TinyML: Meta-data for Wireless Networks

Nathan Ota
Berkeley Manufacturing Institute
University of California at Berkeley
Email: nota@kingkong.me.berkeley.edu

William T.C. Kramer
Lawrence Berkeley National Laboratory
One Cyclotron Road
Berkeley, CS 94720
Phone: 510-486-7577
Email: kramer@nersc.gov

Abstract

The growing number of deployed heterogeneous networks and applications is resulting in a sporadic isolated embedded sensor network environment. The TinyML project addresses the need for an embedded sensor network standardized "markup language" for intra-network, as well as inter-network, communication. The GML-based SensorML (SML) markup language provides a sensor-centric approach for satellite-based sensor coordination for global location services. However, SML relies on characteristics that differ from embedded wireless sensor networks, such as scale, hardware, energy, and infrastructure. TinyML focuses exclusively on embedded sensor network features, constraints, and capabilities. TinyML is capable of leveraging the flexibility of XML data structures with embedded sensor network reprogrammability. We present the concept of virtual sensors and actuators and implemented the necessary components to overcome the SML deficiencies. TinyML is a step forward in making sensor networks more accessible to the non-expert user and for archiving data retrieved from senor networks in a self-documenting manner.

Introduction

Embedded sensor networks are rapidly becoming a mainstay in many areas. Currently, these networks are custom assembled for specific applications. The applications entail complicated software, one of a kind combinations of hardware and unique deployment details. But take a moment to imagine the future.

A new homeowner wants to instrument his home to monitor and control temperature and manage the locks on his doors and windows. He determines the best value temperature control sensor network system is from RadioHut, while the best solution for this locking system is from HomeDump. Of course these two sensor systems, from different vendors, and are implemented with entirely different technology.

Now, Mr. Homeowner tries to integrate the two systems because he wants locking system to be able to react if the temperature system detects the presence of people in the room. He method is to have the temperature readings approximate the presence of a warm body. If a person is detected in a room at the proper times, the locks on the windows and doors open. In order to do this, he has to interface to both systems, then integrate the data from the two systems and generate commands to the locking system. Rather than deal with two different interfaces, data definitions and infrastructures, our Homeowner decides to use TinyML for this integration and control task since it has the capability to provide a generalize interface to any sensor network. Fortunately, both manufacturers support TinyML in addition to their proprietary interfaces. Because TinyML is based on XML, our Homeowner also has the ability to use many other tools manipulate, transform and manage his data, including translation to Web data and databases.

As sensor networks become more prolific, the need for methods to retrieve the data save the data in understandable formats, set aspects of the network and understand the parameters of the network will grow. Furthermore, today, each sensor network has its owns way to interface and retrieve data values. In the future, it will become critical for sensor networks to have simple ways to interface to standard web methods. This will enable many users to access data in a sensor net. Access should be archived or real time with the same methods as accessing archive data.

In the future, it will be imperative to be able to exchange and manage data from multiple sensor nets. Since it is likely most serious applications will have multiple sensor networks involved in their deployment, it is key to have the ability to easily interface between networks, and to facilitate the ability for data to flow between the components of the networks.

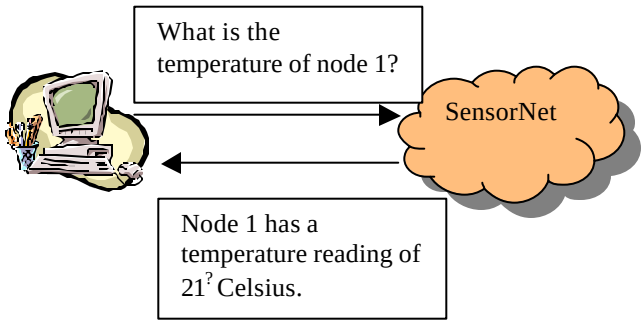

Figure 1 shows what a user (person or program) of a sensor network desires in the future. In essence they want to make a simple query of the sensor network in terms they can understand and receive back the information in the format they requested.

Markup Languages

Markup Languages have benefits of generalizing how information is organized and interpreted. This is most widely shown in the HyperText Markup Language (HTML) that is the lingua franca of the Web. Now there are new, improved markup languages defined that extend the concept of HTML and improve many features. These new ML's are the underlying basis for "Web Services" just as HTML was the basis for the browsers of the first generation Web.

A major goal of this work is to make use of this new technology to access and store data from embedded sensor networks, as well as controlling them as well.

XML

The Extensible Markup Language (XML)[1] is designed to organize working with and exchanging data in a structured manner on the Web. It is a standard set by the World Wide Web Consortium (W3C). "Extensible" means that the standard is very simple, but very flexible. It is designed to create tree relationships with data, originally for document descriptions. Because XML is extensible and flexible, it is being used in many ways, well beyond its original goal of describing written documents.

XML files have to be well formed and valid[2]. An XML file definition is called an XMLSchema (or the older style Document Type Definition - DTD). The XMLSchema describes how data is organized and what is allowable for valid data. It defines elements that can have attributes, default values, data types, and other features. Elements can be simple, or consist of other elements and attributes – therefore becoming complex. Applications are created by defining an XML Schema, and then using either common tools or custom code to transform the data. Using standard API's for XML, it is possible to use the Simple API for XML (SAX) for sequential one time parsing of XML or the Document Object Model (DOM) for random access parsing. XML is designed with the future in mind. Thus many of the evolving Web Services, such as 6h4 Simple Object Access Protocol (SOAP)[3], can use XML as their underlying descriptions.

GML

The Geographic Mark Language (GML)[4] is an XML based schema focused on the defining geographic and graphical information. This is a building block of other Markup Languages, including SensorML

GML is designed for modeling, exchanging , and storing geographic information by providing a the ability to describe geography including features, coordinate reference systems, geometry, topology, time, units of measure and generalized values.[5] GML support complex geometries, spatial and temporal reference systems, topology, units of measure, metadata, gridded data, and default styles for feature and coverage visualization. There are also packaging tools to create custom GML Schemas from the core XMLschemas.

Sensor Markup Languages

SensorML

SensorML[6] was created to handle data dissemination for satellite and other sensor networks. It defines a data schema that describes the geometric, motion characteristics and observations of a sensor[7] and is one of the first attempts to make a generalized sensor oriented Markup Language implementation using GML. SensorML is primarily designed for complex sensor platforms with a relatively few sensors but the sensors have a relatively large data flow. SensorML is also oriented on platforms that have complex positioning characteristics, such as satellites. Finally, SensorML is oriented to provide a structure for archived sensor data- including data in databases.

SensorML does not really have the concept of an actuator in the context of we use them in sensor networks. Rather it it has the concept of "Sensor Planning Services" who defines a way for a user to exrpress the desire to reserve and manipulate a sensor platform. An example of this is a scientist making a request to have a satellite positioned in a certain way

It is possible to consider using SensorML for sensor networks, but the resulting system may be overly complex

for current sensor networks. Further, SensorML is not yet released for general use until it is approved as a standard. Currently there are only reference implementations out for comment.

SensorML does not have the full concept of in-network processing. Rather, it has element definitions "Sensor Collection Services" that provide collected (observed) values. The Sensor Planning Service is a way for a acquisition request to be indicted. Once requested, there is a Web Notification Service that is used to alert requestors and other things from the Sensor Planning Service

Cougar

Cougar[8] is a method of handling queries for sensor networks. The major aspects of this are that queries are processed with a "query optimizer" that generates a query that is efficient on the particular network, reducing the use of network resources. Cougar introduces the idea of declarative queries as the manner to specify queries. The query optimizer determines how data flows between nodes and establishes aggregation filters that allow an optimization between computation and communication.

TinyDB

TinyDB[9] is an query processing architecture for TinyOS sensor networks. It is an SQL like interface that is distribute in natural, with a front end interface for users and the functionality to respond to the queries distributed in the network. It has the ability to efficiently use network resources and to perform limited operations on the data values – either with in network or at the interface.

