Architecture for WSN nodes integration in context aware systems using semantic messages

Iker Larizgoitia, Leire Muguira and Juan Ignacio Vazquez

MoreLab Research Lab, DeustoTech, Avda Universidades 24 48007 Bilbao, Deusto, Spain,
ilarizgo@tecnologico.deusto.es, lmuguira@tecnologico.deusto.es,
ivazquez@eside.deusto.es,
Web site: http://www.morelab.deusto.es/

Abstract. Wireless sensor networks (WSN) are becoming extremely popular in the development of context aware systems. Traditionally WSN have been focused on capturing data, which was later analyzed and interpreted in a server with more computational power. In this kind of scenario the problem of representing the sensor information needs to be addressed. Every node in the network might have different sensors attached; therefore their correspondent packet structures will be different. The server has to be aware of the meaning of every single structure and data in order to be able to interpret them. Multiple sensors, multiple nodes, multiple packet structures (and not following a standard format) is neither scalable nor interoperable. Context aware systems have solved this problem with the use of semantic technologies. They provide a common framework to achieve a standard definition of any domain. Nevertheless, these representations are computationally expensive, so a WSN cannot afford them. The work presented in this paper tries to bridge the gap between the sensor information and its semantic representation, by defining a simple architecture that enables the definition of this information natively in a semantic way, achieving the integration of the semantic information in the network packets. This will have several benefits, the most important being the possibility of promoting every WSN node to a real semantic information source.

1 Introduction

In recent years the use of WSN is moving forward from the mere monitorization of certain environmental variables to its widespread adoption for context aware systems. The reduction in the price of these devices and their increasingly powerful features, though limited compared to a normal PC, are still enough for some scenarios unaffordable years ago to be realized.

WSN are highly qualified to provide a great deal of information on a realtime basis. However, seldom does this information follow standard formats and it is usually dependent on the type of network and sensors used. Because of this, there will always be a process by which the sensor information is preanalyzed in order to extract its real meaning for the system, before it can be used in a context aware manner (e.g. converting an n-byte raw data packet of a concrete noise sensor into a statement that declares the environment as noisy).

The research presented in this paper has been focused on how to provide a framework that enables the use of semantic-like definitions using ontologies in the network packets themselves. Taking as a prerequisite that the information has to be semantically defined from the very moment it leaves the sensor node, the aim is to find a way to use semantic structures, such as triples or statements in an affordable way, that is, not causing too much overhead in the network, at least not one that is not likely to be accepted in some application scenarios. Using semantic information at sensor-data level can contribute to advances in some of the current challenges in this kind of systems [1], such as facilitating the exchange of information among nodes, the integration of different kinds of sensors with minimum effort and the introduction of some intelligence in the WSN nodes, not just using them as a source of information. The application field that we have chosen is the area of smart objects. The idea is to create intelligent environments by integrating these small nodes and sensors in real-life objects. This field has great potential for the development of innovative applications, more precisely all those related to activity based systems.

The rest of the paper is organised as follows. Section 2 presents some related work in this area. Section 3 describes the mechanism used to adapt and represent the sensor node messages as semantically defined messages. Then, Section 4 describes how this mechanism can be integrated in a more comprehensive context aware platform. In Section 5 the application scenario that has been developed to test this mechanism is detailed. The objective of this scenario is to illustrate how sensors can be integrated in a more flexible way following these mechanisms. Finally, Section 6 presents the conclusions and the future work of this research.

2 Related Work

Current research in WSN suffers from some major drawbacks. The overwhelming heterogeneity of sensors, the lack of a common development framework or an application level communication standard are among the biggest challenges for future research projects [2]. More generally, context aware applications present some other technical challenges informationwise [3]. One of the most important concerns is that sensor data, high-level context information, actuators and human interfaces need a standard representation.

In the attempt of converting this sensed information into a meaningful dataset, different approaches might be taken. Until now, the most popular was having a central server or a gateway that is able to interpret sensor information and then transform it into a meaningful representation for the rest of the system to use it in a higher level [4] [5]. Some inniciatives, like the OGC Sensor Web Enablement [6] try to define a comprehensive framework to enable the capture, discovery and use of sensor information through the Web. However, this approach present some problems, because it does not use a formal ontology for data definition and the communication paradigm is based on passive services with interfaces (pulling mechanisms) [7].

Besides, the role of sensors is moving on to a more sophisticated scenario, where the information they provide is directly understandable. For example, in [8], WSN nodes can evaluate their sensed information using some schemas, which enables them to work at event level, not just sending raw packages. The use of rule-based systems in the WSN nodes is also another alternative. Systems like [9] or [10] define rule languages to represent the information a sensor provides and how it has to react to certain conditions.

