoubtedly more computing power and that has in general more resources than one mote. From this point of view, it seems quite useful to compute a task in JAWS so some additional battery of mote can be saved. It is necessary to point out here that we need to find proper border that says, which tasks are better to compute directly on mote and which data are better to be sent to BS. For example, it is obvious that evaluation of topology of network won't be counted by each mote of network. It is more elegant solution to send agent that will gather necessary data and counting itself execute in JAWS. On the other side, when mote should just measure temperature we can measure 5 times and then count average value to partially avoid sensor inaccuracy. In this case few additions and a division (to count average value) looks better than sending 5 values instead of one.

In the next section we will describe our system that runs on motes so the reader gets context of Wireless Sensor Network and its possible behaviour. Then we describe basic concepts of our system JAWS including architecture description. Then we describe Services that we have created for possibility to control and monitor network. In section about services we also show main protocols that we use and that are bound to subscribing and using services in our system. Finally we describe some of the problems we encounter during use of our system and also some solution of problems outlined.

This paper is extension of the paper presented on ISDA 2012 Conference[21].

II. WS Agent

In this section we will describe our agent platform for sensor nodes, which is called WSageNt. WSageNt platform is loaded into all sensor nodes in the network and it interprets a code of an agent, which is written in ALLL (Agent Low Level Language).

Whole platform has to be minimalistic. Sensor node does not have enough resources to run more complex multi-agent systems such as Jason[14] or JADE[7]. Further in the text we will focus mainly on IRIS motes, that are supported by our WSageNt platform. Such node contains up to 8 kB of RAM, 802.15.4 compliant radio module, various types of sensor boards and its MCU runs at 8 MHz. We have to note that there is also one more important part of sensor node that limits its performance. This most limiting parameter of sensor node is the battery.

Sensor nodes are usually placed into inaccessible area so the maintenance of nodes and change of battery are impossible. Node should be able to run for months or years without change of battery (depends on application). Our WSageNt platform solves also one part of problem mentioned before. If we want to change a behaviour of a node it can be sometimes tricky. There are nodes, which can not be reprogrammed remotely through the radio. Since each ALLL agent is interpreted by our platform we are able to maintain all of the supported nodes remotely.

We will mention parts of each agent, some services and agent actions that are supported by our platform. Our platform supports interpretation of one agent at each sensor node. We can classify our platform to be a derivative of BDI (belief, desire, intention) system. Agent is composed from its belief base, input base, plan base, intention and 3 registers. More about agent parts can be found in [3].

In each step of interpretation, an agent can perform one action or run some built-in service. Actions include updating belief base, reading actual data from sensors, use of mathematical operations, sending message or migration of the agent to another node. Built-in services are for example periodical reading from sensors and preprocessing such data, neighbour discovery or footmarks of agents at each node.

We will demonstrate capabilities of our platform on the example bellow. There will be described an agent that will return an minimal value from data, which are read periodically from sensor. Such agent is written in ALLL code as follows:

```
$(d,(m,10))&(1)?(m)!(1,&1)
```

Platform senses data from sensor periodically. It is done automatically after the sensor node is turned on and simultaneously with performing of agent code. History of measured data is stored in circular log at the flash memory of sensor node. First action will select a minimal value from last 10 measured values. Minimal value is then put into the input base in the form of tupple `(m,MINIMAL_VALUE)`. Next two actions `&(1)?(m)` will transfer this tupple to the register number 1. Last action `!(1,&1)` send this tuple to the basestation (node with network address that defaults to 1).

This example of WSageNt should demonstrate capabilities of our platform briefly. However, there could be interpreted only one agent at each node at this moment. We intend to expand it to run up to 4 agents at one time in next version of our platform.

III. JAWS

JAWS is an abbreviation for **J**ADE[7] **A**gents to **W**ireless **S**ensors. We've chosen JADE platform because we try to follow standards in our agent based systems. Such standards are defined by *FIPA*[6], which is an organisation that creates standards in the field of agent based systems. There are some other organisations that cover this area but FIPA is the only one that defines standards for *whole* agent system and it will be shown further that we stick with its standards in all areas we can. JADE is a reference implementation of FIPA agent platform. It controls agents life cycles and it has knowledge about all agents present in JADE platform and further it defines communication infrastructure for inter-agent communication.

Firstly, we should point out that JADE is system designed to aim to agents services as a main concept of cooperation among JADE agents. Now we explain how we understand the term *service* and how it is realised in JADE. A service by the means of JADE is a piece of behaviour or ability that an agent is able to perform and that the agent is willing to perform for any other agent. For this purpose JADE contains a special agent called *Directory Facilitator* (further DF) that holds service called yellow pages. This concept means that any regular JADE agent can expose its services to Directory Facilitator and DF creates a list of services for every agent. When agents present in system expose their services (typically after start of platform), DF has list of agents and for each agent it has list of services that is that particular agent able to perform. At this point, any agent can ask DF different questions about provided services in system. Firstly, agents can ask about specific service by providing its name. DF returns list of agents that are able to perform this task. Any agent can also ask about services provided by other specific agent. Basically, any agent can search DF yellow pages by specifying various parameters from service description[9].

