Exploiting Web Technologies to Build Autonomic Wireless Sensor Networks

Flávia C. Delicato[2], Luci Pirmez[1], Paulo F. Pires[2], José Ferreira de Rezende[3]
[1]NCE – Federal University of Rio de Janeiro, Brazil
[2]DIMAp - Federal University of Rio Grande do Norte, Brazil
[3]GTA - Federal University of Rio de Janeiro, Brazil
{fdelicato, luci, paulopires }@nce.ufrj.br; rezende@gta.ufrj.br

Abstract. Most of the current wireless sensor networks are built for specific applications, with a tight coupling between them and the underlying communication protocols. We present a more flexible architectural approach for building WSNs, in which application-specific features are decoupled from the underlying communication infrastructure, although affecting the network behavior. We propose a framework based on Web technologies that provides a standard interface for accessing the network and configurable service components tailored to meet different application requirements, while optimizing the network scarce resources. Also, a set of ontologies is defined as part of the framework for representing shared knowledge of the WSN domain.

1 Introduction

Wireless sensor networks (WSNs) are distributed systems composed of hundred to thousands of low cost, battery-powered and reduced size devices, endowed of processing, sensing and wireless communication capabilities.

One major reason for the increasing interest in Wireless sensor networks (WSNs) in the last few years is their potential usage in a wide range of application areas such as health, military, habitat monitoring and security [1]. However, before WSNs can be widely employed, they must be cheaper, easier to use and more flexible than they are at present.

Currently existent software for WSNs is not flexible enough to meet the different demands of their potential applications. Most of existing WSNs require loading application code in sensor nodes before they can be deployed. Once the WSN becomes operational, applications can be only slightly modified to adapt their behavior to changes in the execution context. By context, we mean everything that can influence the behavior of an application [1]. We can distinguish three specific levels of awareness, in the context of WSNs: *device awareness*, *environment awareness* and *application awareness*. Device awareness refers to everything that lies on the physical device the application relies on, such as memory, battery, processing power and so on. Environment awareness refers to everything that is external to the physical device, such as bandwidth, network connectivity, location, neighboring nodes, and so on. Finally, application awareness refers to all

application-specific information, such as QoS parameters, values of sensor-collected data, query and sensing task descriptions.

It is important to point out that the WSN execution context is extremely dynamic, since it mirrors the network and application states. WSN nodes can have their battery depleted, new nodes can be added in the network after the initial deployment, and both bandwidth and the quality of wireless links are likely to change a lot along a sensing task lifetime. Furthermore, the application interests may be truly dynamic, with query parameters and QoS requirements changing along the time.

Several works, such as [3], highlights the close relationship among application requirements and the WSN performance. Energy saving is a key issue in WSN environments, which dictates the network operational lifetime. Application-specific optimizations may increase the WSN overall performance, mainly regarding the energy consumption. At the same time, such optimizations assure that a minimum level of QoS is provided to applications. Each class of application has specific features and different QoS requirements and it is best served by a different network configuration. WSN configuration comprises the network logical topology and the data dissemination protocol, among other factors. To sum up, the careful choice of the WSN configuration may increase the efficiency from both the network and the application point of views. However, configuration decisions should not be left in charge of application developers, which should deal with a higher abstraction level. Therefore, applications should be able to interact with the WSN through a standard high-level programming interface (API). Through this API the clients can issue their queries and QoS parameters, dynamically monitor or modify the network behavior according to their variable requirements and receive the sensor-collected data.

Furthermore, most applications need to directly access the data generated and pre-processed by the WSN in order to use it as inputs for their internal analysis and processing tasks. Such kind of interaction characterizes an application-to-application interface. Web Services technologies [4] have being successfully adopted as a feasible solution for enabling such kind of application interoperability.

The pervasiveness and the wireless nature of sensor devices require network architectures to support *ad hoc* configuration. A key technology of true *ad hoc* networks is service discovery, functionality by which "services" (functions offered by nodes) can be described, advertised, and discovered by other devices or applications. All the current service discovery and capability description mechanisms are based on *ad hoc* representation schemes and rely on standardization. A crucial requirement for the future, widely accessed WSNs is interoperability under unpredictable conditions, i.e., networks which were not designed for specific, predefined purposes, should be able to be accessed by different applications, which dynamically discover their functionality and take advantage of it. The tasks involved in the dynamic utilization of WSN services involve service discovery and description. Service description may involve representing information about the sensing task and QoS parameters. Thus, an ontology language is useful to describe the characteristics of WSN devices, their sensing capabilities, and specific information of applications accessing the WSN.