While these and other approaches manage to define the sensed information at a higher level, still suffer from the lack of standardization. However, it seems like the next steps in this field are directed to use semantic technologies at different levels in a Semantic Sensor Network infrastructure [11] [2]. Semantics can be applied at different levels, with different objectives. Our research focuses on sensor-data level. Working at this level, a first approach is to adapt the sensor information to a semantic representation after it has been captured [12]. Providing that sensed information needs to be adapted to the semantic world, it would be better if that conversion is carried out as soon as possible. This usually includes a definition of the information with some semantic knowledge representation mechanisms, such as OWL (Ontology Web Language) [13] or RDF (Resource Description Framework) [14]. Any of these mechanisms (and others alike) defines a basic framework to share and model the knowledge of any system, but their use in the WSN part of a system is problematic. First, this kind of representations is computationally expensive for the nodes to process them internally, due to their limited capabilities. Working with XML in a WSN node is almost impossible or at the very least, inefficient. Regarding this aspect, there will always be a trade-off between the level of representation and the overhead it might produce in a network. The more self-descriptive is a message, the larger size it will have.

3 Semantic Adaptation

A typical application of WSN for context aware systems basically consists of two steps. First, the information from the sensors is sent to an entity, usually acting as a context manager, which has the responsibility for interpreting the different types of messages. Then that information has to be transformed into some meaningful representation for the rest of the system. Taking this into account, the general objective of this research is to merge these two steps into a more general one. One that will integrate the semantic information in the network packets, saving time in the conversion process but also gaining flexibility sensor informationwise because if a new node with new sensors wants to share its values it just needs to represent them semantically to make them understandable for whoever is in the network. To achieve this, three main aspects have to be defined:

- A mechanism to represent the information semantically (Section 3.1).
- A packet structure that could fit in a WSN platform, considering the limited characteristics of these environments (Section 3.2).
- A protocol to share this semantic information (Section 3.3).

3.1 Information representation

Semantizing any information could be seen as the process of formalizing its representation so that it can be easily understood by many participants with no other prior knowledge than the formal representation itself. Currently the most common semantic representation mechanisms are RDF and OWL, being OWL able as well to be represented using RDF.

RDF is a knowledge representation mechanism based on resources and statements that can be said about those resources. Basically a statement links two different resources in a structure of the form subject-predicate-object, usually called triples. These triples are the base of the entire model; therefore they are the smallest structure needed to represent knowledge.

These triples can be represented using different formats and syntaxes such as N3, N-Triple or RDF/XML. Unfortunately these representations are not suitable for a WSN environment because the limited devices do not have the needed resources for them. Bearing in mind the concept of triple, which is how the information from the different sensors should be modelled, the next step is to define the prerequisites of the representation mechanism for the WSN messages:

- The basic structure to share in the network is a triple.
- A triple should reference resources that anybody could be able to understand. The information provided by the different sensors is then supposed to have been previously formalized in any kind of knowledge representation, for example an ontology.
- The representation of those triples should be adapted to fit into the WSN environments, so it has to provide some compression or codification mechanism to reduce the size of the network packages as much as possible.

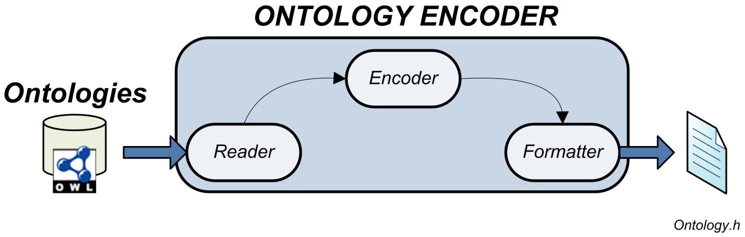


Fig. 1. Ontology codification architecture

Codification of the ontology information Once it has been established that the information is going to be represented as triples, the specific representation has to be analyzed. Following the RDF model concepts, each part of the triple will be represented with a URI. A URI is usually too long a string to be used in a WSN packet directly. So it might be a good solution to try to find a way to reduce those URIs without losing expressivity. This is where a codification scheme can be used. A codification scheme is just a function (see 1) that given a long URI converts it to a simplified representation, where the length of the code generated is much smaller than the original value. This function could be reversible, but it does not need too, as long as the codes are conveniently stored.

$$f : URI \rightarrow code \,|\, legth(code) \ll length(URI) \tag{1}$$

The idea is to encode every single term in the ontology, that is, the different URIs in a simplified representation of each one applying the codification function. These codes should then be used in the correspondent implementation to create the packets the node is sending to the network. Creating the codes for all the terms of the ontology might be a hard task. However, it can be easily automated. For this purpose a simple codification architecture has been defined. Providing the ontologies to the system and defining the concrete codification scheme different output formats can be generated. For example, in the prototype described later (see Section 5) the WSN is based on tinyos (nesC programming language) so the list of the codes is automatically formatted to a C header file. Figure 1 shows this general architecture for the codification of an ontology following a generic component model. The responsibilities of each component are as follows:

- *Reader*: this component is able to read OWL files and parse them, extracting every term (URI) in the defined ontology.
- *Encoder*: this component implements the correspondent codification scheme, basically it has to receive all the terms (URIs) of the ontology and encode them accordingly.
- *Formatter*: this component gathers all the encoded terms and formats them in a predefined format. The output format depends on the programming language to be used in the WSN.