Overview of TinyML

TinyML is a lightweight implementation following some of the SensorML ideas that are built on XML. Appendix A has a complete description of the elements defined for TinyML, and Appendix B gives the XMLSchema for TinyML. This section provides a basic overview of its structure. The basic platform components relate to physical devices, such as the Mica2 platform.

There are a few fundamental elements for TinyML. The first is the concept of a Platform. A platform consists of a basic infrastructure with some type of processor, an energy source and a radio or other communication device. In addition to the infrastructure, platforms have sensors and/or actuators. Basic sensors are such things a thermistors for temperature readings and microphones for sound detection.

A sensor field (TinyML's term for a collection of sensor nodes) is made up of a collection of platforms. The platforms may be uniform or different in their capabilities and attributes. A sensor network also has elements that could provide data to link the field to an external reference points. An example is a sensor field that has platforms with self organizing location data and a field reference description that has a link to GPS data at one or more points. It is possible to have multiple sensor fields grouped together into what is called a "super sensor field". This allows the ability for independent sensor networks to be joined and share data. In our first example, the homeowner is attempting to create a superSensorField from two sensor fields that have their own purpose and implementation.

All the elements mentioned above, as well as the virtual sensors and actuators mentioned below, have sub elements that provide detailed descriptions and additional information. This means descriptions of components in a field can be stored and queried. Thus if a user needs to know the details of a themistor's manufacturer or model number, it is possible to retrieve it.

There is a query flag, that when set, means the XML file is requesting a response from the component. A query indicator can also have an associated start time and duration. Many components also have a Set flag. For actuators, this means the XML file has values that set the actuator in some way, maybe positioning it. Set flags also are used to define the functions of a virtual sensor described below.

Table 1 provides a high level summary of the attributes and goals for SensorML, TinyDB, Cougar and TinyML. Essentially, each is different and provides unique features. TinyML is unique in it focus with embedded sensor networks while dealing with both basic sensors and actuators as well as virtual sensors and actuators. Figure 2 shows the potential interaction among the four systems. TinyML is capable of providing an interface between TinyDB, Cougar and SensorML. TinyML can also be used to archive sensor network data, just as SensorML does.

Virtual Devices

TinyML has the important concept of virtualizing physical components. Virtualization provides the ability to do several things. First, when associated with a platform, a virtual sensor or actuator can be created from physical devices. For example, if a platform has a thermistor that provides voltage readings as an output, a virtual sensor could be defined that would use the platform's processor to take thermistor output and, using calibration information, transform it to Celsius or Fahrenheit responses. Virtual devices can also be a collection of sensor outputs or actuator actions.

There are two major types of virtual sensors/actuators: those focused on platforms and those focused on sensor fields. Platform virtual sensors/actuators are associated only with basic sensors and/or actuators on a physical platform. For example, sensor field can have a virtual sensor or actuator associated with it. A field virtual sensor is an aggregate virtual sensor that can take readings from all the same sensors in the field and use a function such as Average, Maximum, or Minimum as possible virtual sensor output. Virtual sensors can also be associated with groups of sensors in the sensor field. This creates subgroups of platforms that use a function to develop a composite value. For instance, consider a sensor field throughout a building. A field virtual sensor could be the temperature sensors in a room providing a single temperature reading for the room. A more sophisticated virtual sensor function might use temperature differences to determine whether there is a person in the room.

Virtual sensors have a function associated with them. The function defines how a virtual sensor takes data from basic physical sensors and transforms it to become output of the virtual sensor. These functions have basic operators, such as add subtract, divide and multiply, as well as boolean operators for And, Or, Not, etc.. Further, it is possible for a virtual sensor to support intrinsic functions such as MAX, AVERAGE, etc.

The operators and functions can be performed by the physical platform or by the sensor network proxy that provides external network connectivity for the sensor network. The actual implementation details will vary for operators and functions and could be a predefined list that the sensor network supports, a set of library calls or, in more complex functions, over the net programming. The implementation is independent of TinyML since it is a feature of the network specific part of the TinyML Proxy. In this case, it is TinyDB

Consider a sensor network made up of platforms that have two sensors on them – represented in the diagrams below as a triangle and a circle. Figure 2a. shows a single platform with two sensors. Figure 2b shows a platform where a Virtual sensor is made by combining the output of two sensors. This would be a virtual sensor. The platform virtual sensor would have a function and a list of members – in this case the types of sensors that make up the virtual sensor.

Field sensors are also virtual sensors, but a more complex type. They are collections of sensors that exist on different platforms, but may not be all the platforms. Consider that a virtual sensor made up of the same sensor on a set of platforms – shown in Figure 2c. This would be a sensor field virtual sensor. Likewise, A sensor field virtual sensor could be created from a set of non-uniform sensors on collection of platforms as shown in figure 2d. Finally, as shown in Figure 2e, a field sensor may be a collection of virtual sensors on different platforms.

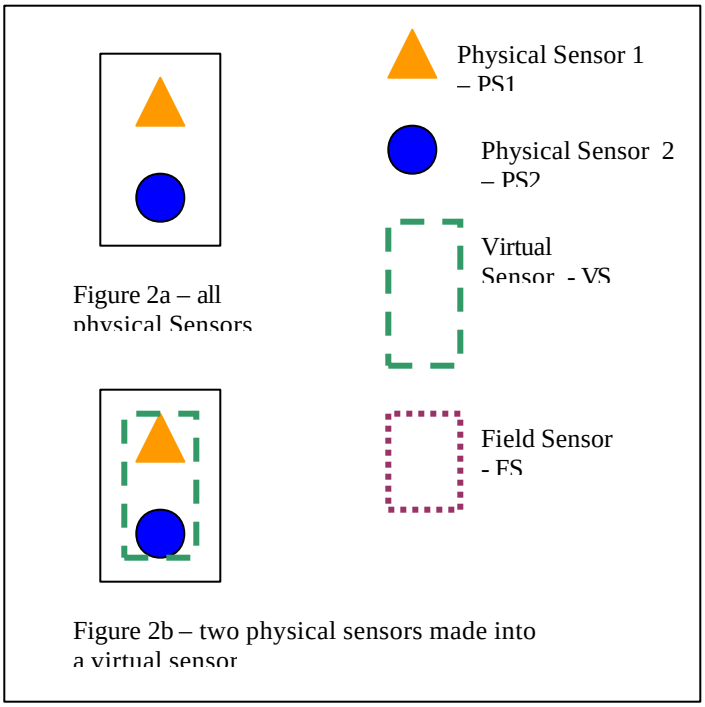

Figure 2a – all physical Sensors

Figure 2b – two physical sensors made into a virtual sensor

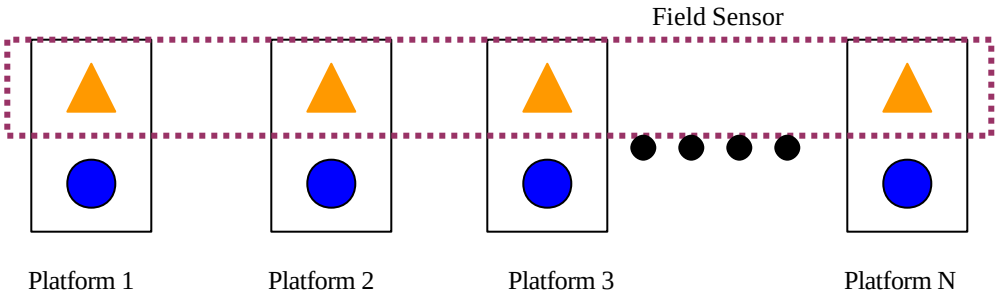

Figure 2c – a field virtual sensor made up of homogeneous physical sensors on different nodes. This requires a list of node ids to be associated with the field senssor

