An Adaptable Framework for Interoperating Between Wireless Sensor Networks and external Applications

Thanh-Dien Tran[1], David Nunes[1], Carlos Herrera [2] and Jorge Sá Silva[1]
[1] *Department of Informatics Engineering, University of Coimbra, Plo II, Pinhal de Marrocos, Coimbra 3030-290, Portugal*
[2] *Escuela Politcnica Nacional in Ecuador, Quito Canton, Ecuador*
[1] {*than, dsnunes, sasilva*}*@dei.uc.pt* [2]*carlos.herrera@epn.edu.ec,*

Keywords: Wireless Sensor Networks, Integration, Interoperability, Adaptability, STDL.

Abstract: Wireless Sensor Networks (WSNs) are considered the bridge to connect physical and digital worlds and thus an important element of the Future Internet. Consequently, integrating WSNs with external applications is an undeniable requirement. A gateway-based solution in which the sensed data and functions of WSNs are exposed as web services is a common approach. The problem of current integration solutions for WSNs is their adaptability, i.e., the ability to reuse gateways and proxies in a multitude of sensor networks with different types of applications and data frames. In this paper, we present our proposal for this problem by proposing a framework that uses a language for describing the traffic in sensor networks named Sensor Traffic Description Language (STDL). In order to reuse the framework on a new sensor network, it is only necessary to describe the network's frame structures using STDL.

1 INTRODUCTION

Wireless Sensor Networks (WSNs) have been receiving a great deal of attention from both research and industrial communities due to their potential in almost every field including military, industrial, healthcare and smart home environments. The major functions of WSNs are monitoring and controlling their target environments. They are as a bridge between physical to digital worlds. Therefore, in most of the cases, WSNs cannot operate in a complete isolation, and controlled by central servers, which host Supervisory Control and Data Acquisition (SCADA) systems or other applications.

Although it has been proven that it is possible to implement both IP protocol stack, i.e., uip (Dunkels, 2003), 6LowPAN (Hui and Culler, 2008), and web services (Priyantha et al., 2008), (Guinard, 2009), (Dawson-Haggerty et al., 2010), and (Shelby et al., 2013) on sensor nodes, this approach is not a solution for all types of sensor networks. Notably, allowing direct access to sensors from the Internet brings many challenges such as security, energy efficiency and routing. Altogether, it is necessary to have an infrastructure that easily, securely and efficiently supports the interoperability between sensor networks with external applications.

At the highly abstract level, the interoperability framework has to deal with two fundamental functions: (1) providing an interface for external applications to interact with the sensor networks; (2) communicating with the sensor networks. The first function can be solved by exposing the data and functionality of sensor networks using web services as the works in (Grosky et al., 2007), (Guinard, 2009), (Tran et al., 2012). On the other hand, the second one is a challenge because of the differences in communication protocols between WSNs and other networks, and diversity of possible sensor data formats. To deal with these issues, we can either implementing the same protocol on both environments or adding a gateway or a proxy between them.

The problem of current gateway-based integration solutions is their adaptability, i.e., the ability of the gateway or proxy to be reused, unchanged, for other networks with different data frames. It is difficult or even impossible to create a standard for the structure of the data inside the frame because there are so many possible formats. In this paper, we present our approach for this problem. The Sensor Traffic Description Language (STDL) was proposed for describing the structure of the sensor networks' data frames, allowing the framework to be adapted to a diversity of protocols and applications without reprogramming.

The remainder of the paper is organized as follows: section 2 summarizes the main related works.

Then the proposed model is presented in section 3. Section 4 describes the STDL, the illustrated examples, and STDL engine. The final section concludes the paper.

2 RELATED WORKS

Recent researches has proven that it is possible to implement both IP protocol stack (Dunkels, 2003) and 6LowPAN (Hui and Culler, 2008) and web services, e.g., Tiny web services (Priyantha et al., 2008), Web of Thing (WoT) (Guinard, 2009), sMAP(Dawson-Haggerty et al., 2010), CoAP (Shelby et al., 2013) on sensor nodes. However, the limitations of sensor nodes in terms of memory, computation, communication and battery power make it inefficient to deploy the full TCP/IP protocol stack and web services into all sensor nodes. In addition, allowing direct access to sensors from the Internet brings many challenges such as security and routing. To deal with these problems, some optimal mechanisms such as IP compression (e.g., 6LowPAN) (Hui and Culler, 2008), HTTP compression and UDP binding (Shelby et al., 2013), (Priyantha et al., 2008), EBHTTP (Tolle, 2010), and packed JSON (Dawson-Haggerty, 2010) need to be applied to the original standards. It is important to note that these mechanisms lead to incompatibility between the original standards and those for sensor networks. Consequently, a bridge, proxy, or gateway is still needed for interoperability between WSNs and external applications. As a matter of fact, the gateway or proxy approach for interoperating between WSNs and user applications will continue to exist in the foreseeable future.