Now we must say that JAWS itself is not meant to run as a solitude system. By our understanding, JAWS is more an extension of JADE because it is meant to enable other JADE agents to be able to use all WSN features. Motivation for creating such a system is to create middle layer between application (application agents in JADE) and WSN running WSageNt system. When application programmer creates ready-to-deploy system he has prepared JADE platform with JAWS extension where JAWS encapsulates whole WSN. From application programmer point of view whole WSN resides on agent platform (JADE) and programmer can obtain any data and control whole network through JAWS. We could say that JAWS is an abstraction of whole WSN on desktop and it provides WSN features by exposing JAWS services to DF. We can JAWS

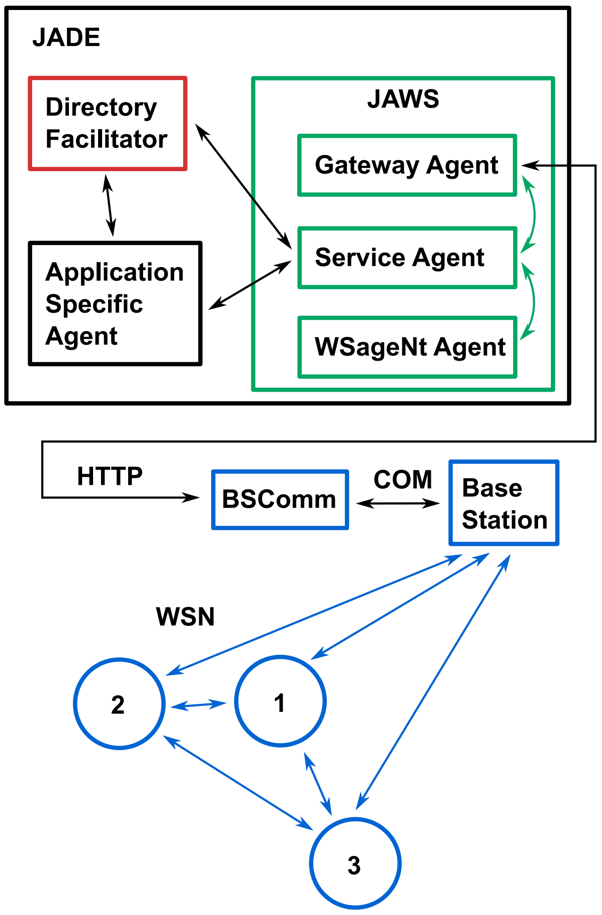


Fig. 1. JAWS Architecture

A. Architecture

In this section, we briefly describe architecture of the system depicted in Fig. 1. This is important for two reasons. Firstly, it should be clarified flow of information between sensor network and JADE agent system with JAWS. Secondly, it will be explained what kind of data travels which parts of system. This information raises new challenges for future research.

Natural information flow board in the system is situated between wireless sensors and JADE application on desktop computer. This board consists of two applications. First application is command line *BSComm*, which resends data between JAWS and Base Station. This part of functionality is dedicated to solitude application because of hardware purposes. When we change Base Station for different type, we just write new BSComm application for another piece of hardware and the whole system works without change.

It is worth mention here that between BSComm and *Gateway Agent* in JAWS is established asynchronous communication. We developed a small library that is able to play both basic roles, which are server and client. Server runs naturally in separate thread so it does not interfere with rest of application. Application is able to send three types of messages. First type of message is simple text message. This message is dedicated to send orders to agents on motes. Second type of message is agent message. This type of message is assigned to sending ALLL agents between applications. If we want to inject ALLL agent to network it is understood as agent message type between JAWS and BSComm. Last type of message is ACL message. Our client is able to connect to JADE itself through its interface so we are able to send message basically to any agent in JADE. This is for interconnection with our other tool called T-Mass[20]. Second application is on Base Station itself. This is special version of WSageNt application for Base Station.

The rest of the system can be divided into two parts. First part is represented by JADE platform with JAWS agents and at least one application agent running on desktop. On Fig. 1 there is depicted only one mandatory agent and that is Directory Facilitator. The other agents are not important in this paper so they were let out from picture. Added functionality in JADE is JAWS and Application Specific agent. As it can be seen, JAWS consist of three agents. Previously mentioned Gateway Agent is responsible for resending data between *Service Agent* and BSComm. This agent represents gateway to the world of JAWS. At this time, its responsibility is only to resend data but in the future there can be added functionality. For example, Gateway Agent can be responsible for application safety. When there is mote full of agents (up to 4) Gateway Agent can acquire this information and not let rewrite agent on that particular mote. Second is *WSageNt Agent*. This agent is responsible for guarding data about ALLL agents in system. There are two “kinds” of ALLL agents. First is list of *agents* *known* by system. These are ALLL agents that are inserted by an application programmer in the phase of designing an application. We can see this list as a library of plans where one plan corresponds one agent in list. These are agents that can be used in the running instance of deployed system. Second is list of agents that are currently used on motes. This list contains subset of *agents known* and also contains information about agents positions on motes. For this to be possible we presume that at start of the system motes are populated with agents by JAWS and in extension by demands of Application Specific Agent. Last but not least of our agents is Service Agent. This is JAWS key agent. This agent communicates possibly with all agents present in JADE at runtime. On one side, it operates with JAWS specific data that it receives via communication with JAWS agents. On the other side it communicates with Application Specific Agent/s to fulfil their application specific demands. Third channel of communication is connected to Directory Facilitator. Service Agent uses DF so concept of services is implemented in JADE/FIPA manner.