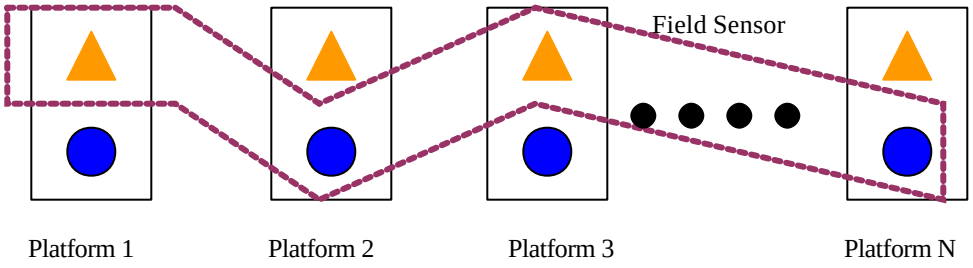

Figure 2d – a field virtual sensor made up of heterogeneous physical sensors on different nodes. This requires a list of node ids, and the sensor types on each node to be associated with the field sensor

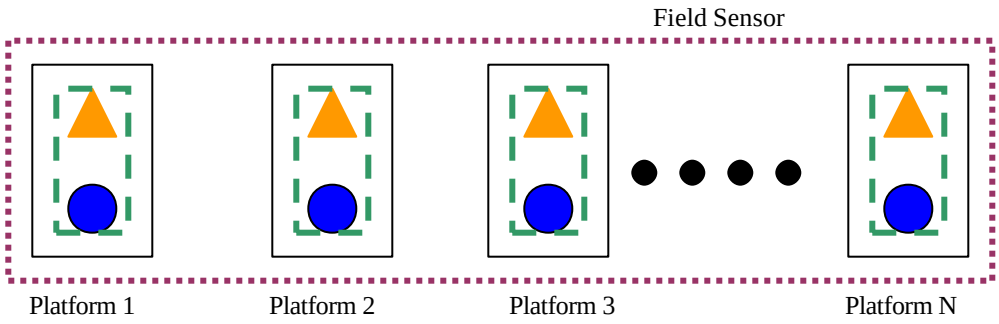

Figure 2e – a field virtual sensor made up of virtual sensors on different nodes. This requires a list of node ids and the type of sensors to be associated with the virtual sensor

	SensorML	TinyDB/ TinySchema	Cougar	TinyML
Objective	"Functional Description … not detail"	Application Specific Implementations	Application Specific Implementations	General Sensornet descriptions
Focus	Small number of platforms and sensors	Single networks with large numbers of platforms and sensors	Single networks with large numbers of platforms and sensors	Multiple networks with large numbers of platforms and sensors
Actuators	Sensor Planning Service	No	No	Yes
Aggregation	Sensor Collection Service	Yes	Within network	Virtual Sensors, Actuators and Fields
SensorNets Data Exchange	Yes	No	No	Yes
Routing	No	Some	Query specific routing	No
Query Structure	XML	Declarative – SQL like	Simple - "declarative query" details left for the future	XML
Location	External to sensors distributed in servers and www	Distributed between nodes and front-end	Distributed between nodes and front-end	External to a sensor network
Query Processing	Off nodes	Distributed between front end and www	Front end	Interfaced to the Sensor Network by a Proxy
Permanent	Yes – focus on archive data	No method for permanent storage – but does have a basic schema	No	Yes – data definitions are usable for archived data

Table 1- A comparison of different interfaces for sensor networks. This high level comparison explinas what is similar and different about TinyML.

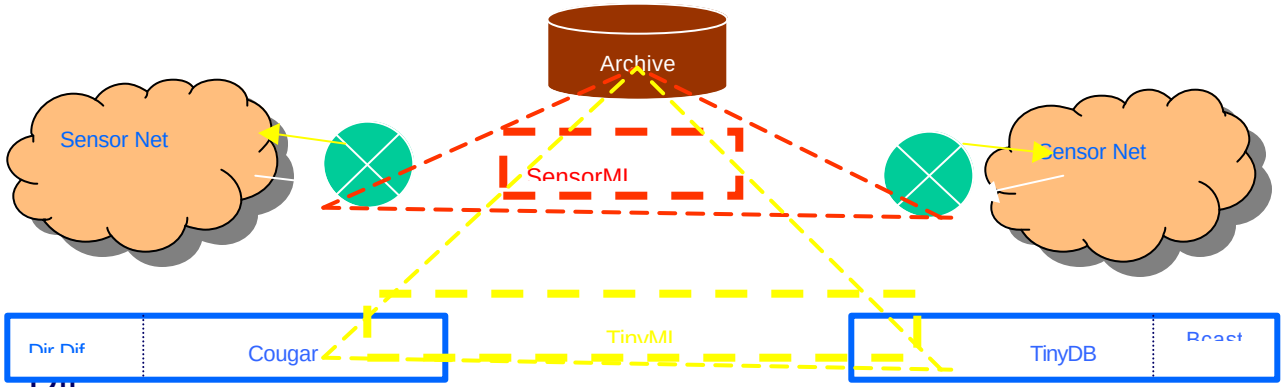

Figure 2 Positions TinyML and some other components discussed above in a typical sensor network. It shows two sensor networks, each with a different software implementation being linked by TinyML. It shows Cougar and TinyDB as components of the sensor networks, with the function of providing an interface to get data into and out of the network. TinyML can also describe data in a permanent archive just as SensorML can.

TinyML Implementation

Unfortunately, it is not possible to create sensor network interfaces that accommodate the simple model presented in Figure 1. Implementing the transparent solution requires placing XML into the sensor network itself, which means each platform must have the capability to parse, interpret, respond and formulate XML. In other words, creating a DOM on each node. This is not practical with current sensor/node limitations. Furthermore there would be significant increases in network traffic and power utilization in the implementation – which is counter to the dising gols of most embedded networks..

Placing an interface that translates XML to and from the application specific sensor network format is a workable solution and one we use to implement our proof of concept. Such an implementation uses sensor network resources efficiently and also is more straightforward to implement. Until, the Sensor/Platforms are sufficiently powerful ,sensor networks will have at least one gateway to the outside that is a more standard system (and more powerful). This is likely to continue for some time. So, the implementation of using the application or sensor net specific interface is sustainable. It is on this external interface that we implement the sensor field Proxy. In our case, we chose to use TinyDB as the interface.

Figure 3 shows a more complicated interaction where XML travels over the network to a sensor Net interface, is changed and then the corresponding sensor net specific format is used with the network. Thi replaces the simple t concept in Figure 1.

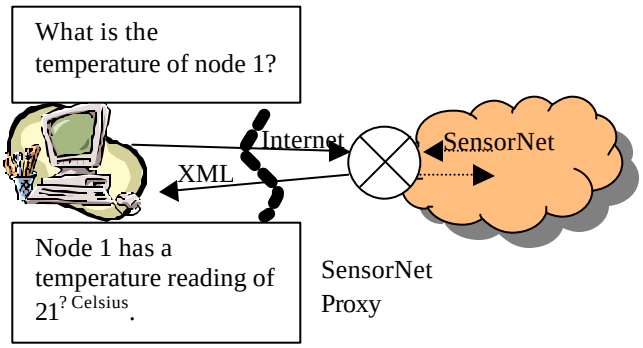

Figure 3 – The actual TinyML implementation of a query.

Network Setup and Assumptions

Sensor networks commonly consist of hundreds of spatially non-uniformly distributed nodes. All sensor nodes have the same core platform and energy reserves, but may be heterogeneous in the sensors contained on each node. Each sensor node has a library set of standard operators, mathematical and logical, that are publicly accessible. For example, a node can compute the sum given one or more arguments.

The routing and MAC layers are arbitrary however the network is assumed fully connected, as shown below in by the inter-node lines. Communication between nodes is symmetrical. Post-deployment, all nodes initially constitute a single sensor field, or a collection of collaborating nodes, and identify themselves with that default sensor field. As shown in Figure 4, all nodes are initially elements of sensor field SF1.

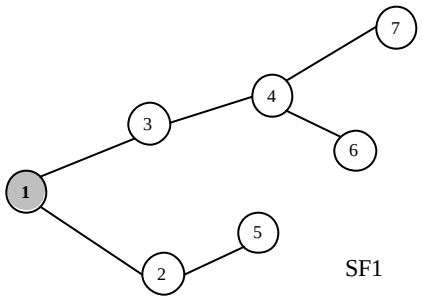

Figure 4 – A simple schematic of a sensor field.