Other works on integrating WSNs with the Internet environment includes GSN (Aberer et al., 2007), VIP Bridge (Shu et al., 2007), SensorWeb (Grosky et al., 2007). Most of these, excluding VIP Bridge, expose the functionalities of sensor nodes and networks as web services, which makes the interoperability over the Internet easier. However, the current gateway-based integration solutions are not adaptable to different types of data formats. This means that the new drivers or parsers need to be developed when applying the gateway to a new sensor network or when adding new sensors or applications to an existing one.

Open Geospatial Consortium's (OGC) (OGC, 2013) have been working on standards, i.e., Sensor Web Enablement (SWE) specification series, for making sensors discoverable and interoperable over the Internet. SWE consists of a series of open standard specifications for discovery of sensors and sensor systems, for exchanging and processing sensor observation, and for tasking of sensor and sensor system. The SWE standards can be divided into 2 groups: (1) models and schemes for encoding sensors and sensor observations; and (2) open web service interfaces. The former consists of Sensor Model Language (SensorML), and Observations and Measurements (O&M). The latter comprises Sensor Observation Service (SOS), Sensor Planning Service (SPS), and PUCK (OGC, 2013). The SOS exposes a web service interface for the client applications to retrieve either description or measurement from sensors. The response formats of SOS are in XML-based data encoding using SensorML or O&M. SensorML is used to return descriptions, while O&M specification is employed when the results are the measurements and observations. In addition, the user can task the sensors to perform appropriate actions through SPS web service interface. The SPS provides a list of observations that the users can assign to sensor networks. The PUCK protocol (OReilly, 2012) was integrated into SWE in 2011 to enable plug and play sensor networks.

The above solutions mainly provide a method for client applications to access sensor data through the gateway. An equally important aspect is how the gateway interacts with the sensors and sensor networks, how it can discover, analyze, extract data and issue command to WSNs. The traditional approach is to request the sensor nodes to organize the data according a specified format required by the provided drivers as in GSN (Aberer et al., 2007). Another method is to add a software driver or analyzer as in GSN (Aberer et al., 2007), SensorWeb (Grosky et al., 2007). In order to make the sensors as plug and play components of the gateway, the IEEE 1451 family standards (NIST., 2009) has been proposed. One of the core components of these standards is the definition of Transducer Electronic Data Sheets (TEDS), which are embedded into sensors or actuators to allow the interoperability between different manufacturers, and to make their data analyzable. Although it provides a standard way to exchange data, the software drivers and TEDS documents also need to be manually developed and installed on every sensors. Recently, SWE integrated PUCK protocol (OReilly, 2012) to store and automatically retrieve metadata and other information to/from the sensor nodes. The information stored in the nodes' memory is called PUCK memory and may include the IEEE 1451 TEDS, SensorML, or even the driver code. The gateway, which supports PUCK protocol, can automatically retrieve and utilize the information from the sensor node when it is installed (OReilly, 2012). If the driver code is available on the sensor node, it is downloaded into and executed on the host to translate the sensor raw data, using TEDS or SensorML, to the required format, e.g., SWE O&M object. The PUCK protocol brings another level of plug and play capability for sensor devices. However, it requires implementing PUCK documents on every sensor device. In addition, to make a device as plug and play, the driver code has to be physically stored in the PUCK memory before deployment. Consequently, it is not appropriate for WSNs that comprise many sensor nodes.

The work in this paper focuses on the solution for analyzing and extracting useful data from every sensor data frame between the framework and WSNs. It employs the sensor description method similar to TEDS (NIST., 2009) or PUCK documents (OReilly, 2012). However, it does not require adding the description documents to every sensor but only to the proxy. For the interaction with client applications, the proposed framework employs open web service standards similar to those of SWE (OGC, 2013). However, RESTful web services (Richardson and Ruby, 2007) and an encoding method similar to that those used in sMAP (Dawson-Haggerty et al., 2010) are employed. The following sections present the general integration framework, STDL, and its prototype.

3 THE GENERAL MODEL FOR INTEROPERABILITY

The main aspects that were considered when designing this model were interoperability, reusability, scalability and extensibility. The first aspect refers to the ability of the gateway to provide methods for external application to easily and transparently interact with sensor networks. The reusability, also considered as adaptability, refers to the ability to use the proxy or gateway for different WSNs without reprogramming or modifying. The third concern is about the ability of the model to handle the increasing number of sensor nodes and networks. The final aspect, extensibility, is about the ability to easily add new components to the model. Consequently, we come up with a general model as presented in Fig. 1.