We propose a framework based on Web technologies for designing WSNs. The framework has three main goals: (i) to establish a programming model for WSNs,

aiming to standardize the design and the interoperability of applications and to increase the flexibility of network usage; (ii) to provide a standard interface for accessing the WSN, allowing both the retrieval of information about the execution context, and the submission of tasks and requirements to adjust the network behavior according to each application; and (iii) to supply components that offer functionalities needed by WSN applications from different domains.

The proposed programming model follows a service-based approach in which WSNs are service providers to client applications. The interface provided by the framework for accessing the network is implemented by a communication service based on Web services technologies. The set of network-supplied services is published and accessed through an XML-based language [5]. XML messages exchanged between the WSN and applications are formatted and packed through the SOAP protocol [6].

The functionalities provided by the framework are implemented as a set of configurable service components. These components are responsible for managing the network behavior such that requirements of different applications can be met, while the network resources are optimized. The communication service can be customized through extension mechanisms, allowing different communication protocols, devices and services to be seamlessly incorporated in the WSN architecture.

The proposed framework provides mechanisms to acquire, to reason about and to adapt the WSN behavior according to context information. Such capability allows sensor nodes to maintain consistent contextual knowledge and change their behavior according to it. Such knowledge is achieved through sharing context information among different entities of a WSN system, namely, sensor nodes, applications and infrastructure components. Sensors monitor and periodically send context information. Applications inspect the current context and eventually change previously stated execution policies. Framework service components must guarantee that defined QoS parameters are met, and that the current execution context is valid. To meet this goal the framework adopts a set of common ontologies to support the communication among the several entities that comprise the WSN domain.

The main benefits of the proposed framework can be assessed under two points of view: the application and the WSN. From the application point of view, the framework provides an abstraction of a generic WSN, which offers services for several application domains, with different requirements. Such services are accessed in a flexible way and through different high level programming languages (Java, C++, etc), according to the application developer choice. The utilization of the proposed framework allows building autonomic WSNs, customized according to specific application needs. At the same time, it leverages the development of custom and context-aware applications for WSN environment. Another important aspect is that, through the use of XML language and the SOAP protocol, both *de facto* Web standards, the proposed framework naturally provides interoperability between the WSN and the Internet.

From the network point of view, the framework services supply mechanisms to obtain the best match between network configuration and application requirements, as well as to inspect and dynamically adapt the network behavior according to

changes in the execution environment. By using these services it is possible to reach high network efficiency in terms of energy consumption.

Although the framework provides several service components, the focus of this work is mainly in the communication service, which adopts Web technologies. Therefore, such component is described in details and evaluated. The remainder of this paper is divided as follows. Section 2 presents related works. Section 3 details the proposed framework. Section 4 describes the developed system prototype and Section 5 concludes the paper.

2 Related Work

Our work has features that distinguish it from other existent proposals for designing WSNs. First of all, it proposes a service-based approach for the WSN design, in which all interactions among applications and the underlying wireless network rely on a consumer-provider relationship. Such approach was inspired in the area of Web Services [4] and its major advantage is offering a flexible and generic programming model for WSNs. In contrast with the service-based approach, there are works that propose database approaches [7, 8] or event-based programming models [9] for designing WSNs.

The second particular feature of our work is the proposal of a high-level interface for accessing the network. This interface, instead of relying on proprietary formats or languages, such as [9], which adopts a proprietary procedural script language, provides a standard mechanism for representing data and formatting messages exchanged between applications and the network. The adoption of the ubiquitous standards XML and SOAP provides high portability and flexibility to WSN applications. The XML high degree of extensibility allows it to be adopted both as a query and as a tasking language. On the other hand, the adoption of proprietary languages hinders interoperability among applications and WSNs.

The third feature of our framework is that it provides configurable service components that allow dynamically configuring and customize the underlying network infrastructure and protocols. In spite of there being other works [2] that share such a goal, these works do not address the issue of representing and interpreting application requirements. Another relevant feature of our framework is the use of mechanisms for providing applications with context awareness. Such feature is also supplied by CARISMA [2], which, however, was designed for generic wireless networks, and does not address WSN specific requirements.

In Garnet [10], an architectural framework for WSNs is presented. Garnet focuses on problems regarding the management of data-streams in the context of WSNs. The mechanisms proposed in Garnet are complementary to our work, in the sense that they can be incorporated as service components in our framework.