The network has at least one *gateway* node, depicted as the grayed node 1, which provides inter-network connections, specifically via the Internet. The gateway node consists of a sensor node and a computationally-superior unlimited-energy hardware platform, commonly a PC. The gateway maintains a list of sensor node ID's contained in the sensor network. The sensor network and gateway communicate through a predetermined syntax and semantics. The arrangement between the TinyDB API and TinyOS TinyDBApp[10] exemplifies this type of design. The gateway node utilizes a sensor network interface, such as TinyDB, to format packets into to the predetermined syntax. The proxy could manages message transmission, particularly for messages that exceed the data payload limits of one packet. The proxy may also contain a *query generator* that translates queries to the proper syntax and semantic meaning understood by a sensor node. Again, TinyDB exemplifies this functionality. Importantly, the sensor network, by definition, includes the proxy and query generator software modules.

Applying TinyML

Core Functions

A common procedure exists for querying a TinyML-enabled sensor network, independent of the type and specific knowledge of the sensor network. A query is initiated by a remote user, either a human or external network, via a proxy. Despite the details of the remote user, any remotely-initiated query is first translated by the user's Proxy-to-DOM translator into a Document Object Model (DOM) following the TinyML schema. The Proxy-to-DOM translator converts the remote user's syntax and semantics to a generic DOM. The Proxy-to-DOM translator is proxy-specific and, if applicable, query-generator-specific. The proxy and Proxy-to-DOM translator must provide methods for handling and translating, respectively, all valid TinyML elements (simple and complex) and applicable attributes. Our prototype utilized the org.w3c.dom package in the J2SE1.4.2 API to parse a set of input strings ("light", "temp", etc…) from the prototype GUI to the correct TinyML-based DOM elements. In this case, the remote user proxy is the GUI since it translates queries.

Once the DOM is created, it is exported as an XML file with an XML-to-DOM parser and validated with the TinyML and GML schemas. The conversion to XML file format is a generic process and can be performed by any common utility. Our prototype utilized the standard java and javax packages in the J2SE1.4.2 API. If the validation fails, the query will fail back to the Proxy-to-DOM translator and requires modification. Else, the query is transferred to the destination gateway node.

Note that in all the components, the direction of translation is reversible. Hence, there is a Proxy-to-DOM interfaced with both the GUI Proxy and the sensor network Proxy. The GUI Proxy parses and prepares requests from an entity such as a person. On the outbound, the Proxy-to-DOM converts the GUI Proxy format into a DOM. At the sensor network, the process is inbound where the Proxy-to-DOM converts DOM to the particular format needed by the sensor network proxy. This reverses when the sensor network replies to the query. Then the sensor network Proxy-to-DOM is processing an outbound message and the GUI Proxy-to-DOM is processing an inbound message. Thus, there is a great deal of similarity in the code for either the GUI or Sensor network Proxy-to-DOM.

The means for transport of the TinyML formatted document are application specific. The prototype utilizes a generic TCP client/server connection implemented through the java.net package of the J2SE1.4.2 API. Our example utilizes a persistent server connection, though other approaches are viable. File transfer reliability and quality of service are left to the underlying protocols of the transport methods.

Once the transfer completes, the gateway node follows the reverse process as at the remote user. The figure below shows the local gateway node modules and sensor nodes.

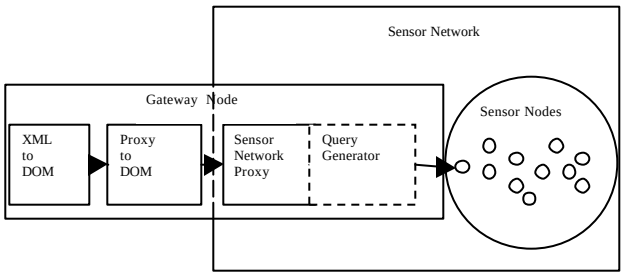

Figure 5 – The architecture of the sensor network side of TinyML. A sensor network is defined by a collection of sensor nodes and their proxy/query generator. The XML-to-DOM component is generic – that is it is only implemented once. The Proxy-to-DOM component needs to be specific to the sensor network proxy.

The XML file is parsed into a DOM structure via a generic XML-to-DOM parser. The DOM is then converted to the proxy/query-generator-specific structures and representations using a Proxy-to-DOM translator, referencing the TinyML and GML schemas. This Proxy-to-DOM translator may be different from the originating translator if the networks are from separate vendors or have different query-generators, such as Cougar and TinyDB.

At this point, the query is at the edge of the sensor network, by definition. In response to the query, the local proxy must return a valid response. At the time of this writing, valid responses are: null (rejection of query) or a returned value in the data type specified by the TinyML schema. The null response is utilized by a local proxy in the situation a query can not be completed any further, such as a query destination that does not exist in the sensor network. If the query can proceed, then the three cases below describe the following events.

Back to the Homeowner

Notice that there is a relatively simple way for our homeowner to set up his system. If the two manufactures provide a TinyML interface to their gateways, all is well land the homeowner creates a simple control program to send XML queries and actions. If not, then the homeowner only has to implement one TinyML component, a customer Proxy-to-DOM for the sensor network proxy. The XML-to-DOM component in Figure 6 is generic and will work with any interface. However, because there are standard tools available, this should be relative straight forward.

Scenarios

In order to better understand how a facility such as TinyML would work, it is instructive to think of some typical uses of such a system. Below are three use cases that demonstrate how a user might gather data from a sensor that support TinyML.

Scenario 1: Basic Sensor Request in a Standard Identity Network

The standard identity is the minimal sensor node state configuration that a TinyML-enabled network can possess. The node must have a default sensor field to which it is a member. Each node must also contain valid values for the required Platform sub-elements: PlatformCharacteristics, BasicSensor(s), and BasicActuator(s) and all the respective required sub-elements as defined by the TinyML schema. By definition, a Standard Initial Identity does not contain any FieldSensors, FieldActuators, VirtualSensors, or VirtualActuators.

In this simplest case, the remote user queries the thermistor basic sensor at node 4 in sensor field 1 by setting the basic sensor subelement QFlag (see TinyML schema) under the. Following the core procedures, the query is at the edge of the sensor network at the gateway node 1. The gateway injects the query into the network since the destination is a valid sensor node.

Upon reception, node 4 processes the query as per the predetermined syntax and semantics with the gateway. In this case, node 4 knows which ADC channel to poll for the "thermistor" since the standard identity mandates defining this information. Node 4 returns to the gateway the Qflag tag as true and the value as ADC reading; subsequently the message returns to the remote user through the established socket connection.

The situation may occur where node 4 does not contain this basic sensor. Since the gateway only manages the sensor node ID's and not the basic sensor(s) or virtual component(s) on each node, it is plausible that the gateway forward a query that can not be fulfilled. It is the responsibility of each TinyML-enabled sensor node to provide a null response in the event of such a query. Specifically, the sensor node returns the Qflag value as true, since the query was accepted, but the value is set to null.

Scenario 2: Virtual Sensor Query in a Virtualized Identity Network

A virtualized identity contains virtual components in addition to the standard identity. As an example, the remote user

queries an "occupancy" virtual sensor at node 4 in sensor field 1 by setting the "get" element. Assume node 4 has a pyroelectric basic sensor (a pyroelectric sensor is a specialized infrared sensor) in addition to the thermistor. Assume the "occupancy" virtual sensor function calculates the value of this virtual sensor by performing a set of logical operations on the two basic sensor values. Following the core procedures, the query is at the edge of the sensor network at the gateway node 1. The gateway injects the query into the network since the destination is a valid sensor node.

Upon reception, node 4 processes the query as per the predetermined syntax and semantics with the gateway. Again, node 4 knows which ADC channels to sample. Node 4 also knows the logical mathematical operations to perform as defined by the virtual sensor function. The "occupancy" value is returned to the gateway, and subsequently returned to the remote user through the established socket connection. Like the basic sensor query, the Qflag is also returned true.