In order to create a system that is able to respond to a large number of concurrently requests, we employed a multi-layered software architecture. As shown in Fig. 1 the model uses a proxy and gateway as a mediate layer for interoperability between sensor networks and the client applications. The proxy interacts directly with the WSNs, getting and analyzing data frames from the sensor networks, and then sending them to the gateway for storage. It also passes commands from the user applications to the sensor networks. The gateway allows front-end applications to interact with sensor networks, acting as a bridge between the sensor networks and the Internet. Both proxy and gateway may also comprise some other facility services such as authentication and authorization.

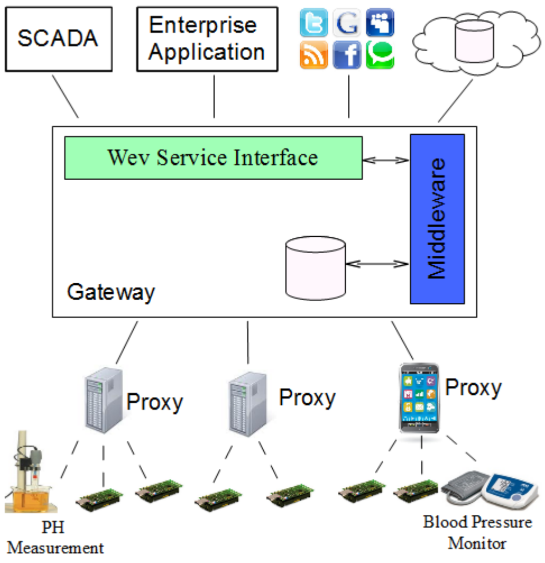


Figure 1: The General Model for Interoperability.

Both the proxy and gateway are designed to use an event-based principle. Developers can register the listeners that capture events of interest that happen in the network. For instance, to support localization, one can register to the *"data frame arrival"* event of the proxy and send location request to the localization engine on the gateway. This event-based model allows the system to be easily extensible.

It is worth noting that every sensor network is associated with at least one proxy and that a proxy can serve more than one sensor network. Every proxy is associated with at least one gateway to communicate the traffic of the sensor network to the data storage. There can exist multiple proxies and gateways in a single network. This type of architecture makes the proposed model scalable to be applied in large sensor networks. The detailed description of the middleware of the gateway was presented in (Tran et al., 2012). This paper focuses on the components of the proxy that make the integration framework adaptable with different types of sensor networks.

The proxy is responsible for obtaining the data frame from the sensor networks, analyzing and publishing them to the gateway. In addition, it also accepts commands from the user applications, and sends them to the sensor networks. The core components of the proxy are shown in Fig. 2. The proxy consists of

Traffic Listener and *Command Sender*. Because there are several methods for communication between sensor networks and the proxy, it is necessary to have different type listeners, one for each communication method. Currently, we implemented two types of traffic listeners on the proxy: one for serial port and the other for IP communication.

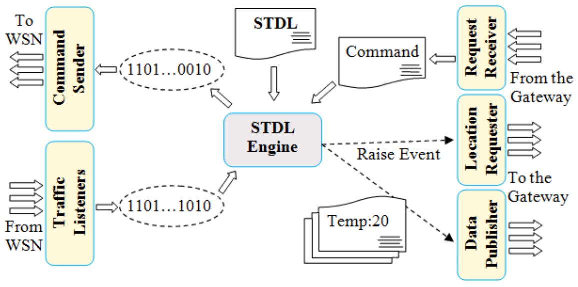


Figure 2: The Components of the Proxy.

The most important component of the proxy is the STDL engine that makes the proxy adaptable to different sensor networks. It is responsible for translating the raw data frame into meaningful data for other components. When receiving a raw frame from the listener, the STDL engine translates it into the one or more messages and raises the corresponding data events, which are handled by other components such as the *Data Publisher* or *Localization Requester*. The *Data Publisher*, when triggered by a data event, gets associated message and sends it to the gateway for storage. The *Location Requester* collects the necessary data from multiple events and sends them to gateway's localization engine, which estimates the position of a device.

The *Request Receiver* forwards user application commands coming from the gateway to the proxy engine, which translates them into a format that the sensor nodes can process. The engine then passes the command to the *Command Sender* component which sends it to the sensors. The STDL engine is independent from the type of sensor networks and can be reused without reprogramming. The details of STDL document are presented in the following section.

4 Sensor Traffic Description Language (STDL)

The STDL is an XM-based (Bray et al., 2006) language that adds adaptability to the infrastructure for sensor networks. It uses XML tags in a general enough way to describe structures for many different types of data frames. It is both formal and concise but simple and easy for developers to use. Like other languages, STDL provides a set of key words (vocabulary) and rules (grammar) that constrain how key words are combined together to describe the structure and permissible content of the data. As STDL is a XML-based, it is natural to use XML schema (XSD) (Gao, et al., 2012) for creating its constraint rules. Because XSD is a text-based language, reading it directly is tedious and it is difficult to convey its concepts. Therefore, it is useful to present the syntax of the language graphically.