3.2 General packet structure

The next step is to define how the packet structure must be to hold the semantic information. For this purpose at least three control fields are necessary:

- *Codification Scheme*: the codification algorithm used in the triples.
- *Number of Triples*: the number of triples that are contained in the packet.
- *Ontology URI*: a reference, using the correspondent codification scheme, to the URI of the ontology the collection of triples refers to.

Besides these general fields, the packet must contain a group of triples, each of which defines a new statement about the context information the node shares in the network. For each part of the triple, it can be intuitively deduced that at least its size (in bytes) and type (whether it is a resource or a literal value, etc.) are needed in order to completely define the packet. This general packet structure is shown in Figure 2.

Packet structure optimization Each part of the triple does not have a fixed size because their content depends on the codification scheme used in the packet. Assuming that the content is basically a group of statements and analysing how they would be represented, the packet structure for each part (subject-predicate-object) can be optimized. Instead of having a type and size field for every part of the triple, some of these values can be deduced from the codification scheme used, as well as the possible alternatives for each part.

The value of a *subject* represents the resource the statement describes, so two situations are possible. The resource is already defined in the ontology, having therefore a predefined known URI or the resource the statement describes is not defined yet, so a unique identifier must be provided, for which a UUID is an appropriate solution. Considering these aspects, the subject size field can be removed, because that information is inherent to the codification scheme used, which needs to be necessarily known.

A *predicate* is bound to be a URI, because it has to be a valid OWL property (either ObjectProperty or DataProperty) predefined in the ontology of the domain. Therefore, in the predicate part the type and size can be eliminated, because that information is also deduced from the codification scheme.

For *objects*, instead, there are more options, because, apart from the resource URI and UUID, objects can be literals which refer to any XSD valid type. To support different sizes of objects while trying to optimize the packet size, the object size is an optional field which can be included when the object corresponds to a variable XSD type, for example, string, but ignored when the type is an encoded URI or a UUID.

Codification example: Hash code Now it will be illustrated how the packet structure would be for a concrete codification mechanism, selecting a simple hash codification scheme. For illustrative purposes we have just used the algorithm of the Java String class. This simple hash codification would just take the URI as a string and compute it with the formula shown in Eq. 2, where h is the URI string to transform:

$$h(s) = \sum_{n-1}^{i=0} s[i] * 31^{n-1-i} \tag{2}$$

With this formula, each encoded value has 32 bits, with which it can be defined the packet structure. Figure 3 shows the packet structure when this simple hash algorithm is used.

3.3 Network protocol

Once the semantic packet structure is defined, it has to be specified how this information is going to be shared in the network. Strictly speaking, what has been defined is not a packet structure but a payload that could be part of a network packet of almost any

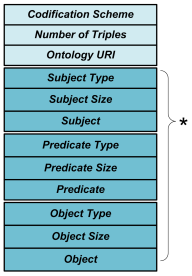


Fig. 2. General semantic packet structure

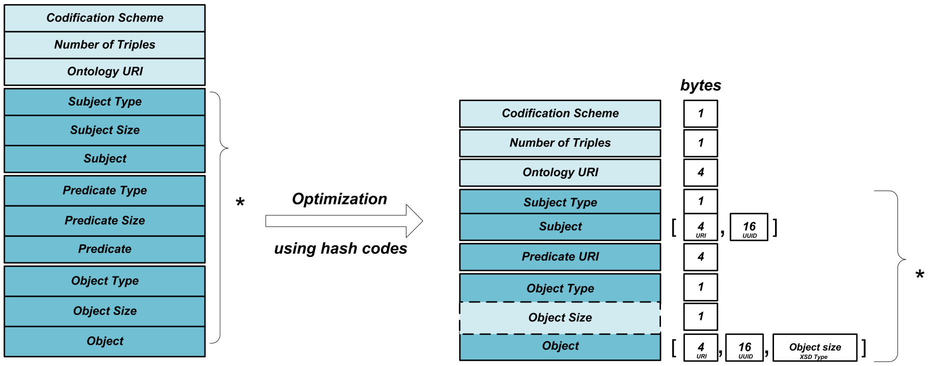


Fig. 3. Optimized packet structure with hash code

WSN protocol or architecture. If a central server is present, a collection protocol might be more appropriate than a P2P approach, for example. Regardless the network protocol used, this semantic payload can still be used and properly analysed by the correspondent entities. As this mechanism is thought to be part of a WSN, the network protocol should at least provide the means to create a mesh network and a reliable mechanism to send the node packets. The selection of this protocol is implementation specific and might vary from one application to another.