Like the previous case, the situation may occur where node 4 does not contain this virtual sensor. Again, it is the responsibility of each TinyML-enabled sensor node to provide a response in the event of such a query, the null response. Unlike the basic sensor query, the gateway has a second option to obtain the virtual sensor sample as described in the next scenario.

Scenario 3: Virtual Sensor Query in a Standard Identity Network

The problem was presented in the previous scenario that a node is queried for a non-existent virtual sensor. Assume that the "occupancy" virtual sensor is not defined at node 4. The first valid response by the node is a null return value with a true Qflag. Null responses allow a node to communicate to the gateway that although the query was received, the virtual sensor does not exist.

When the null value and true Qflag tuple is received at the gateway, the gateway can elect to initiate a virtualization with the sensor node if the virtual sensor function is defined at the gateway. The gateway node first queries the sensor node for permission to initiate a virtualization. To do so, the gateway sends virtual sensor query again, but with the Sflag subelement set instead of the Qflag and the number of operations to send in the function subelement. If the sensor node does not permit virtualization, the Sflag is returned false. Otherwise, the sensor node returns a true Sflag signifying it is willing and ready to accept virtualization.
To virtualize the sensor node, the gateway passes the virtual sensor function (see TinyML schema) operations and operation number (from the total number of operations) to the sensor node, who then defines a new function appropriately. For example, the "occupancy" virtual sensor function initially needs to compute the temperature in degrees Fahrenheit from the thermistor voltage value. Assuming a linear relationship between the voltage and temperature, the gateway passes the instruction to the sensor node as a new variable name (temperature), a tuple of arguments (thermistor and a coefficient) and the multiplication operator. The sensor node defines and assigns, utilizing its library of mathematical operators, the temperature variable to the product of the thermistor voltage reading and the coeffiecient. This process continues until the total number of operations are received and the new "occupancy" virtual sensor function is complete. After receiving the last instruction, the sensor node returns a true Sflag signifying the completed virtual component. Finally, the gateway can again query node 4 for the "occupancy" virtual sensor.

A second situation may arise where "occupancy" virtual sensor is not defined at the gateway. Here, the initial null response passes through the gateway to the remote user. Identical to the gateway-initiated virtualization, the remote user can virtualize the sensor node using the same process of sending instructions.

This virutalization process can be repeated an arbitrary number of times for virtualized nodes that lack a specific virtual sensor. In this manner, virtual components enable a sensor network to evolve in a piecewise fashion, bounded only by the basic sensors and library of operators on each node.

Proof of Concept Implementation

In order to prove the concepts discussed in the paper, a simple version of a TinyML based query/response system was created. The system used the XML Schema presented in Appendix B as the description of TinyML. The sensor network consisted of Mica2 platforms running Tiny OS version 1.1. Each platform had a basic thermistor and photosensor on it. The sensor network used TinyDB, from that TinyOS release as the sensor network proxy and query generator. The TinyML components were created using Java, in the details are discussed above.

This prototype had the GUI query generator running on one PC, and transferred XML across the Internet to a PC running TinyDB, the TinyML XML-to-DOM, and Proxy-to-DOM components. Simple queries were demonstrated. The code for the components, along with the TinyML schema are available on the project web site.

Summary

The goal of this project is to investigate the issues involved with the creating a general interface, TinyML, that facilitates two way interaction between an internet users and a set of sensors, as well as interactions between sensor networks themselves. The project surveyed likely existing systems that partially met these goals, and determined none were sufficient for sensor networks.

TinyML was conceived and created in order to explore the interface between sensor nets and the internet services in a way the appropriately partitions and assigns data synthesis and control to the network and the standard servers in the most effective manner.

GML and GML-based SensorML markup languages schemas currently have a focus for satellite-based sensor coordination for global location services. These interfaces rely on certain characteristics/needs that differ in the following ways from mote-centric wireless sensor networks.

The project addressed the need for such an embedded sensor network standardized “markup language” for intra-network, as well as inter-network, communication is apparent as more systems are deployed.

The project created the TinyML schema including the concept of virtual sensors and actuators and implemented the necessary components to enable a working prototype. Future work is to full evaluate the TinyML schema for robustness by implementing one or more really application interfaces using TinyML. Additionally, performance study and optimization are needed, as is the inclusion of more advanced XML and Web Services functions. Nonetheless, TinyML is a significant step forward in making sensor networks more accessible to the non-expert user and for archiving the data retrieved from senor networks in a self documenting manner.

Appendix A Table of Elements

Element	Description
superSensorField	A collection of sensor fields.
SensorField	A basic sensor network – consisting of platforms, sensors and actuators and reference data
Platform	The basic physical platform in a sensor field, which contains processing and communication elements. It may also contain on or more sensors and actuators. An example is the MICA 'mote"
BasicSensor	A basic physical sensor that is attached to a platform. It is identified by type.
BasicActuator	A basic physical actuator that is attached to a platform. It is identified by type.
FieldReference	Appearing once in a Sensor Field, this is location information that may be used to connect the sensor information to the real world reference. An example would be a node with GPS location ability.
FieldDescription	A description of a sensor field consisting of routing description, a field description and other information
FieldSensor	This is a virtual sensor associated with a set of sensors on different platforms in a sensor field. These could be basic sensors or virtual sensors. An example is getting the average temperature value for an entire sensor field.
FieldActuator	This is a virtual actuator associated with a set of actuators on different platforms in a sensor field. These could be basic actuators or virtual actuators.
VirtualSensor	A virtual physical sensor that is associated with one or more basic sensors on a platform. The virtual sensor performs a transformation on physical sensor data according to a function. An example is a virtual sensor that transforms a thermistor value (with calibration) into a reading in Celsius.
VirtualActuator	A virtual physical actuator that is associated with one or more basic actuators on a platform. The virtual actuator performs a transformation on physical actuator according to a function. An example is a virtual actuator that ????
Query	A complex flag that is associated with a number of elements. If set to TRUE, then the fields associated with the flag are returned. This allows any element in the sensor field to be queried. Query has an optional start and duration time.
SetFlag	A complex Boolean flag that is associated with a number of elements. If set to TRUE, and the element is settable, then the fields associated with the flag are set to the provided value(s). Examples of use are setting the function associated with a virtual sensor or setting the values of an actuator. SetFlag also has a start time and duration.
Type	A general element the definition for the type of a component or element. This could be a manufacturers part number or a specification that indicates the type of a photo sensor.
Vendor	A general element describing the vendor of the component. This element consists of Vendor Description and a Universal Resource Indicator (URI).
Range	A description of the range or a sensor or actuator. It is composed of a distance measure and the units used.
Accuracy	Describes the accuracy of a component.
Location	The location of the parent element. It can be either absolute or relative coordinates.
Orientation	The orientation description for a platform, sensor or actuator. It is in one of several formats
Hardware	An element that describes the hardware of a platform. It could consist of descriptions for processor, radio and energy source.
Software	The software associated with a platform. This includes the OS description and the version.
Function	The function specification for a virtual sensor or actuator, including field sensors and actuators. This function can be either preset or specified by a user if allowed.
SpanofMembers	A list that is used to describe a virtual sensor actuators and other composite components.
Coordinates	An element that describes coordinates for several other elements. It consists of an X and Y values, and possibly a Z and timestamp.
Datavalues	The element for any data values from a sensor or actuator. Data values can be integer, floating point or string.
CurrentSettings	The current setting of an actuator.
Timestamp	A timestamp for data value readings and settings and for query responses