Second part of the system is represented by Wireless Sensor Network running WSageNt. As said previously, WSN collects data in some environment. That's its main purpose. Then it sends data to JAWS. These data are represented by tuples with strings. So to the JAWS comes only the tuple and address of mote that sent data. This is the only information we have in JAWS. Now we need to send data accordingly to services subscribed by application agents. We have only mote address but we need to create some representation of data incoming in message from network. This is difficult task. We can imagine following scenario. At the beginning we send ALLL agent to the WSN. That agent only measures two physical quantities each with its period of time. It sends data back to the Base Station instantly after measurement. Since to the JAWS comes only tuple of data we don't know, which quantity was measured (in reality mote sends only resistance of sensor so actual value needs to be recounted according to proper formula written in sensor/mote datasheet). We can know the initial order of measurement but if one measurement is lost (some packets are lost when transmitted) JAWS won't be given one measurement so initial ordering is lost. Ordering is not the solution then. We can also add to each measurement information about sensor but this solution increases transmitted data volume thus depletes batteries more quickly. Current state of art in JAWS is following: We presume carefulness of application programmer so when some data come from mote it can be represented in only one possible way. For example, if that mote is able to send only one type of data (temperature) it is clear what type of data came from it. This presumption gives a lot of space for possible errors because all responsibility is on application programmer. This solution is not quite robust and it gives a new space for further research. We have proposed and tested some techniques that lower a little uncertainty of data coming from the network and we dedicated to this area section V.

IV. JAWS Services

JAWS is meant to be extension of JADE's concept of services. More precisely, it is a dynamic set of services that enables application programmer to tailor ready-to-deploy systems. As said earlier, access to services is provided by Service Agent. This agent is responsible for handling services to both sides of JAWS. To the side of WSN, Service Agent gives orders to send messages or agents to specific motes. When a message comes from the network this agent also decides if its content should be given to Application Specific Agent/s (further ASA) according to subscribed services. To the side of JADE, Service Agent communicates with ASA in two scenarios. Firstly, ASA needs to subscribe for some service. Initiator of conversation is ASA in this case. More detailed description of this process will be given in subsection IV-C. In the second case, Service Agent resends data incoming from Wireless Network if it finds the data suitable for resending according to previous ASA subscription.

At this point, we should explain our use of ontologies[10] in the system. We use the term ontology in two situations. First situation is ontology concept as it is understood by JADE. We can provide ontology for subscription as an example. By the means of JADE ontology is set of classes that fully describe one concept. This concept is subscription in our case. When ASA wants to subscribe for some service it needs to be able to describe service it wants to use. It simply fills out some fields on prepared object and it sends that object to Service Agent. The biggest advantage coming from this approach is that JADE is automatically able to create message from ontology and send it as string via its communication channels. Agent on the other side only uses prepared empty objects and let JADE to decode message. So by using concept of ontology this way, we stick with standards and we don't force application programmer to learn new techniques to send messages/concepts among agents.

The second usage of the term *ontology* is related to ALLL agents. By the means of JAWS, each of ALLL agents has field for its ontology string. This string contains description of list of services that is agent able to provide. Since agent on mote is able to receive messages and act according to them it is able to run various services. This means that one ALLL agent can provide more than one service. This creates necessity to describe its services somehow in JAWS. So ontology in the meaning we understand it in the relation with ALLL agents is list of names of services that are separated by a delimiter character.

Basically, we have two kinds of services in the JADE manner of understanding. First kind is present in each ready-to-deploy system created by application programmer. This type of services is called *Pure JAWS Services* (subsection IV-A). These services were created because there are common tasks and routines that can be used generally in every application. Second kind of services are *ALLL Services* (subsection IV-B). Each ALLL Service is represented by one ALLL agent.

As a practical example of use of our services we designed simple experiment. We created application agent that subscribed for ALLL service of one agent. This ALLL agent had simple task. It should come to tome then perform 10 measurements of temperature and then return back. This experiment is quite minimalistic but at this stage it is enough as a simple proof-of-work. For more about this experiment see

[19].

A. Pure JAWS Services

Every service provided by JAWS is described by previously mentioned ontology. Now we describe parameters used for description of service.

- **type** – This field represents type of service (Pure JAWS vs. ALLL).
- **name** – Name of service should represent its purpose (i.e. topology).
- **ontology** – Ontology can represent some additional parameters for each service.
- **periodicity** – Some services are inherently periodical (i.e. “all data from WSN” - when come any data from WSN they are resend to agent that subscribes this service). Some other services can be set up as periodic explicitly (mainly those where agent or message for agent is sent to network).
- **period** – How often should be service provided.
- **address** – Mote address to send possible agent
- **agentSlot** – Slot on mote for agent (default is 0, up to 4 agents).

First of Pure JAWS Services is service **Data**. This service sends ASA data from network. What data is resent is decided according to parameters of service. Type and name of service are obvious. Ontology in this case represents type of data that should be resent from Wireless Sensor Network. For example string “temp” stands for getting temperature data. When left blank all data are resent. This service is inherently periodical so fields periodicity and period doesn’t have meaning here. When address filled with -1 it means that data should be resent from any mote. Numbers 2 and higher (address 1 is reserved for Base Station) mean address of mote it wants data from.