4 Context Management

The mechanism for sharing the semantic information of the nodes has been defined but it needs to be integrated into a more comprehensive context aware system. There are many design alternatives for the architecture of a context aware system as well as for how the information is used in the system. A very challenging application field of context aware systems involves the integration of small WSN nodes in real life objects, what might help improve activity-based systems. Context-aware systems are usually divided into three main parts: capture, reasoning and reaction (or proaction). To integrate these functionalities, a general architecture has been defined, in which the WSN nodes are seen as part of the capture phase and are seamlessly integrated as a knowledge source for the reasoning part of the architecture. The reactivity model is generally based on behaviours, which are components that can monitor the knowledge base and react accordingly to the state of the context at a given moment. Figure 4 defines the general architecture of the context-aware platform, following a component model. The responsibility of each component is as follows:

- *WSNDriver*: This component acts as a driver for the entire WSN infrastructure to be connected to the context management platform. This includes the implementation of the packet listeners through the corresponding hardware device acting as a gateway. The concrete hardware platform will be explained in the prototype scenario. The objective is to define a component that is able to receive any kind of WSN packets based on plugins for the different options or transports and share them with the rest of the architecture.

- *Ontology Management*: This component implements the mechanisms to decode the semantic messages based on the ontologies that have previously been loaded or defined. It provides an interface for converting the codes in the full URI of the term being represented in the triple.
- *RDF Converter*: the semantic information provided to the reasoning component needs to comply with a standard format. At this level, one of the more flexible alternatives is RDF, so this component just transforms the triples (subject-predicate-object) into its RDF representation and prepares them to be injected in the reasoning capabilities of the platform.
- *Reasoner*: The range of reasoning techniques that can be applied in a context-aware system is extremely wide, mostly different AI strategies (rule engines, semantic reasoning, etc). To take advantage of the semantically defined information in the prototype (Section 5) an ontological reasoner has been included, which can be augmented with other techniques to add more reasoning capabilities to the system. As the reasoning is decoupled from the capture of data, it will not be difficult to add other reasoners with different techniques and coordination mechanisms.
- *Behaviours*: the behaviours define how the system is going to react when certain conditions are met (based on new knowledge, either added or inferred). These components should be notified when new knowledge is inferred, in order to evaluate the conditions of activation. Accordingly, the behaviours could be able to query the reasoner for other kind of information retrieval. These behaviours have to be defined for each concrete reactivity scenario. This approach has been adopted because while using a semantic reasoner as a central unit, every behaviour can add any other desired extra reasoning capabilities to its concrete purpose.

4.1 System general operation

Figure 4 shows the different steps that are carried out in a common operation of the context management server:

1. The ontology codification module preloads the ontologies that the semantic messages are going to use and exports its interface to let other components decode ontology URIs from the WSN packages.
2. The WSN driver is physically connected to a node and receives the semantic packages following the previously defined protocol and structure.
3. These packages are then handed to the RDF Converter component which decodes the URIs and content of every packet and transforms them automatically into a valid RDF representation.
4. The Reasoner component receives the RDF information and merges it in the Knowledge base, ensuring that the ontology remains consistent, as well as classifying the ontology again to generate new knowledge.
5. When new knowledge is inferred, the Behaviours have to check whether its conditions are met in order to carry out some reaction.

6. Besides, behaviours can interact directly with the Reasoner if they need to extract or update knowledge themselves.

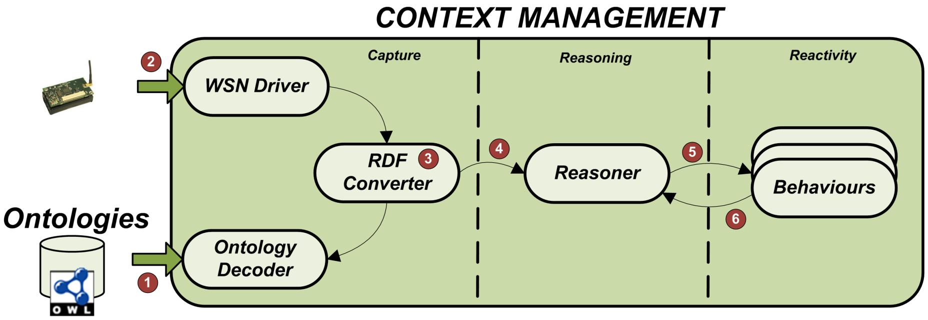


Fig. 4. Context management general architecture and operation

5 Prototype

To test the semantic messages architecture a prototype has been implemented following the structure defined in Section 4 for the context-aware system.

5.1 Hardware platform

For the hardware platform, there are different WSN platforms in the market. For this prototype the Crossbow family of WSN was analyzed. Crossbow[1] motes are a family of embedded sensor nodes sharing roughly the same architecture. There are many different kinds of Motes. Table 1 summarizes the analysis that was made in order to select a concrete platform. The main versions (most used up to now) are compared, presenting their principal characteristics. Finally the platform MicaZ was chosen. Even though iMote2 was better, technologically speaking, MicaZ was more accessible, cheaper and had a smaller OEM version, which is more likely to be integrated in real applications. The hardware platform is programmed in nesC language, so when the ontology codes are generated, the most appropriate format is a C header file with the correspondent codes as constants. Table 1 how this file looks like when formatted as a C header using a simple hash code codification scheme. The constants have a first part with the short-name of the ontology (e.g. SMARTMOTES_OWL) and a second part with the name of the property or instance defined in the ontology.