Fig. 3 shows the elements for specifying data frames. Each data frame can be described by three elements: (1) attributes; (2) header; and (3) content. The attributes uniquely identify a specific frame and determine the table of the data storage into which the frame's data should be saved. The header element describes how to get the identification data from the instance frame, i.e., the raw frame received from a sensor network. The content element describes what data needs to be extracted from the instance frame. The following subsections describe these components.

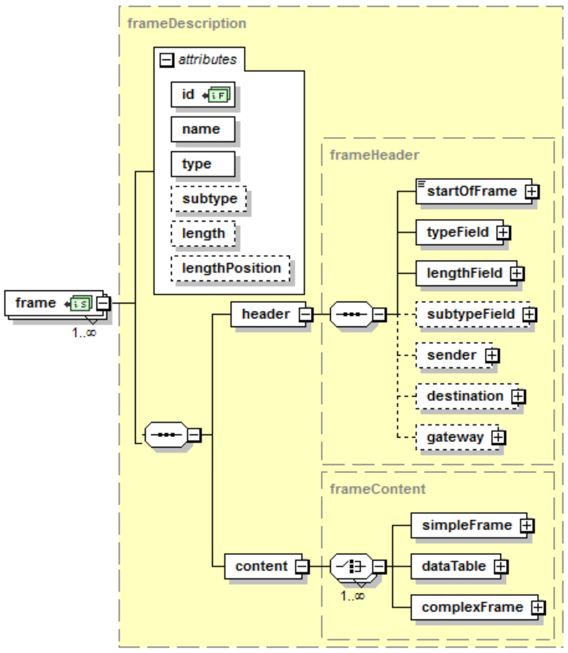


Figure 3: The General Content Model of STDL.

4.1 Frame Identification

The first issue that needs to be considered is the description of information that distinguishes the different frame types of a sensor network. The attributes of an STDL frame element are used to specify this information. Their contents can be divided into two functions: (1) identifying the raw frame; and (2) naming the frame (semantic). The former comprises the *type*, *subtype*, *length*, and *lengthPosition* attributes. The latter includes the *name* and *id* attributes.

The *type* attribute contains the value corresponding to the type field in the raw frame. It is the key for the STDL engine to associate the raw frame to its description in the list of frame structures for that network, each one describing the composition of a raw message. It is important to note that in some cases the *type* attribute cannot uniquely determine the frame. Hence, the optional *subtype* attribute comes into to play. Both attributes accept non-negative integer as their values and the combination of both allows every possible frame type to be identified. During the analyzing process, when the value(s) of a type and/or subtype (if applied) field(s) of a received raw frame match those of a stored frame structure, the STDL engine will use this frame structure as a reference to extract the rest of the data.

In some cases, the sensor frame does not have a length field but a fixed length, instead. In these situations the optional *length* attribute can determine whether a raw frame is valid. If this field exists and its value is greater than 0 then the matching raw frame has a fixed length. A *lengthPosition* attribute is used to specify the start position from which length is to be considered. The default value is 0 and means the length applies to the entire frame.

The data extracted from the raw frame eventually needs to be stored for further processing. This means that the proxy must know where to send the decoded data. Because a table is usually considered the outermost element of a database with which an outside entity may interact, we employ the table's name as a mechanism to link a frame to the data storage. The *name* attribute of the frame element serves this purpose and its value indicates the corresponding table name in the database into which the content of this type of frame should be stored.

The *id* attribute is used as the unique identifier of a frame, i.e., its primary key. The usage of this attribute will be discussed in the section that describes the frame content.

Because the *field* data type is used intensively in describing the header and the content of a frame, the next subsection is dedicated to this data type.

4.2 Field Description

The most important and useful aspect of STDL is to help the engine in gathering the needed information and meaning from the raw frames. Therefore, STDL must have the ability to specify the fields' positions as well as explain how to extract and decode each one. In addition, the semantics of each field are also very important because a value extracted from a raw frame is useless unless the users know their meaning. The *field* type is designed to fulfil this requirement. This type has six attributes: *name*, *unit*, *dataType*, *startPosition*, *numberOfBit*, *byteOrder*.

The last four attributes are used by the STDL engine to extract and to process raw data. The *startPosition* attribute specifies the position of the field in the raw frame. The *numberOfBit* attribute indicates its length in bits. Both these attributes are restricted to non-negative integers. The *dataType* attribute is used to indicate the type of the data. Currently, STDL supports the following data types: *string*, *uint8*, *int8*, *uint16*, *uint16*, *uint32*, *int32*, *ulong*, and *long*. The *byteOrder* attribute specifies the encoding method and its value is either *'little_endian'* or *'big_endian'* for little endian and big endian byte orders, respectively. By default, *'little_endian'* is assumed. The other two attributes are used to add semantics to the data field. The *name* attribute is used to name the described field, in the same manner as its respective table in the data storage. The *unit* attribute is a string that indicates the measurement unit of data content of the field.