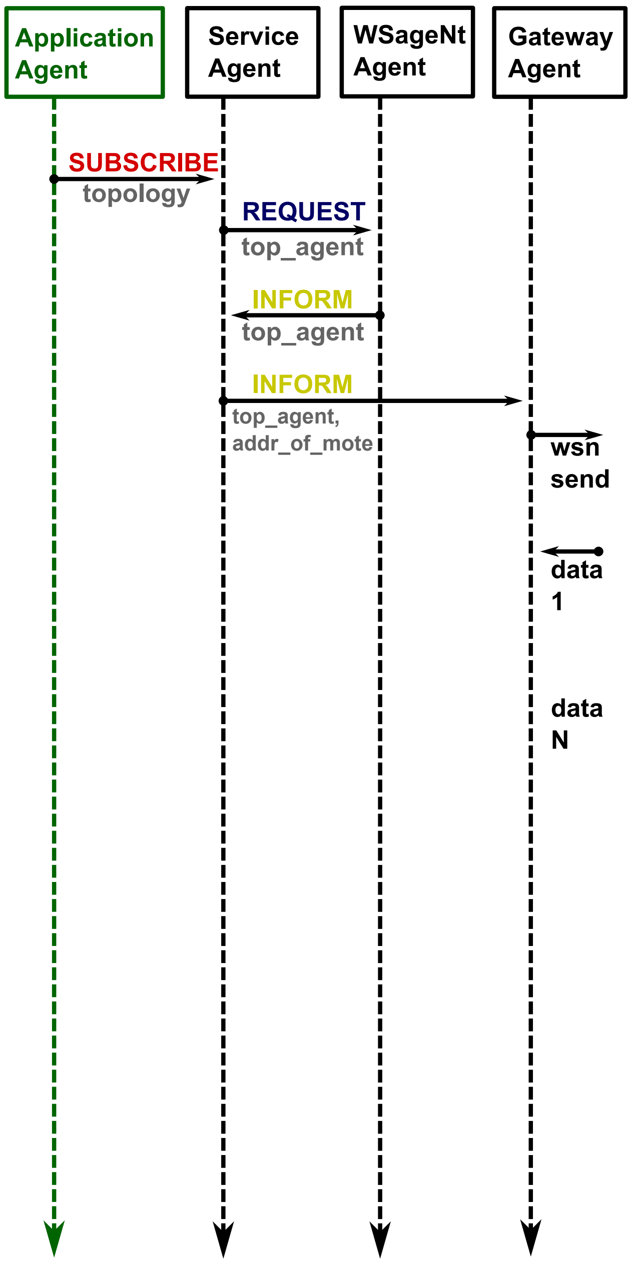
Next Pure JAWS Service is service **Limit**. This service sends notification when limit in incoming data is reached. Ontology in this service represents limit, whose overstepping is guarded. Before should be added one of characters $\{<,>\}$ that determine, which limit side should be guarded. Periodicity is solved in the same way as in Data service. The same situation is in the case of address and agentSlot.

Topology

Topology measurement is one of the key services in our system. By obtaining topology we are able to provide one service that is vital for larger networks. This service is routing inside WSN. Our routing algorithm uses counted topology for creating lattice. Through the lattice is consequently ALLL agent driven to its desired location but this description is rather simplifying and for more information about our routing algorithm see[16]. Topology counting has also other interesting features. One of them is visualisation of the network. The visualisation can be seen by possible user of the network if he does not have access to the deployed sensors. This might be for example in some experiment with fish when sensors are deployed in the water and there is no access to them. In this section will be explained topology counting from two points of view. The First is JADE agent’s point of view. This communication is depicted on Fig. 2 where is shown sequence of messages that is sent among group of JADE agents and there are also shown messages to/from WSN. The second point of view is part of the sequence diagram where are messages aimed to WSN nodes. This algorithm in WSN will be briefly also explained in this section. Firstly, we will describe sequence diagram. On the left side highlighted green there resides Application Agent (further AA). This is the only one agent in the diagram that needs to be created by application programmer. There can be seen that this agent only needs to send one message to obtain topology of the whole network. This task can be quite time consuming so for the AA can be more reasonable to do meanwhile something else. In JAWS is first agent in the chain Service Agent (further SA). SA provides service of topology discovery. So when requested it asks WSageNt Agent (further WA) for proper discovery ALLL agent for the mote. WA is the only agent that is aware of ALLL agents in the system, that is why he is in the chain of messages. When is proper agent picked up it is sent through Service Agent and Gateway Agent to some networks mote. What happens on that mote is described in some close future paragraph that deals with WSN part of the topology discovery service. Important is that WSN part ends up when comes back agent from the network. At this stage it is necessary to point out that Gateway Agent possibly sends other data from network by the time that **data1** to **dataN** arrive from network. It is Service Agents responsibility to recognize and sort messages properly. Luckily we have specific class id for agent that is created for getting network topology so we are able to recognize messages for the right protocol. When ALLL agent arrives from network that is signal for Service agent that it can count topology from incoming data. Once topology is count in the form list of coordinates `[x,y]` in virtual space it can be sent to Application Agent and the service by this message ends. We should mention that there is slight possibility that topology ALLL agent does not come from the network back (e.g. mote where it is present runs out of battery). In this case we set up timeout for Service Agent. After that timeout, Service Agent decides that something terrible happened and sends `failure` message to AA. Timeout must be set quite high in this case and if we have some clue about WSN size we can update this timeout accordingly. Now we get to the part of explanation where we show what happened in previous conversation on the Wireless Sensor Network side. We show quite simple pseudo code algorithm that shows what is the life cycle of topology ALLL agent on a mote.

```
store(origin_mote,flash_memory);
broadcast(show_yourselfs);
obtain(list(id_neighbour, sig_str));
create(slave);
send(slave,home,neighbour_list);
wait(slave,here);
find(next_mote,neighbour_list);
  OK: move(next_mote);
  NO: move(origin_mote);
```