With this helper file, the developer can then create the correspondent triple objects according to the information that the node is going to share in the network.

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

Table 1. Example C Header of the encoded ontology

```
#ifndef SMARTMOTES_ONTOLOGY
#define SMARTMOTES_ONTOLOGY
enum
{
SMARTMOTES_OWL_ONTOLOGY_URI 0̄x5b9f59f9,
SMARTMOTES_OWL_HASDIRECTION 0̄xac5f284f,
SMARTMOTES_OWL_RIGHT 0̄xa747a2f2,
SMARTMOTES_OWL_STANDING 0̄x2d422796,
SMARTMOTES_OWL_UP 0̄x34e7c565,
SMARTMOTES_OWL_LEFT 0̄x9a07c851,
SMARTMOTES_OWL_DOWN 0̄x9a044cec,
SMARTMOTES_OWL_UNKNOWN 0̄x9b60aba0,
SMARTMOTES_OWL_SHAKEN 0̄x7bbdd7f2,
SMARTMOTES_OWL_MEASURE 0̄xb97d18d4,
...
...
}
#endif
```

5.2 Network Protocol

The network protocol defines how the information is spread in the WSN. The semantic package is going to be part of the payload, so the network protocol can add any headers needed. For the prototype a collection protocol has been chosen. It fits the purpose of gathering information from the sensors, sending packets with the semantic information, and collecting the data generated in the network into a base station, which is connected to the context management system [15].

5.3 Software implementation

For the implementation of the server architecture the OSGi platform has been chosen [16]. OSGi provides a perfect platform for developing applications where multiple components interact in a dynamic way. OSGi will also give us scalability and a better manageability of the life cycle of the different components, which in OSGi are called bundles. There are several implementations of OSGi R4, the latest version. Equinox, the platform behind Eclipse is the alternative used in this prototype. This platform has an active community support and almost every optional service defined in the specification is implemented and tested. Every component has been developed in an OSGi bundle, which exports or imports certain interfaces to communicate with the rest of the components. The implementations details of every component are the following:

- *WSN Driver*: uses the TinyOS 2.x infrastructure to receive the network packets. It is physically connected to a WSN base station via a serial com port.

- *Ontology Manager*: loads the different ontologies used in the application and implements the different codification schemes in order to be able to decode the semantic messages. This component used the Jena library [17] to read the ontologies and process them.
- *Converter*: this component receives the semantic messages, parses them and generates a standard RDF-XML representation that can be used as a direct knowledge source for the reasoner component.
- *Reasoner*: this component wraps a Pellet [18] reasoner associated to the ontologies the application is going to use. It can notify when new knowledge is added through its interface to the behaviours defined for the concrete application.
- *Behaviours*: these components implement the reactivity of the system, extending the capabilities of the ontological reasoner and defining how the system should react. In the prototype this behaviours just have a set of conditions that when are fulfilled some actions are executed, but more complex behaviours might be easily added in the future.

5.4 Application scenario

Context-awareness applications can be applied to many scenarios. Home, work or health-care scenarios are among the most popular ones. Using WSN embedded in real-life objects enables the development of new applications beyond the monitoring paradigm. For this prototype the basic scenario is focused on a child's play scenario, where not only the environment can be instrumented with WSN nodes, but also toys, books or similar artefacts children can interact with. Each object might share its context information using the semantic messages that have been described in this research.

The scenario is based on a puppet, inside of which a WSN node has been deployed (MicaZ platform). This node has several sensors, from which only two are going to be used to simplify the explanation, the light sensor and a 2 axis accelerometer. With this sensors the puppet is going to be able to send messages about its manipulation state (shaken, turned up, down, left, right) or the light sensor it has inside. Based on this information many reactivity scenarios can be defined.

Deploying the scenario Figure 5 shows the deployment process and the scenario used in the prototype (see Figure 6). To deploy any scenario using the context aware platform defined in this paper, the next steps have to be followed:

1. Define or reuse a context ontology. For the prototype, a simple ontology with the objects, characteristics and the context information they provide has been modelled.
2. Convert the ontology to the correspondent encoded version and create the header files to be used in the MicaZ Platform. This action can be performed automatically using the components of the software platform.
3. Once the network protocol is chosen, prepare the TinyOS program to send the semantically augmented packets. These packets are created by the WSN programmer using the helper files created in the previous step.
4. Define and implement the correspondent behaviours that are going to be applied in the scenario. In this prototype the next behaviours have been implemented:

  - When the puppet is manipulated (shaken) in the upright position, a comic application is started and displayed on a nearby screen.
  - Once the application is started, moving the puppet left and right turn the pages over.
  - If the light sensor level is raised, then it is considered that the puppet is open (because the sensor has been placed inside the puppet, so a warning is displayed on the screen.
5. Deploy all the components in the OSGi server and start the platform, connecting the WSN gateway to the server and activating the objects, the puppet in this scenario.