4.3 Frame Header Specification

The objective of the frame header is to help the engine to matching a raw frame to a frame description. Frames are intercepted through the start of frame, i.e., a unique sequence of bits that marks the beginning of a new frame. As mentioned in previous sections, a raw frame can be uniquely specified by a type and an optional subtype. The STDL header section of a frame specification dictates how the above fields are retrieved from a raw frame. It is worth noting that the header specification is not used to describe the real header of the frame but its main purpose is to get enough information to uniquely identify the description of a particular raw frame.

As shown in Fig. 3, one of the required elements of a frame header is the *startOfFrame*. This element is useful to deal with the problem of heterogeneity and diversity of frames in WSNs, since different sensor networks use different start of frame sequences. The content of this element can be a sequence of bits or a list of hexadecimal numbers in the form of *0xnn* ... *0xnn*. It also has an *numberOfBit* attribute which specifies the number of bits of this field.

The next three elements, *typeField*, *subtypeField*, and *lengthField*, specify how to get the values of the type, subtype and length fields from the raw frame, respectively. These elements are described by the *field* type discussed in previous section. The first two elements are crucial because they help to associate a frame description to the raw frame.

Besides these essential elements, the description of a frame's header may also comprise three optional elements: *sender*, *destination* and *gateway*. These elements are included in the header part because they are present in most of the frames. The purpose of these elements, as implied by their names, is to describe the address of the original transmitter, receiver and gateway devices, respectively. These elements have the same structure as that of the previous three elements.

4.4 Frame Content Specification

The objective of frame content specification is to specify what information is needed and how to get it from instance frames. Working experience with sensor networks showed us that frames in WSNs can be organized into three main categories: *simple frame*, *data table frame* and *complex frame*.

As shown in Fig. 4, the *simpleFrame* type is used to describe the frames that merely consist of a list of data fields. Each field of a simple frame corresponds to a field of the raw frame and is described by the *field* type discussed in the previous section.

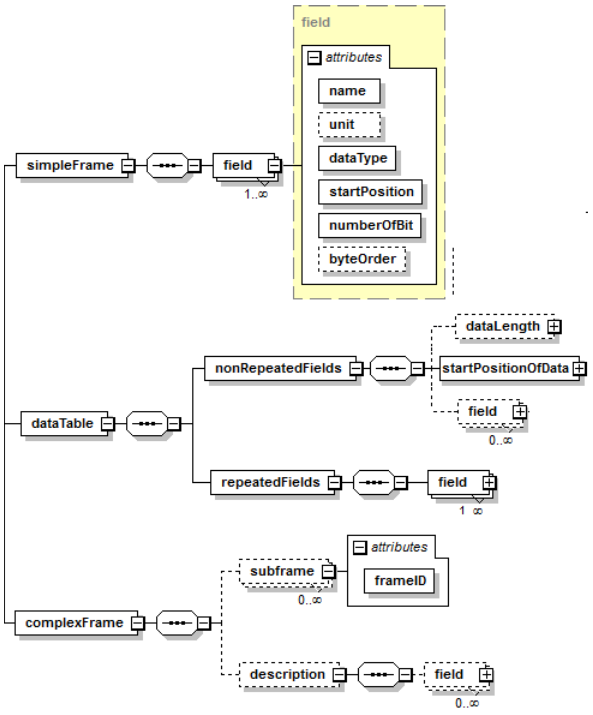


Figure 4: The General Content Model of STDL.

The second type of frame is called data table because its body contains a list of repeated measurement of different types of data. This type of frame can be used to describe aggregated data in WSNs. One possible scenario for its usage is when a sensor node buffers multiple samples before offloading data to the base station. Another possible case is when the intermediate nodes combine their collected data with that received from other nodes into a single frame and then send it to the base station. As shown in Fig. 4, the content of a *dataTable* frame is divided into two parts described by *nonRepeatedFields* and *repeatedFields* elements. The former comprises a list of elements that describe the information related to repeated data in the raw frame such as its start position (*startPositionOfData*) and its number of elements (*dataLength*). It also comprises a list of elements to be stored in the data storage such as sender, time, etc. The latter is used to describe the repeated fields in the data portion, with each field being described by the *field* data type.