As you can see first agent stores on flash memory of the mote information about its class and previously visited mote. Then agent sends broadcast message to discover motes within radio signal. When all motes react agent obtains list of visible motes and how strong is signal to each mote. Now agent creates other simple ALLL slave agent that should do simple thing for him. It should get back to the Base Station where it should deliver list of neighbours (and signal strengths) of mote. Implicitly it delivers also information about mote id. Slave agent is able to find path back home because of the information (in flash memory) about previously visited mote by agent with class of its masters. If on every mote is information about previous one, agent is able by sequence of hops get ultimately back to the BS. It also uses the same mechanism as its master when visiting motes. On each mote writes to flash memory information about previously visited mote. When slave give to BS all the information it carries slave travels back to its master. Now slave follows path that it created on the way to the BS. Ultimately slave reports to its master that information has been safely carried to the network sink and it is freed. This follows last part of the algorithm and that is moving of discovery agent to next mote. From the list of neighbours agent picks mote that has not been visited by it (there is no record about agent class and previously visited mote). When such mote is found agent moves to it. In the other case agent moves back to previous mote. From not visited motes on the neighbours list is randomly picked one neighbour. On the next mote algorithm repeats. When we look at the higher picture we see basically applied backtracking algorithm in WSN environment that ensures finding all motes within network. This algorithm uses principles of scent algorithms and Ant Colony Optimization algorithms to be able to find path within WSN. So we connected two well known algorithms to create algorithm that is able to gather data necessary for establishing information about topology of the network. Last part of the picture is processing of the data incoming to Service Agent. It is the part that is depicted on Fig. 2 as *count topology* in Service Agent. From the network came information about each motes list of neighbours with their signal strengths. Now we assign each signal strength relative value in the range `[0-255]`. We make average values for each pair of motes and now we have distance for each pair of motes. Now we can use trigonometry to count relative coordinates for whole network. We described more closely this problem in [18]. If for some reason topology ALLL agent doesn't come back to Base Station the whole service fails. For this reason we have set up timeout. That is the amount of time that Service agent waits for ALLL agent to get back from the network. At the end we should point out that protocol for obtaining topology is only specification of FIPA subscribe protocol that we describe in section IV-C but since **Topology** is JAWSs built in service it is present in every deployed system and therefore there is no need to refuse subscription. Refusing has meaning only in ALLL services when requested service is not present in the system.



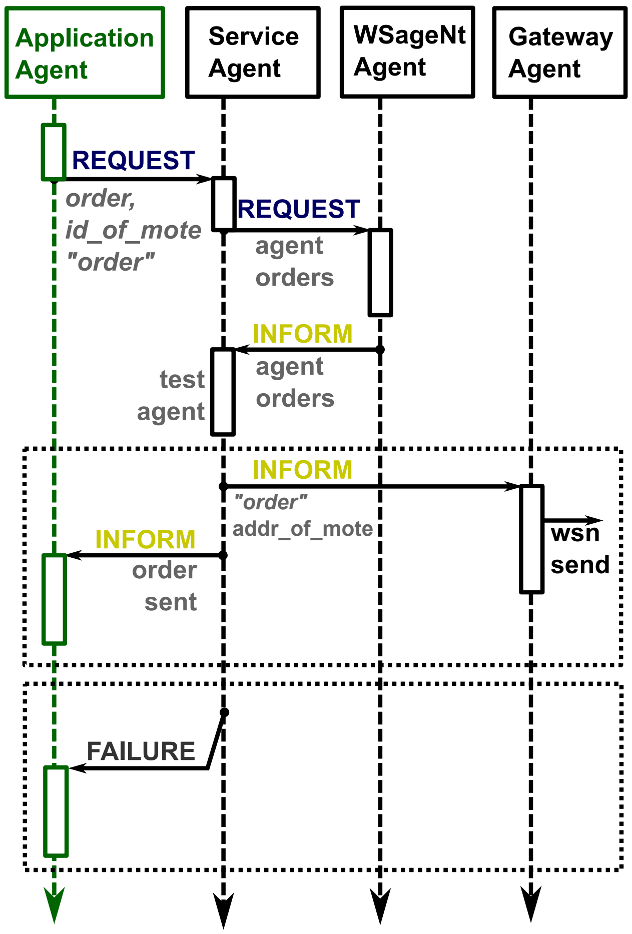


Fig. 3. Sending order to ALLL agent

the time of designing application and design of ALLL agents is in the hands of application programmer. In the future we want to help application programmer with creation of ALLL agents by providing methodology that will guide him through design phase when creating ready-to-deploy system.

Basically in our understanding ALLL agent is a plan for a mote to do something meaningful. In agency library of plans proved to be sufficient enough instead of creating plans from scratch. So we provide some default agents for known and frequently performed tasks. In next paragraphs we will describe protocol for using ALLL services. We try to outline situations that can occur during ALLL service use.

Since using ALLL service falls to the same situation as using Pure JAWS services we use again subscribe protocol. In this case protocol form is quite similar as in our previously stated example with topology measurement. In principle Application Agent again subscribes for service and then waits for data sent from network. There is only one change. It is possible that agent in network is not supposed to send messages (or itself) back to JAWS. In this case there are not incoming data from network as on Fig. 2. Application Agent only subscribes for service and then agent is sent and service ends by this step. This might be the case when ALLL agent should for example go through whole network and inform agent on each mote about some fact. As an example we might say that we

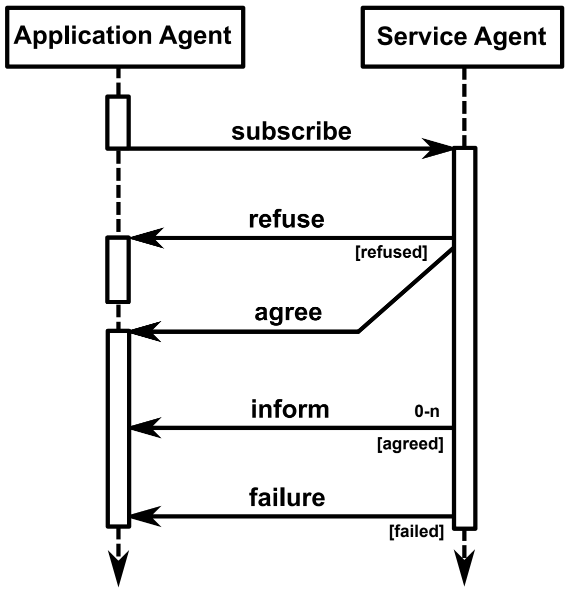


Fig. 4. Subscribe Protocol

need to lower transmitting power of the radio module for each mote. We can send agent that orders each mote platform about lowering its radio power.