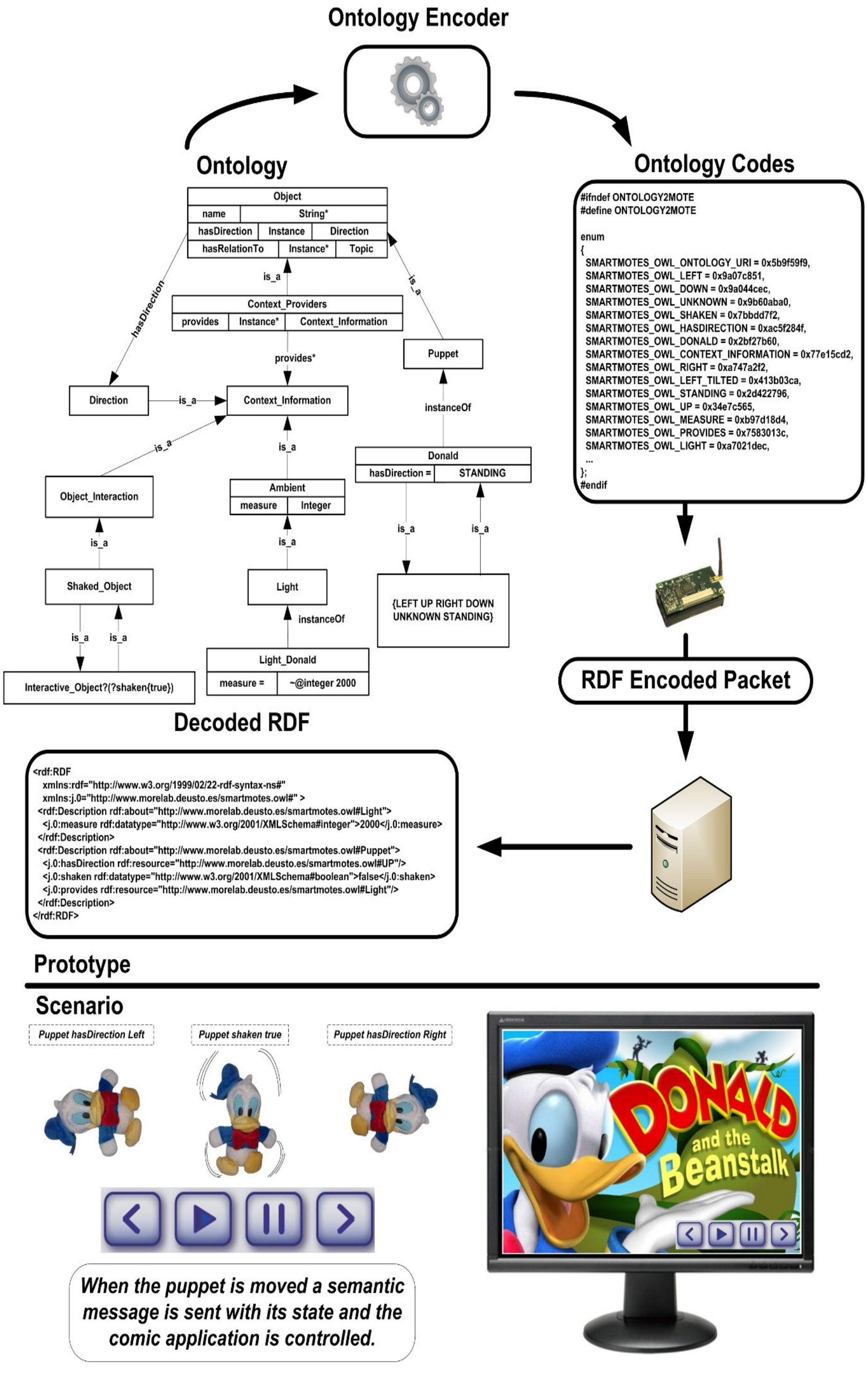
5.5 Prototype analysis

After implementing and deploying this simple prototype some conclusions can be drawn:

- The overall deployment process, regarding the ontology-to-sensor-packets procedure is rather simple. The conversion process is automatic and compared with any other ad-hoc mechanism, the developer here does not have to worry about how to encode sensor information, because this activity is now shared with Ontology Engineers.
- In this prototype a simple codification mechanism has been included but adding other kind of codifications is straightforward. Due to the plug-in structure of the WSN driver, even different codification mechanisms could coexist.
- Even though the prototype is centred on MicaZ platform, it can be applied to other platforms, e.g. SunSpots or iMote2.
- New nodes and new sensors can be easily incorporated into the system, as long as the information they provide is semantically defined in an ontology.
- Ontologies usually evolve along the life cycle of an application and this could affect already deployed sensors but this can be overcome with a careful version control of the ontologies.
- One of the benefits of using this codification strategy, which still remains to be explored, is that different sensors could understand each other with the only prerequisite that they import the codes of the ontologies they want to be able to understand.

One of the major drawbacks of this codification system is the overhead it introduces in the network packages. Considering the payload used in the prototype, it has 4 triples and two control fields (number of triples and ontology URI) which has 73 bytes altogether. An ad-hoc simple packet format could be considerably reduced, nevertheless losing the benefits of the semantic representation. For some applications this might be a problem, but others might benefit from it, like the one presented in the prototype, focused on smart objects.