EnergySource	The energy source of a platform. Components are the Power level and Source Description
ID	The ID field for a component. This is used by sensor fields, and platforms primarily
ModelDecription	The platform model as part of the description of the platform.
Absolute	One of three types of orientation specifications. This is in absolute X, Y, Z values
Quatenion	Quaternions extend the concept of rotation in three dimensions to rotation in four dimensions. This avoids the problem of "gimbal-lock" and allows for the implementation of smooth and continuous rotation. In effect, they may be considered to add a additional rotation angle to spherical coordinates ie. Longitude, Latitude and Rotation angles that are calculated from the combination of the three coordinates of the rotation axis and the rotation angle.
Euler	An orientation specification in Eulian angles – Pitch, Roll and Yaw
Xvalue	A value representing an X value
Yvalue	A value representing an Y value
Zvalue	A value representing an Z value
Svalue	A value representing an S value for a quaterion description of the orientation
Pitch	The angle relative to the X axis (e.g. on an aircraft the axis defined by the wings)
Roll	The angle relative to the Y axis (e.g. on an aircraft the axis defined from tail to nose)
Yaw	The angle relative to the Z axis (e.g. on an aircraft the axis defined by pointing up)
Value-int	An integer value – a sub element of Datavalues
Value-flt	A floating point value – a sub element of Datavalues
Value-string	A string value – a sub element of Datavalues
Processor	An element describing a processor on a platform – consisting of a description and speed specification
ProcessorSpeed	The clock rate of a processor
ProcessorDescription	The description of a processor
Radio	An element to describe a radio on a platform
Routing	A description of the routing of a sensor field
Reference	A reference description of a sensor field
OtherInformation	A description of the other information related to a sensor field
Units	A string describing the units of a range or value.
PowerLevel	The power level of an energy source on a platform
SourceDescription	A description of an energy source. Examples include battery or photo
OSDescription	The operating systems description.
Version	A string description of the version (for the software, hardware, etc.
RelativeLocation	The relative location of a platform or other component
AbsoluteLocation	An absolute location of a platform or other component
Distance	The distance of a range specification
Qflag	The Boolean flag associated with a query Request
StartTime	The time a query is supposed to start (for a request) or time it did start (for a response).
Duration	The duration of a query request
Date	The date of a time stamp
Time	The time of a time stamp
VendorDescription	The string description of a vendor
URI	A Universal Resource Indicator – the XML version of a URL.

Appendix B TinyML XMLschema Listing

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XMLSPY v2004 rel. 3 U (http://www.xmlspy.com) by Bill Kramer (UCB) -->
<?xmlspysps C:\Program Files\Altova\XMLSPY2004\Examples\ExampleSite\examplesite.sps?>
<xs:schema xmlns:gml="http://www.opengis.net/gml"
xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
attributeFormDefault="unqualified">
   <!-- ======================================================================
           includes and imports
      ====================================================================== -->
   <xs:import namespace="http://www.w3.org/2001/XMLSchema"
schemaLocation="http://www.w3.org/2001/XMLSchema"/>
   <xs:import namespace="http://www.isotc211.org/iso19115/"
schemaLocation="D:\Data\School\CS 294-Embedded
Networks\Project\tinyML_files\iso19115\0.1.21\iso19115.xsd"/>
   <!-- ======================================================================
           The set of Sensor Fields
      ====================================================================== -->
   <xs:element name="superSensorField">
      <xs:annotation>
            <xs:documentation>A collection of sensor fields</xs:documentation>
      </xs:annotation>
      <xs:complexType>
            <xs:sequence>
                  <xs:element ref="SensorField" maxOccurs="unbounded"/>
            </xs:sequence>
      </xs:complexType>
   </xs:element>
   <!-- ======================================================================
          Funduamental set of platforms and fields
      ====================================================================== -->
   <xs:element name="SensorField">
      <xs:annotation>
            <xs:documentation>SensorField is the root element</xs:documentation>
      </xs:annotation>
      <xs:complexType>
            <xs:sequence>
                  <xs:element ref="Platform" maxOccurs="unbounded"/>
                  <xs:element name="FieldDescription" minOccurs="0">
                        <xs:complexType>
                              <xs:sequence>
                                    <xs:element name="Routing" type="xs:string"
minOccurs="0"/>
                                    <xs:element name="Reference" minOccurs="0"/>
                                    <xs:element name="OtherInformation"
type="xs:string" minOccurs="0"/>
                              </xs:sequence>
                        </xs:complexType>
                  </xs:element>
                  <xs:element name="ID"/>
                  <xs:element ref="Query" minOccurs="0"/>
                  <xs:element ref="FieldActuator" minOccurs="0"/>
                  <xs:element ref="FieldSensor" minOccurs="0"/>
            </xs:sequence>
      </xs:complexType>
```

```
   </xs:element>
   <!-- ========================================================================
         Elements that make up a sensor field
         ======================================================================== -
->
   <xs:element name="Platform" abstract="false">
      <xs:annotation>
            <xs:documentation>Platform object </xs:documentation>
      </xs:annotation>
      <xs:complexType>
            <xs:sequence>
                  <xs:element name="Sensor" minOccurs="0" maxOccurs="unbounded">
                        <xs:complexType>
                              <xs:sequence>
                                    <xs:element ref="BasicSensor" minOccurs="0"
maxOccurs="unbounded"/>
                                    <xs:element ref="VirtualSensor" minOccurs="0"/>
                              </xs:sequence>
                        </xs:complexType>
                  </xs:element>
                  <xs:element name="Actuator" minOccurs="0">
                        <xs:complexType>
                              <xs:choice>
                                    <xs:element ref="BasicActuator"/>
                                    <xs:element ref="VirtualActuator"/>
                              </xs:choice>
                        </xs:complexType>
                  </xs:element>
                  <xs:element ref="Location" minOccurs="0"/>
                  <xs:element ref="Orientation" minOccurs="0"/>
                  <xs:element name="PlatformCharacteristics" minOccurs="0">
                        <xs:complexType>
                              <xs:sequence>
                                    <xs:element ref="Vendor" minOccurs="0"/>
                                    <xs:element ref="Software" minOccurs="0"/>
                                    <xs:element name="ID" type="xs:string"
minOccurs="0"/>
                                    <xs:element ref="Hardware" minOccurs="0"/>
                                    <xs:element name="ModelDescription"
type="xs:string" minOccurs="0">
                                          <xs:annotation>
                                                <xs:documentation>A description of
model/attributes/version</xs:documentation>
                                          </xs:annotation>
                                    </xs:element>
                                    <xs:element ref="Query" minOccurs="0"/>
                              </xs:sequence>
                        </xs:complexType>
                  </xs:element>
                  <xs:element ref="Query" minOccurs="0"/>
            </xs:sequence>
      </xs:complexType>
   </xs:element>
   <xs:element name="FieldReference" abstract="false">
      <xs:annotation>
            <xs:documentation>For providing geolocation models</xs:documentation>
      </xs:annotation>
```

```
      <xs:complexType>
            <xs:sequence>
                  <xs:element ref="Location"/>
                  <xs:element name="LinktoRealWorld" type="xs:string" minOccurs="0"/>
                  <xs:element name="HowDetermined" type="xs:string" minOccurs="0"/>
            </xs:sequence>
      </xs:complexType>
   </xs:element>
   <xs:element name="FieldSensor">
      <xs:annotation>
            <xs:documentation>A virtual sensor that incorporatessensors across the
entire field</xs:documentation>
      </xs:annotation>
      <xs:complexType>
            <xs:sequence>
                  <xs:element name="Get" type="xs:boolean" default="false"
minOccurs="0"/>
                  <xs:element ref="Query" minOccurs="0"/>
                  <xs:element ref="Datavalues" minOccurs="0"/>
                  <xs:element ref="Type" minOccurs="0"/>
                  <xs:element ref="Function"/>
                  <xs:element ref="Range" minOccurs="0"/>
                  <xs:element ref="Accuracy" minOccurs="0"/>
                  <xs:element ref="SpanofMembers" minOccurs="0"/>
            </xs:sequence>
      </xs:complexType>
   </xs:element>
   <xs:element name="FieldActuator">
      <xs:annotation>
            <xs:documentation>A virtual actuator that incorporatessensors across the
entire field</xs:documentation>
      </xs:annotation>
      <xs:complexType>
            <xs:sequence>
                  <xs:element ref="SetFlag"/>
                  <xs:element ref="Setvalues" minOccurs="0"/>
                  <xs:element ref="Type" minOccurs="0"/>
                  <xs:element ref="Function"/>
                  <xs:element ref="Range" minOccurs="0"/>
                  <xs:element ref="Accuracy" minOccurs="0"/>
                  <xs:element ref="SpanofMembers" minOccurs="0"/>
                  <xs:element name="Coordinated" type="xs:boolean" minOccurs="0"/>
            </xs:sequence>
      </xs:complexType>
   </xs:element>
   <!-- ========================================================================
      Sensor and Actuator Models
   ========================================================================= -->
   <xs:element name="BasicSensor" abstract="false">
      <xs:annotation>
            <xs:documentation>Physical Sensor object</xs:documentation>
      </xs:annotation>
      <xs:complexType>
            <xs:sequence>
                  <xs:element ref="Query" minOccurs="0"/>
                  <xs:element ref="Datavalues" minOccurs="0"/>
                  <xs:element ref="Orientation" minOccurs="0"/>
```