There is one other situation when we send only message to network without waiting for reply. Since ALLL agents are able to react on incoming messages we need a protocol in JAWS that enables Application Agent to send order to agent on mote. This situation is depicted on Fig. 3e. In this situation is firstly send ACL message with *REQUEST* performative to Service Agent. In contents of this message must be id of mote to send order to (including *agentSlot*) and of course *order* itself. Each ALLL agent has list of orders (strings) that it is able to react to when it comes to its input base.

Service Agent keeps list of textitagent_on_mote that is the list that stores information about mote occupancy with agents. When request from AA comes Service Agent checks this list and finds out what agent in on requested mote on specified slot. Subsequently Service Agent asks WSageNt Agent if ALLL agent is able to react to given order. If this test goes through SA sends through Gateway Agent order to requested mote. Otherwise *FAILURE* is sent to AA. Now two situations can occur. When ALLL agent is supposed to response to order from AA it sends back some data (or itself). In the case when order does not imply some sort of reaction that leads to sending data back service ends by sending order to WSN.

C. Subscribe protocol

There are several used protocols in JAWS. Some of them are standard and some others are designed for specifically for JAWS. In this section we describe protocol used for subscription of service. This is standard *subscribe*[12] protocol as it is defined by FIPA organisation. Protocol is depicted in Fig. 4. Initiator is Application Agent and responder is Service Agent. Firstly, initiator creates object with ontology describing service it wants to use and sends ontology to responder. Service Agent checks whether ontology is filled correctly (according to rules indicated in previous sections). After this check it is able to respond to initiator. Sending *agree* or *refuse* message is voluntary but we use it because we need to have the opportunity to notify the user that service description was filled in the wrong way. When the message is filled correctly Service Agent sends agree and refuse otherwise. At this point service is subscribed for particular agent. Finally when comes proper data Service Agent resends them to Application Agent. Last portion of this protocol is deregistration of service. This works in the same manner as in the case of FIPA specification [12].

Since we follow FIPA standards we use Directory Facilitator as said before. Now we briefly describe basic principles that are used when registering JAWS services by DF. At the start of the system we register Pure JAWS services because they are changeless. Then Service provider obtains list of ALLL agents known by system. These agents are then registered as ALLL Services. Now, when services are exposed to DF, Application Agent can use prepared behaviour to obtain all exposed services then pick appropriate one and subscribe by Service Agent.

V. Problems and Solutions

In this subsection we describe some of the problems we encountered during experiments with our system. Biggest problem of all is the problem with data incoming from the network. There are two possible data types that are able to come from the network. First data type are data that are sent by an ALLL agent that resides on mote with some **id** (address). This data come in the form of tuple. Example is shown bellow.

```
((5),((1,12.0)(2,12.5)(3,13.2)(4,12.7)))
```

First value (5) is id of mote that sent the message. Rest of the message represents values measured by mote in some period of time. We can see that there are 4 indexed measurements. There are measured some temperature values and we can see that temperature was around 12C. There are two problems with this example. First problem is cosmetic. The temperatures shown in example aren't really coming from the network because mote measures and stores only resistivity of the proper sensor. This resistivity comes to Base Station and then must be recounted according to proper formula so we get real temperature (light, humidity, ...) value. Next problem is also related to incoming values from sensors. The problem is that if mote has more than one sensor we aren't able to decide which sensor are incoming data from. When only resistivity of sensor comes in message we don't know what sensor performed the measurement. That implies we need another mechanism to recognize content of message from network. We describe partial solution of this situation in some close future paragraph. The other data type possibly incoming from network is whole agent. Since we have in packets information that this is agent message we can handle it properly. Service agent gets the information that came some ALLL agent. From ALLL agents fields we are able to obtain all necessary information. Most important for us is **mote_of_origin** that is mote in WSN that ALLL agent came from. Next useful information is agent name (as a service name, described in IV). We can also use **agent_id**, **agent_class** (topology example) and basically any of the fields of ALLL agent. For data we exploit agents **belief_base**.