Other critical aspect is the amount of time spent in the conversion from the packets to the correspondent RDF representation. In reality this conversion would depend on several factors. Depending on the hardware platform, it would take some time for the packet to arrive to the context manager. This packet would need to be converted to the RDF representation, but assuming that the context manager has already loaded the ontology, it is a straightforward conversion from one to the other.



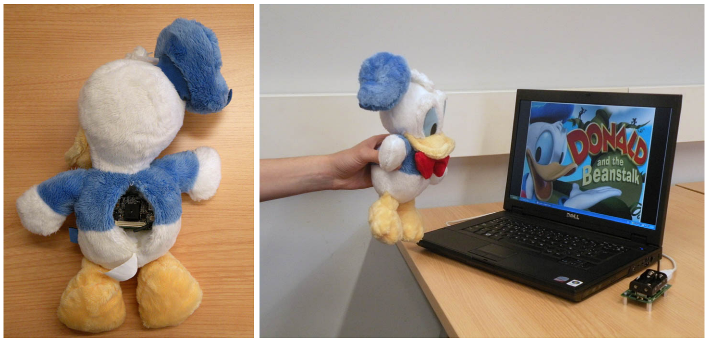
```
#ifndef ONTOLOGY2MOTE
#define ONTOLOGY2MOTE

enum
{
  SMARTMOTES_OWL_ONTOLOGY_URI = 0x5b9f59f9,
  SMARTMOTES_OWL_LEFT = 0x9a07c851,
  SMARTMOTES_OWL_DOWN = 0x9a044cec,
  SMARTMOTES_OWL_UNKNOWN = 0x9b60aba0,
  SMARTMOTES_OWL_SHAKEN = 0x7bbdd7f2,
  SMARTMOTES_OWL_HASDIRECTION = 0xac5f284f,
  SMARTMOTES_OWL_DONALD = 0x2bf27b60,
  SMARTMOTES_OWL_CONTEXT_INFORMATION = 0x77e15cd2,
  SMARTMOTES_OWL_RIGHT = 0xa747a2f2,
  SMARTMOTES_OWL_LEFT_TILTED = 0x413b03ca,
  SMARTMOTES_OWL_STANDING = 0x2d422796,
  SMARTMOTES_OWL_UP = 0x34e7c565,
  SMARTMOTES_OWL_MEASURE = 0xb97d18d4,
  SMARTMOTES_OWL_PROVIDES = 0x7583013c,
  SMARTMOTES_OWL_LIGHT = 0xa7021dec,
  ...
};
#endif
```

```
<rdf:RDF
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:j.0="http://www.morelab.deusto.es/smartmotes.owl#" >
  <rdf:Description rdf:about="http://www.morelab.deusto.es/smartmotes.owl#Light">
   <j.0:measure rdf:datatype="http://www.w3.org/2001/XMLSchema#integer">2000</j.0:measure>
  </rdf:Description>
  <rdf:Description rdf:about="http://www.morelab.deusto.es/smartmotes.owl#Puppet">
   <j.0:hasDirection rdf:resource="http://www.morelab.deusto.es/smartmotes.owl#UP"/>
   <j.0:shaken rdf:datatype="http://www.w3.org/2001/XMLSchema#boolean">false</j.0:shaken>
   <j.0:provides rdf:resource="http://www.morelab.deusto.es/smartmotes.owl#Light"/>
  </rdf:Description>
</rdf:RDF>
```

When the puppet is moved a semantic message is sent with its state and the comic application is controlled.

Fig. 5. Prototype architecture and scenario

Decoding the content of the objects (e.g. integers, strings, floats, etc.) could be considered negligible compared to the rest of the functions of the context aware system.



(a) iMote inside Donald puppet

(b) Prototype scenario view

Fig. 6. Prototype view

6 Conclusions and future work

The use of WSN nodes as semantic information sources is a breeding ground for several context aware applications that were not easily realized years ago. The research presented in this paper has been focused on providing an infrastructure for sharing context information optimized for WSN nodes, trying to solve one of the crucial problems that these systems are currently suffering: the lack of a standard representation adapted to WSN nodes characteristics. The fact that a new sensor node can directly make itself understandable based on a predefined ontology is a key step towards the full interoperability of different nodes with different sensors and even with different protocols. Besides, using semantic representations enables the context-aware systems to take advantage of all the work done in the reasoning part of these systems, making the development of smart environments an easier task.

A possible future direction of this kind of systems is the recent vision of Internet of Things [19]. This movement embraces the concept of many gadgets and real-life objects connected to the Internet sharing state and information. For this purpose, information that comes from the sensors directly in a semantic format is more likely to be published directly on the Internet. This integration of small sensors in real-life objects is also an emerging application field for activity recognition systems, where objects might not only react to certain inner-conditions but also be part of a context-aware system that is aware of the whole picture and can improve the adaptability and reactivity of the environment.