```
                  <xs:element ref="BasicActuator" minOccurs="0"/>
                  <xs:element name="SensorCharacteristics" minOccurs="0">
                        <xs:complexType>
                              <xs:sequence>
                                    <xs:element ref="Query" minOccurs="0"/>
                                    <xs:element name="Vendor" type="xs:string"
minOccurs="0"/>
                                    <xs:element ref="Type" minOccurs="0"/>
                                    <xs:element ref="Accuracy" minOccurs="0"/>
                                    <xs:element ref="Range" minOccurs="0"/>
                              </xs:sequence>
                        </xs:complexType>
                  </xs:element>
            </xs:sequence>
      </xs:complexType>
   </xs:element>
   <xs:element name="BasicActuator">
      <xs:annotation>
            <xs:documentation>Physical Actuator object</xs:documentation>
      </xs:annotation>
      <xs:complexType>
            <xs:sequence>
                  <xs:element ref="Query" minOccurs="0"/>
                  <xs:element ref="Setvalues"/>
                  <xs:element ref="Orientation" minOccurs="0"/>
                  <xs:element name="AcutatorCharacterisitcs" minOccurs="0">
                        <xs:complexType>
                              <xs:sequence>
                                    <xs:element ref="Query" minOccurs="0"/>
                                    <xs:element ref="Vendor" minOccurs="0"/>
                                    <xs:element ref="Type" minOccurs="0"/>
                                    <xs:element ref="Accuracy" minOccurs="0"/>
                                    <xs:element ref="Range" minOccurs="0"/>
                              </xs:sequence>
                        </xs:complexType>
                  </xs:element>
                  <xs:element ref="SetFlag" minOccurs="0"/>
            </xs:sequence>
      </xs:complexType>
   </xs:element>
   <xs:element name="VirtualSensor">
      <xs:annotation>
            <xs:documentation>A combination of Sensor objects </xs:documentation>
      </xs:annotation>
      <xs:complexType>
            <xs:sequence>
                  <xs:element ref="Query"/>
                  <xs:element ref="Datavalues" minOccurs="0"/>
                  <xs:element ref="Function"/>
                  <xs:element ref="SpanofMembers" minOccurs="0"/>
                  <xs:element name="SensorCharacteristics" minOccurs="0">
                        <xs:complexType>
                              <xs:sequence>
                                    <xs:element ref="Query" minOccurs="0"/>
                                    <xs:element name="Vendor" type="xs:string"
minOccurs="0"/>
                                    <xs:element ref="Type" minOccurs="0"/>
```

```
                                    <xs:element ref="Accuracy" minOccurs="0"/>
                                    <xs:element ref="Range" minOccurs="0"/>
                              </xs:sequence>
                        </xs:complexType>
                  </xs:element>
                  <xs:element ref="SetFlag" minOccurs="0"/>
            </xs:sequence>
      </xs:complexType>
   </xs:element>
   <xs:element name="VirtualActuator">
      <xs:annotation>
            <xs:documentation>A combination of Actuator objects</xs:documentation>
      </xs:annotation>
      <xs:complexType>
            <xs:sequence>
                  <xs:element ref="Setvalues" minOccurs="0"/>
                  <xs:element ref="Type" minOccurs="0"/>
                  <xs:element ref="Function"/>
                  <xs:element ref="Range" minOccurs="0"/>
                  <xs:element ref="Accuracy" minOccurs="0"/>
                  <xs:element ref="SpanofMembers" minOccurs="0"/>
                  <xs:element name="Coordinated" type="xs:boolean" minOccurs="0"/>
                  <xs:element ref="SetFlag" minOccurs="0"/>
            </xs:sequence>
      </xs:complexType>
   </xs:element>
   <!-- ========================================================================
   Boolean Flags
   ======================================================================== -->
   <xs:element name="SetFlag">
      <xs:annotation>
            <xs:documentation>A binary flag that if true - sets the value or
funciton</xs:documentation>
      </xs:annotation>
      <xs:complexType>
            <xs:sequence>
                  <xs:element name="SFlag" type="xs:boolean" default="false"/>
                  <xs:element name="StartTime" type="xs:time" minOccurs="0"/>
                  <xs:element name="Duration" type="xs:duration" minOccurs="0"/>
            </xs:sequence>
      </xs:complexType>
   </xs:element>
   <xs:element name="Query">
      <xs:annotation>
            <xs:documentation>This si a generalized flag for a query</xs:documentation>
      </xs:annotation>
      <xs:complexType>
            <xs:sequence>
                  <xs:element name="QFlag" type="xs:boolean" default="false"/>
                  <xs:element name="StartTime" type="xs:dateTime" minOccurs="0"/>
                  <xs:element name="Duration" type="xs:duration" minOccurs="0"/>
            </xs:sequence>
      </xs:complexType>
   </xs:element>
   <!-- ========================================================================
   General Global Elements
   ======================================================================== -->
```

```
   <xs:element name="Type" type="gml:stringOrNull">
      <xs:annotation>
            <xs:documentation>Generic type - a string identifier</xs:documentation>
      </xs:annotation>
   </xs:element>
   <xs:element name="Vendor">
      <xs:annotation>
            <xs:documentation>Vendor Decrtiption - a string
identifier</xs:documentation>
      </xs:annotation>
      <xs:complexType>
            <xs:sequence>
                  <xs:element name="VendorDescription" type="gml:stringOrNull"
minOccurs="0"/>
                  <xs:element name="URI" type="xs:anyURI" minOccurs="0"/>
            </xs:sequence>
      </xs:complexType>
   </xs:element>
   <xs:element name="Location">
      <xs:annotation>
            <xs:documentation>Location - relative and/or absolute</xs:documentation>
      </xs:annotation>
      <xs:complexType>
            <xs:all>
                  <xs:element name="RelativeLocation">
                        <xs:complexType>
                              <xs:sequence>
                                    <xs:element ref="Coordinates"/>
                              </xs:sequence>
                        </xs:complexType>
                  </xs:element>
                  <xs:element name="AbsoluteLocation" minOccurs="0">
                        <xs:complexType>
                              <xs:sequence>
                                    <xs:element ref="Coordinates"/>
                              </xs:sequence>
                        </xs:complexType>
                  </xs:element>
            </xs:all>
      </xs:complexType>
   </xs:element>
   <xs:element name="Hardware">
      <xs:annotation>
            <xs:documentation>Hardware description</xs:documentation>
      </xs:annotation>
      <xs:complexType>
            <xs:sequence>
                  <xs:element name="Radio" type="xs:string" minOccurs="0"/>
                  <xs:element name="Processor">
                        <xs:complexType>
                              <xs:sequence>
                                    <xs:element name="ProcessorSpeed" type="xs:integer"
minOccurs="0"/>
                                    <xs:element name="ProcessorDescription"
type="xs:string"/>
                              </xs:sequence>
                        </xs:complexType>
```