Simulation: We can categorize ALLL agents into two kinds by the mechanism they measure values and then send informations into Base Station (JAWS). In first case agents can measure some values periodically and after measurement they send data (or itself) into Base Station. In the other case they measure something (probably periodically) and in the case of some event they send message to JAWS. Typical example is our service **Limit**. In this service if limit of some value is reached or overstepped mote sends message about this event to JAWS. In the first case we can obtain information about agent time plan (it sends messages periodically so we have information that in some specific timeslots comes message with specific value). This looks simple. It is enough to store period and we know when a message with data arrive. But it is not that simple. Since WSN is highly distributed system it is possible that agent on mote will be asked to stop measurement for a while and do something else for other agent on some close mote. Our period has stayed the same but it has shifted for a small amount of time. Now time scheduling (in JAWS) of message sending is broken and in JAWS we have no idea what might happen. In the end we don't actually know what really happens in the WSN. We just know what should happen. So we proposed another solution. We simulate the whole WSN on personal computer (Fig. 5). We named our simulator **GDEFALL**, which is abbreviation for **G**raphical **D**esktop **E**nvironment **F**or **ALL**L agents. This tool enables us to simulate in real time whole deployed network. Current state-of-art of the simulator is following. At the beginning we create WSN (in simulator) by hand (we have already known topology) and start simulation at the same time WSN motes start. At the beginning we populate both *real* and *simulated* WSN with ALLL agents. This task is in the hand of Application agent. It subscribes for ALLL service on each mote. Now almost everything what happens in real network also happens in simulated network. Also both networks send its messages to JAWS. So, when message comes from simulated WSN we have information that the same message (well not actually the same, but with the same mote id) should come from real network. If it does not come we know that something wrong happened (e.g. mote run out of battery). As a conclusion, simulation enables us to partially recreate sequence of messages as they should come from real network). This means that simulation helps us to lower a little bit degree of uncertainty in WSN. In the later case agent messaging is driven by event so we have no clue when a message might come. In this case simulation is not helping. Event is produced by real WSN environment, but we can't simulate this (do we know when temperature outside is up to some limit?) so all measurements in simulator does not produce values. Measurement is represented by storing some default value. In this part of communication, when message sending is triggered by event, we are unable to simulate sending messages from network to JAWS. That is why we stated before that in simulation happens **almost** everything the same way as in real WSN.

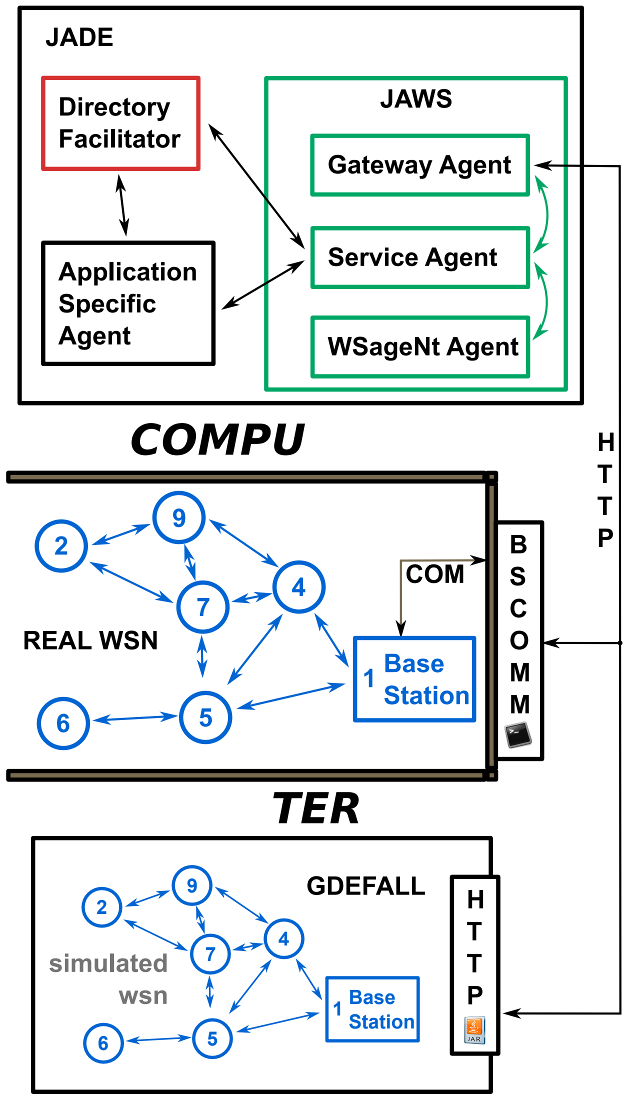


Fig. 5. Real time simulation of the observed network

VI. Conclusion

In this paper we briefly described our system called JAWS. We also introduced motivation behind creating such a system. Then we described architecture of the system from the point of view where services played main role. At last, we presented services and their usage in our system. With services are also connected protocols. We described some of the main protocols used in our application. We also presented some open problems a some partial solutions that we performed to lower uncertainty of data coming from the network.

In conclusion we created system that is able to control WSN. This system highers the abstraction, which we use to operate WSN and also make manipulation with WSN motes easier.

VII. Future Work

In the future we want to create whole methodology that will guide application programmer through the whole process of creating application specific system. This methodology should be supported by JAWS and possibly other created tools including presented simulator. In our methodology we set up goal to safely guide application programmer through series of steps that will lead to working system that is ready-to-deploy. Our methodology will consist from two interconnected phases. In one phase it is necessary to create agent system from JADE agents. Here we need to specify what will created system do. In this phase WSN is abstracted and motes are understood as sensors (in agent meaning of this word) of JADE agent. This is first part of the problem. In second part it is necessary to pick ALLL agents from library of agents or create application specific agents. In our understanding each ALLL agent represents "plan" for one mote. Application Agent can use these plans for motes.

Acknowledgment

This work was supported by the European Regional Development Fund in the IT4Innovations Centre of Excellence project (CZ.1.05/1.1.00/02.0070).