Besides these two fundamental types of frames, when working with real world sensor networks it may happen to run into a scenario where a frame may contain one or more simple frames or data table frames. We deemed such frames as "complex" and described them using the *complexFrame* element. Because the complex frame contains other frames, it is necessary to describe composing frames as a regular frames and then refer to them in the complex frame. Consequently, specifying a complex frame is rather simple. As shown in Fig. 4, complex frames consist of a list of *subframe* elements which only have one attribute named *frameID*. This attribute refers to the id attribute of another frame described in the same document. Additionally, there is also some additional information that describes the general context of the frame. This information is specified using the *description* element.

4.5 Illustrated Example

To help us illustrate how to use STDL to describe the raw frames, let us assume a sensor network that creates a raw frame as the one showed in Fig. 5.

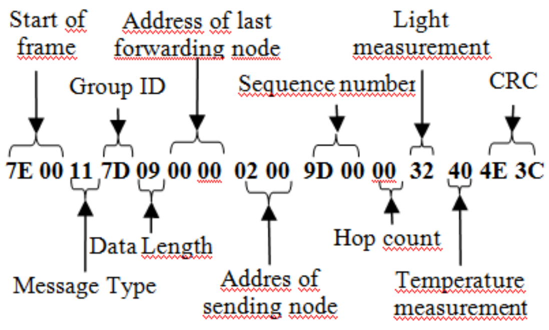
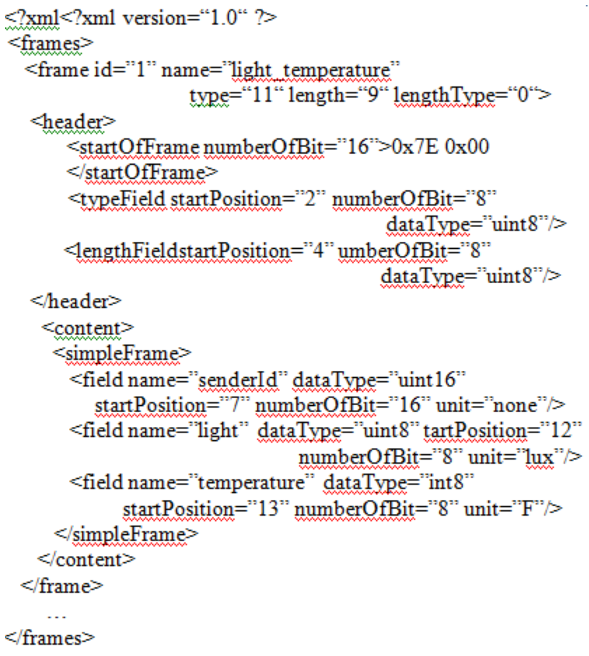


Figure 5: A example of a Simple Raw Frame.

The STDL description of this frame is shown in Fig. 6. In this example, it is assumed that the only information needed is the sending node identification and the light and temperature values. Consequently, the content element of the frame description only comprises three elements: *senderId*, *light* and *temperature*, respectively (see Fig. 6).



```
<?xml<?xml version="1.0" ?>
<frames>
  <frame id="1" name="light_temperature"
                        type="11" length="9" lengthType="0">
   <header>
       <startOfFrame numberOfBit="16">0x7E 0x00
       </startOfFrame>
       <typeField startPosition="2" numberOfBit="8"
                                           dataType="uint8"/>
      <lengthFieldstartPosition="4" umberOfBit="8"
                                          dataType="uint8"/>
   </header>
    <content>
     <simpleFrame>
       <field name="senderId" dataType="uint16"
          startPosition="7" numberOfBit="16" unit="none"/>
       <field name="light" dataType="uint8" tartPosition="12"
                                   numberOfBit="8" unit="lux"/>
       <field name="temperature" dataType="int8"
            startPosition="13" numberOfBit="8" unit="F"/>
     </simpleFrame>
    </content>
  </frame>
    ...
</frames>
```

Figure 6: Description of the Raw Frame in Fig. 5.

Another example is shown in Fig. 7, which is a complex frame containing a data table frame. This is a frame structure used in a sensor network developed by Eneida (Eneida, 2012). The example frame consists of three repeated fields: temperature, humidity and voltage. The STDL description for this type of frame is shown in Fig. 8. In this case, because the complex raw frame does not have a field type to identify it, that of the first inner frame is used instead.