```
                  </xs:element>
                  <xs:element ref="EnergySource" minOccurs="0"/>
            </xs:sequence>
      </xs:complexType>
   </xs:element>
   <xs:element name="Software">
      <xs:annotation>
            <xs:documentation>Software description</xs:documentation>
      </xs:annotation>
      <xs:complexType>
            <xs:sequence>
                  <xs:element name="OSDescription" type="xs:string" minOccurs="0"/>
                  <xs:element ref="Version" minOccurs="0"/>
            </xs:sequence>
      </xs:complexType>
   </xs:element>
   <xs:element name="Version" type="xs:string">
      <xs:annotation>
            <xs:documentation>Version Number - a string</xs:documentation>
      </xs:annotation>
   </xs:element>
   <xs:element name="Accuracy" type="xs:float">
      <xs:annotation>
            <xs:documentation>Accuracy - a number</xs:documentation>
      </xs:annotation>
   </xs:element>
   <xs:element name="Function" type="xs:string">
      <xs:annotation>
            <xs:documentation>Function - a string descriptor</xs:documentation>
      </xs:annotation>
   </xs:element>
   <xs:element name="SpanofMembers">
      <xs:annotation>
            <xs:documentation>Components of a virtual sensor or
actuator</xs:documentation>
      </xs:annotation>
      <xs:complexType>
            <xs:sequence>
                  <xs:element name="Nodeslist" minOccurs="0">
                        <xs:complexType>
                              <xs:sequence>
                                    <xs:element name="ID" type="xs:ID"
maxOccurs="unbounded"/>
                              </xs:sequence>
                        </xs:complexType>
                  </xs:element>
                  <xs:element name="Sensorlist">
                        <xs:complexType>
                              <xs:sequence>
                                    <xs:element name="SensorType" type="xs:ID"
maxOccurs="unbounded"/>
                              </xs:sequence>
                        </xs:complexType>
                  </xs:element>
            </xs:sequence>
      </xs:complexType>
   </xs:element>
```

```
   <xs:element name="Coordinates">
      <xs:annotation>
            <xs:documentation>X,Y,Z coodinate Values</xs:documentation>
      </xs:annotation>
      <xs:complexType>
            <xs:all>
                  <xs:element name="Xvalue" type="xs:float"/>
                  <xs:element name="Yvalue" type="xs:float"/>
                  <xs:element name="Zvalue" type="xs:float" minOccurs="0"/>
                  <xs:element name="Units" type="xs:string" minOccurs="0"/>
                  <xs:element ref="Timestamp" minOccurs="0"/>
            </xs:all>
      </xs:complexType>
   </xs:element>
   <xs:element name="Datavalues">
      <xs:annotation>
            <xs:documentation>Reading of a sensor Value</xs:documentation>
      </xs:annotation>
      <xs:complexType>
            <xs:sequence>
                  <xs:element name="Timestamp" type="xs:dateTime" minOccurs="0"/>
                  <xs:element name="Value-int" type="gml:integerOrNullList"
minOccurs="0" maxOccurs="unbounded"/>
                  <xs:element name="Value-flt" type="gml:doubleOrNullList"
minOccurs="0" maxOccurs="unbounded"/>
                  <xs:element name="Value-string" type="gml:stringOrNull"
minOccurs="0"/>
            </xs:sequence>
      </xs:complexType>
   </xs:element>
   <xs:element name="Setvalues">
      <xs:annotation>
            <xs:documentation>Reading of a actuator seting </xs:documentation>
      </xs:annotation>
      <xs:complexType>
            <xs:sequence>
                  <xs:element name="Timestamp" type="xs:dateTime" minOccurs="0"/>
                  <xs:element name="value-int" type="gml:integerList" minOccurs="0"
maxOccurs="unbounded"/>
                  <xs:element name="value-flt" type="gml:doubleList" minOccurs="0"
maxOccurs="unbounded"/>
            </xs:sequence>
      </xs:complexType>
   </xs:element>
   <xs:element name="Timestamp">
      <xs:annotation>
            <xs:documentation>Time Stamp</xs:documentation>
      </xs:annotation>
      <xs:complexType>
            <xs:sequence>
                  <xs:element name="Date" type="xs:date" minOccurs="0"/>
                  <xs:element name="Time" type="xs:time"/>
            </xs:sequence>
      </xs:complexType>
   </xs:element>
   <xs:element name="EnergySource">
      <xs:annotation>
```

```
            <xs:documentation>The elements associated with power</xs:documentation>
      </xs:annotation>
      <xs:complexType>
            <xs:sequence>
                  <xs:element name="Units" type="xs:string" default="Joules"
minOccurs="0"/>
                  <xs:element name="PowerLevel" type="xs:integer" minOccurs="0"/>
                  <xs:element name="SourceDescription" type="xs:string"/>
            </xs:sequence>
      </xs:complexType>
   </xs:element>
   <xs:element name="Orientation">
      <xs:annotation>
            <xs:documentation>Orientation using Euler Coordinates X,Y,Z (Pitch, Yaw,
Roll in Aeronautics terms</xs:documentation>
      </xs:annotation>
      <xs:complexType>
            <xs:sequence>
                  <xs:element name="Absolute" minOccurs="0">
                        <xs:complexType>
                              <xs:sequence>
                                    <xs:element name="Xvalue" type="xs:integer"/>
                                    <xs:element name="Yvalue" type="xs:integer"/>
                                    <xs:element name="Zvalue" type="xs:integer"/>
                              </xs:sequence>
                        </xs:complexType>
                  </xs:element>
                  <xs:element name="Quatenions" minOccurs="0">
                        <xs:complexType>
                              <xs:sequence>
                                    <xs:element name="Xvalue" type="xs:integer"/>
                                    <xs:element name="Yvalue" type="xs:integer"/>
                                    <xs:element name="Zvalue" type="xs:integer"/>
                                    <xs:element name="Svlaue" type="xs:integer"/>
                              </xs:sequence>
                        </xs:complexType>
                  </xs:element>
                  <xs:element name="Euler" minOccurs="0">
                        <xs:complexType>
                              <xs:sequence>
                                    <xs:element name="Pitch" type="xs:integer"/>
                                    <xs:element name="Roll" type="xs:integer"/>
                                    <xs:element name="Yaw" type="xs:integer"/>
                              </xs:sequence>
                        </xs:complexType>
                  </xs:element>
                  <xs:element ref="Query" minOccurs="0"/>
            </xs:sequence>
      </xs:complexType>
   </xs:element>
   <xs:element name="Range">
      <xs:annotation>
            <xs:documentation>Range of a Sensor</xs:documentation>
      </xs:annotation>
      <xs:complexType>
            <xs:all>
                  <xs:element name="Unit" type="xs:string" minOccurs="0"/>
```

```
                <xs:element name="Distance" type="xs:float" minOccurs="0"/>
            </xs:all>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

References

[1] http://www.w3.org/XML/
[2] Richard Wagner and Richard Mansfield, XML All-in-One Desk Reference for Dummies, Wiley Publishing, 2003
[3] Grid Computing – Making the Global Infrastructure a Reality, Edited by Fran Berman, Geoffrey C. Fox and Anthony J.G. Hey, John Wiley and Sons Publishers, 2003
[4] http://www.gis-news.de/xml/gml.htm
[5] OGC Announces OpenGIS Geography Markup Language Implementation Specification (GML 3). Version 3.0 for the *OpenGIS Geography Markup Language (GML) Implementation Specification*.. Edited by Simon Cox, Paul Daisey, Ron Lake, Clemens Portele, and Arliss Whiteside, At http://xml.coverpages.org/geographyML.html.
[6] Sensor Model Language (SensorML) - http://vast.uah.edu/SensorML/
[7] Sensor Model Language (SensorML)
for In-situ and Remote Sensors, Reference number of this OpenGIS © Project Document: OGC 02-026r4, Version: 0.7, Date: 2002-12-20, Editor: Mike Botts
[8] The Cougar Approach to In-Network Query Processing in Sensor Networks. - http://cougar.cs.cornell.edu/
[9] "Design and Evaluation of a Query Processing Architecture for Sensor Networks", Thesis of Samuel Ross Madden, Fall 2003. At http://www.cs.berkeley.edu/~madden/thesis.pdf)
[10] TinyOS release 1.1 available at ….