References

[1] Mainwaring, A.: Wireless Sensor Networks for Habitat Monitoring; Communications of the ACM; 2002/6; vol. 47, issue 6; pp. 34–40; ISSN 0001-0782.
[2] Fok, Ch., Roman G., Lu Ch.: Agilla: A Mobile Agent Middleware for Self-Adaptive Wireless Sensor Networks; ACM Transactions on Autonomous and Adaptive Systems; 2009/7; vol. 4; issue 3; p 26; ISSN 1556-4665.
[3] Zboril, F., Horacek, J., Spacil, P.: Intelligent Agent Platform and Control Language for Wireless Sensor Networks. In: Proceedings of 3rd EMS; Atny; GR; IEEE CS; 2009; p. 6; ISBN 978-0-7695-3886-0.
[4] Zboril, F., Spacil, P.: Automata for Agent Low Level Language Interpretation. In: Proceedings of UKSim 2009; Cambridge; GB; IEEE CS; 2009; p. 6; ISBN 978-0-7695-3593-7.
[5] J. Hill. System Architecture for Wireless Sensor Networks. PhD thesis, UC Berkeley, 2003.
[6] Welcome to the foundation for Intelligent Physical Agents; [online] [cit. 2012-06-02]; http://www.fipa.org
[7] Bellifemine, F., Poggi, A., Rimassi, G.: JADE: A FIPA-Compliant agent framework. In: Proceedings of Practical Applications of Intelligent Agents and Multi-Agents; 1999/4, pp 97-108.
[8] Padgham, L., Winikoff, M.: Developing Intelligent Agent Systems: A Practical Guide. John Wiley and Sons, 2004. ISBN 0-470-86120-7
[9] FIPA Agent Management Specification; [online] [cit. 2012-06-02]; http://www.fipa.org/specs/fipa00023/XC00023H.html#_Toc526742640
[10] What is an Ontology?; [online] [cit. 2012-06-028]; http://www-ksl.stanford.edu/kst/what-is-an-ontology.html
[11] Horacek, J., Zboril, F.: WSageNt: A case study, In: Proceedings of CSE 2010 International Scientific Conference on Computer Science and Engineering, Kosice, SK, 2010, ISBN 978-80-8086-164-3.
[12] FIPA Subscribe Interaction Protocol Specification; [online] [cit. 2012-06-29]; http://www.fipa.org/specs/fipa00035/SC00035H.html
[13] Janousek V., Koci R., Mazal Z., and Zboril F.: PNagent: A Framework for Modelling BDI Agents using Object Oriented Petri Nets, In: Proceedings of 8th Conference on Intelligent Systems Design and Applications ISDA08. IEEE Computer Society, 2008, pp. 420425.
[14] Woolridge, M., Hbner, F.J., Bordini, H.R.: Programming Multi-Agent Systems in AgentSpeak using Jason, Chichester, West Sussex: John Willey & Sons Ltd., 2007. pages 273. ISBN 978-0-470-02900-8 (HB).
[15] Koci. R., Zboril, F., Zak, J.: Integrating Multiple Modeling and Development Tools for Realization of Distributed Intelligent System, In: Proceedings of the 10th International Conference on Intelligent Systems Design and Applications, Cairo, EG, IEEE CS, 2010, s. 658-663, ISBN 978-1-4244-8135-4
[16] Horacek, J., Zboril, F., Hanacek, P.: Agent Aided Routing in Wireless Sensor Networks, In: Proceedings of CSE 2012 International Scientific Conference on Computer Science and Engineering, Kosice, SK, TU v Kosiciach, 2012, s. 119-126, ISBN 978-80-8143-049-7
[17] Wang, K. I., Abdulla, W. H., Salcic, Z.: A MULTI-AGENT SYSTEM FOR INTELLIGENT ENVIRONMENTS USING JADE; IEE Seminar Digests; 2005; vol. 2005; no. 11059; pp 86-91.

[18] Zak, J., Horacek, J., Zboril, F., Koci, R., Gabor, M.: Remote controlling and monitoring tool for wireless sensor network using WSageNt platform, In: Proceeding of the 2nd International Conference on Computer Modelling and Simulation, Brno, CZ, UITS FI VUT, 2011, s. 1-8, ISBN 978-80-214-4320-4

[19] Zak, J., Zboril, F., Hanacek, P.: Jade Agents to Wireless Sensors: Case study, In: Proceedings of CSE 2012 International Scientific Conference on Computer Science and Engineering, Kosice, SK, TU v Kosiciach, 2012, s. 95-102, ISBN 978-80-8143-049-7

[20] Zboril, F., Koci, R., Zboril, F., V., Janousek, V., Mazal, Z.: T-Mass v.2, State of the art, In: Second UKSIM European Symposium on Computer Modeling and Simulation, Liverpool, GB, IEEE CS, 2008, s. 6, ISBN 978-0-7695-3325-4

[21] Zak, J., Horacek, J., Zboril, F., Koci, R., Kral, J.: JADE Agents Used for Wireless Sensors Control: System Based on Services, In: Proceedings of the 2012 12th International Conference on Intelligent Systems Design and Applications (ISDA), Kochi (Cochin), IN, 2012, s. 6, ISBN 978-1-4673-5118-8

Author Biographies

Jakub Zak (born 1985, Dacice) is leading author of this paper. This author obtained his Master's degree in 2009 at Faculty of Information Technology. Author is now student of Ph.D. study program on Faculty of Information Technology at Brno University of Technology, Czech Republic. He is studying his final year of his Ph.D. studies. Among his interests belongs all related to agent oriented technologies and modelling systems. Additionally author is interested in methodologies that deal with agent systems creating.

Jan Horacek (born 1985, Jihlava) is a Ph.D. student at Brno University of Technology, Czech Republic. He obtained his Master's degree in 2009 at Faculty of Information Technology at University in Information Technology. His research is aimed to wireless sensor networks, artificial intelligent agents and routing protocols.

Frantisek Zboril jr. (born 1974, Olomouc) is an assistant professor at Brno University of Technology, Czech Republic. He obtained his Ph.D. In 2004 at Faculty of Information Technology of this University in Information Technology. His major interests include artificial agents, their application in the area of modelling distributed systems and applications for wireless sensor networks.