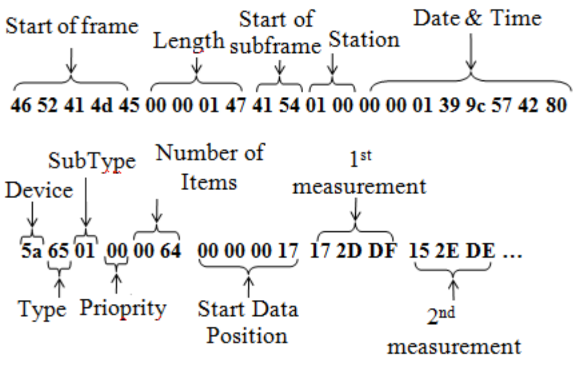
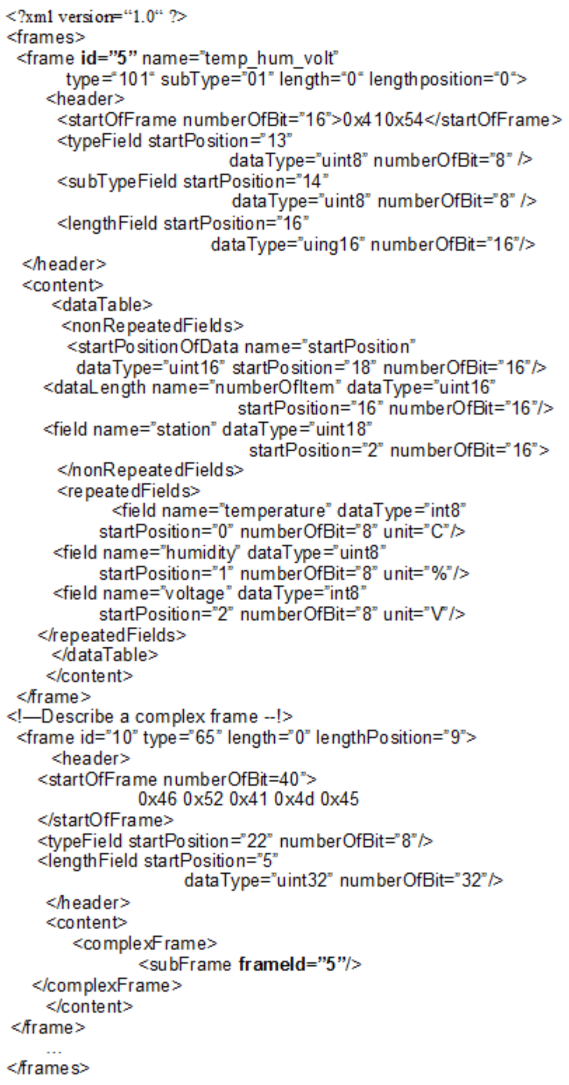


Figure 7: An Example of a Complex Frame Containing a Data Table.



```
<?xml version="1.0" ?>
<frames>
 <frame id="5" name="temp_hum_volt"
        type="101" subType="01" length="0" lengthposition="0">
      <header>
       <startOfFrame numberOfBit="16">0x410x54</startOfFrame>
       <typeField startPosition="13"
                              dataType="uint8" numberOfBit="8" />
       <subTypeField startPosition="14"
                              dataType="uint8" numberOfBit="8" />
       <lengthField startPosition="16"
                          dataType="uing16" numberOfBit="16"/>
  </header>
  <content>
      <dataTable>
       <nonRepeatedFields>
        <startPositionOfData name="startPosition"
          dataType="uint16" startPosition="18" numberOfBit="16"/>
     <dataLength name="numberOfItem" dataType="uint16"
                               startPosition="16" numberOfBit="16"/>
     <field name="station" dataType="uint18"
                                startPosition="2" numberOfBit="16">
       </nonRepeatedFields>
       <repeatedFields>
              <field name="temperature" dataType="int8"
            startPosition="0" numberOfBit="8" unit="C"/>
      <field name="humidity" dataType="uint8"
            startPosition="1" numberOfBit="8" unit="%"/>
      <field name="voltage" dataType="int8"
            startPosition="2" numberOfBit="8" unit="V"/>
    </repeatedFields>
      </dataTable>
     </content>
 </frame>
<!—Describe a complex frame --!>
 <frame id="10" type="65" length="0" lengthPosition="9">
      <header>
    <startOfFrame numberOfBit=40">
                0x46 0x52 0x41 0x4d 0x45
    </startOfFrame>
    <typeField startPosition="22" numberOfBit="8"/>
    <lengthField startPosition="5"
                      dataType="uint32" numberOfBit="32"/>
     </header>
     <content>
        <complexFrame>
              <subFrame frameId="5"/>
    </complexFrame>
     </content>
 </frame>
     ...
</frames>
```

Figure 8: STDL Description for the Content of the Raw Frame in Fig. 8.

4.6 STDL Engine

The STDL engine is the core part of the proxy. As shown in Fig. 2, the STDL document acts as the "brain" of the engine, guiding it through the processing of a received raw frame. The STDL document maps the frame's structure and allows the engine to extract the necessary data. In addition, it also accepts commands in form of messages from the *Request Receiver* component of the proxy and transforms them into the raw packets to send to the sensor network. In order to make the proxy more flexible and extensible an event based model is employed in the STDL engine where an event is raised after the engine processes a raw data frame. Another important point is that the engine employs the JSON encoding method (Crockford, 2006) for encoding data in event's message.