Regarding the infrastructure presented in this paper, the next step forward would be to take advantage of this semantic representation of the information to make some reduced reasoning in the sensor nodes. Having information from the ontology, every node could parse the messages of any other node and make some deductions, which is far more complicated if every node uses a different packet structure. Moreover, working with different WSN nodes platforms with different sensors would not be such difficult a problem anymore, assuming that they use the same network protocol (or even different if the context-aware system is properly configured). No matter what mechanism is used, the semantic definition of all the information coming from WSN nodes is a key step for their real integration in context aware systems in an interoperable and scalable way.

References

1. Jer Hayes, Edel OConor, John Cleary, Harry Kolar, Robert McCarthy, Richard Tynan, Gregory M.P. OHare, Alan F. Smeaton, Noel E. OConnor, and Dermot Diamond. Views from the coalface: Chemo-sensors, sensor networks and the semantic sensor web. In *SemSensWeb 2009 - International Workshop on the Semantic Sensor Web*, 2009.
2. Lionel M. Ni, Yanmin Zhu, Jian Ma, Qiong Luo, Yunhao Liu, S. C. Cheung, Qiang Yang, Minglu Li, and Min-you Wu. Semantic sensor net: an extensible framework. *Int. J. Ad Hoc Ubiquitous Comput.*, 4(3/4):157–167, 2009.
3. Yoshihiro Kawahara, Nao Kawanishi, Masahiro Ozawa, Hiroyuki Morikawa, and Tohru Asami. Designing a framework for scalable coordination of wireless sensor networks, context information and web services. *Distributed Computing Systems Workshops, International Conference on*, 0:44, 2007.
4. S. Helal, W. Mann, H. El-Zabadani, J. King, Y. Kaddoura, and E. Jansen. The gator tech smart house: a programmable pervasive space. *Computer*, 38(3):50–60, 2005.
5. Daniel Salber, Anind K. Dey, and Gregory D. Abowd. The context toolkit: aiding the development of context-enabled applications. In *CHI '99: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 434–441, New York, NY, USA, 1999. ACM.
6. Mike Botts, George Percivall, Carl Reed, and John Davidson. Ogc sensor web enablement: Overview and high level architecture. Technical report, OGC, December 2007.
7. Deshendran Moodley and Ingo Simonis. A new architecture for the sensor web: The swap framework. In *5th International Semantic Web Conference ISWC*, 2006.
8. Kin Wah Chow and Qing Li. Issdm: an in-network semantic sensor data model. In *SAC '07: Proceedings of the 2007 ACM symposium on Applied computing*, pages 959–960, New York, NY, USA, 2007. ACM.
9. Martin Strohbach, Hans werner Gellersen, Gerd Kortuem, and Christian Kray. Cooperative artefacts: Assessing real world situations with embedded technology. In *In Ubicomp*, pages 250–267. Springer, 2004.
10. Tsutomu Terada, Masahiko Tsukamoto, Keisuke Hayakawa, Tomoki Yoshihisa, Yasue Kishino, Atsushi Kashitani, and Shojiro Nishio. Ubiquitous chip: A rule-based i/o control device for ubiquitous computing. In *Pervasive Computing*, pages 238–253. 2004.
11. Karen Henricksen and Ricky Robinson. A survey of middleware for sensor networks: state-of-the-art and future directions. In *MidSens '06: Proceedings of the international workshop on Middleware for sensor networks*, pages 60–65, New York, NY, USA, 2006. ACM.
12. Micah Lewis, Delroy Cameron, Shaohua Xie, and I. Budak Arpinar. Es3n: A semantic approach to data management in sensor networks, 2006.
13. Grigoris Antoniou and Frank van Harmelen. Web ontology language: Owl. In *A Semantic Web Primer*, pages 110–150. MIT Press, 2004.

14. Frank Manola and Eric Miller, editors. *RDF Primer*. W3C Recommendation. World Wide Web Consortium, February 2004.
15. P. Levis, S. Madden, J. Polastre, R. Szewczyk, K. Whitehouse, A. Woo, D. Gay, J. Hill, M. Welsh, E. Brewer, and D. Culler. Tinyos: An operating system for sensor networks. pages 115–148. 2005.
16. Osgi Alliance. *OSGi Service Platform Core Specification*, 2007.
17. hp. Jena - a semantic web framework for java. available: http://jena.sourceforge.net/index.html, 2002.
18. Evren Sirin, Bijan Parsia, Bernardo C. Grau, Aditya Kalyanpur, and Yarden Katz. Pellet: A practical owl-dl reasoner. *Web Semantics: Science, Services and Agents on the World Wide Web*, 5(2):51–53, June 2007.
19. Christian Floerkemeier, Marc Langheinrich, Elgar Fleisch, Friedemann Mattern, and Sanjay E. Sarma, editors. *The Internet of Things, First International Conference, IOT 2008, Zurich, Switzerland, March 26-28, 2008. Proceedings*, volume 4952 of *Lecture Notes in Computer Science*. Springer, 2008.