To illustrate how the JSON message looks like, let us examine the simple frame in Fig. 5. From the frame description in Fig. 6, the STDL engine knows how to extract the necessary fields from the raw frame. It also knows how to add the semantics to the extracted data to compose the message to include in the event. Consequently, by combining this information, the engine creates the following JSON object message:
{*"light_temperature":*[{*"senderId": 2, "light":50, "temperature":64*}]}.
It is worth noting that, in this case, an array of objects is used even though there is only one object. The reason for this is that we can use only one message format for all events raised by STDL engine.

5 CONCLUSIONS

Interoperating between WSNs and application environments is an undeniable demand. The architecture proposed in this paper allows for the use of integration infrastructure with diverse types of sensor networks. By using STDL, developers only need to describe the data frame structures when applying the framework to a new sensor network. The event-based approach also makes it very easy to add new components to the framework. In addition, exposing data and functionalities as web services greatly facilitates the use of WSNs in mash up applications that use them for monitoring, controlling, and visualizing real world data. All of the proposed methods were implemented and are running in our test platforms. As a future work, we will try expand the interface for sending commands to the sensor network, in order to make it easier to find and understand which commands the WSNs offers.

ACKNOWLEDGEMENTS

The work presented in this paper was partially financed by the iCIS project (grant CENTRO-07-ST24-FEDER-002003).

REFERENCES

Aberer, K., Hauswirth, M., and Salehi, A. (2007). Infrastructure for data processing in large-scale interconnected sensor networks. In *Mobile Data Management, 2007 International Conference on*, pages 198–205.

Bray, T., Paoli, J., Sperberg-McQueen, C. M., Maler, E., Yergeau, F., and Cowan, J. (2006). Extensible markup language (xml) 1.1. http://www.w3.org/TR/2006/REC-xml11-20060816/.

Crockford, D. (2006). The application/json media type for javascript object notation (json). Internet RFC 4627. http://www.ietf.org/rfc/rfc4627.txt.

Dawson-Haggerty, S., Jiang, X., Tolle, G., Ortiz, J., and Culler, D. (2010). smap: a simple measurement and actuation profile for physical information. In *Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems*, SenSys '10, pages 197–210, New York, NY, USA. ACM.

Dunkels, A. (2003). Full tcp/ip for 8-bit architectures. In *Proceedings of the 1st international conference on Mobile systems, applications and services*, MobiSys '03, pages 85–98, New York, NY, USA. ACM.

Eneida, I. (2012). Industrial wireless sensor networks. http://www.eneida.pt/produtos/.

Grosky, W., Kansal, A., Nath, S., Liu, J., and Zhao, F. (2007). Senseweb: An infrastructure for shared sensing. *MultiMedia, IEEE*, 14(4):8–13.

Guinard, D. (2009). Towards the web of things: Web mashups for embedded devices. In *In MEM 2009 in Proceedings of WWW 2009. ACM*.

Hui, J. and Culler, D. (2008). Extending ip to low-power, wireless personal area networks. *Internet Computing, IEEE*, 12(4):37–45.

NIST. (2009). Ieee 1451 smart transducer interface standard. The National Institute of Standards and Technology. http://www.nist.gov/el/isd/ieee/ieee1451.cfm.

OGC (2013). Sensor web enablement (swe). Open Geospatial Consortium. `http://www.opengeospatial.org/ogc/markets-technologies/swe`.

OReilly, T. (2012). Ogc puck protocol standard version 1.4. `https://portal.opengeospatial.org/files/?artifact_id=47604`.

Priyantha, B., Kansal, A., Goraczko, M., and Zhao, F. (2008). Tiny web services for sensor device interoperability. In *Proceedings of the 7th international conference on Information processing in sensor networks*, IPSN '08, pages 567–568, Washington, DC, USA. IEEE Computer Society.

Richardson, L. and Ruby, S. (2007). *RESTful Web Services - Web services for the real world*. O'Reilly Media.

Shelby, Z., Hartke, K., and Bormann, C. (2013). Constrained application protocol (coap) draft-ietf-core-coap-18. https://datatracker.ietf.org/doc/draft-ietf-core-coap/.

Shu, L., Cho, J., Lee, S., Hauswirth, M., and Zhang, Z. (2007). Vip bridge: Leading ubiquitous sensor networks to the next generation. *Journal of Internet Technology (JIT)*, 8(3):1–13.

Tolle, G. (2010). Embedded binary http (ebhttp). IETF draft-tolle-core-ebhttp-00. http://tools.ietf.org/html/draft-tolle-core-ebhttp-00.

Tran, T., Nunes, D., Gomes, A., and S Silva, J. (2012). An adaptive model for exposing wsn as a service platform. In *In Proceeding of INForum 2012 conference*.