Regarding the use of ontologies, a pioneer work defines an ontology for sensor nodes [11], which seeks to capture the most important sensor features to describe their functionalities and current state. Such work has similarities with ours in the sense that the contextual information is described by ontologies, with the goal of adapting the WSN behavior to different execution states. However, differently of our

proposal, they do not employ ontologies neither for WSN service discovery nor for definition of execution policies by the application

3 Framework Description

This work proposes a framework based on Web technologies for designing WSNs. The framework supplies the basic underpinning for building flexible and configurable WSNs. Such framework comprises: (i) a service-based programming model for WSNs; (ii) a standard interface for accessing the network services; and (iii) a set of configurable service components to aid the application development and to control the network behavior during the execution of submitted sensing tasks.

The proposed framework can be described according to its logical and physical models. The logical model includes: (i) the description of the services provided by the framework to support the execution of WSN applications; (ii) the specification of logical components which supply such services; and (iii) the description of the interfaces of such components among each other, with applications and with the underlying network infrastructure. The framework physical model includes the detailed description of its components, according to the chosen implementation technologies. Although the framework provides several service components needed to the efficient operation of WSNs, this paper focuses on the communication service.

3.2 Logical Model

The basic service provided by WSNs is collecting environmental data and delivering it to applications. Such delivery depends on: (i) the discovery of the WSN capabilities; (ii) the request of data by applications and (ii) the way through which the communication among data producers (sensor nodes) and data consumers (applications) takes place. Our framework provides applications with an abstraction of such delivery service so that it can be configured according to different needs. This abstraction is supplied by the **communication service** that, among other functions, provides applications with a high level interface for accessing the WSN services. The discovery of the WSN sensing capabilities is accomplished through the **discovery service**. This service allows WSNs to advertise their capabilities and applications to advertise a high level description of their sensing requirements. Furthermore, the discovery service accomplishes a matching function between sensing capabilities and application requests in order to allow applications to find suitable WSNs.

One of the goals of our framework is to facilitate the task of application developers, by dealing with low-level issues regarding network infrastructure and protocols. With this intention in mind, the framework supplies a **configuration service**, detailed in [12], responsible for the choice and setting of network protocols, as well as a service for **active node selection** [13], responsible for the choice of sensor nodes that should be activated to accomplish a given sensing task. Once the framework takes these low level decisions, it should control the execution of the received sensing tasks, and manage the utilization of the network resources. In order

to perform these tasks, **resource management** and **admission control services** are supplied. To deal with the highly dynamic execution context of WSNs, the framework also provides **inspection and adaptation services**. Furthermore, components of generic services are provided, which are useful for all WSNs.

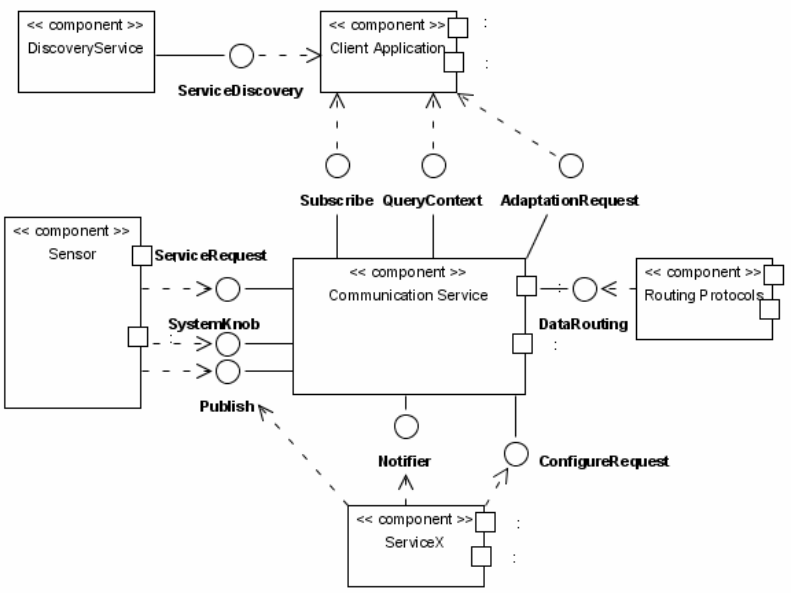

Figure 1: Logical Components and Interfaces

The functionalities of the framework services are provided by a set of components. Figure 1 depicts the main logical components and the interfaces among each other and with the external world (applications, sensor nodes and network protocols). The depicted interfaces represent units of service provision offered by their respective component. The main component is the communication service. All the other services are directly connected to the communication service, except the discovery service. Additional services can be seamlessly integrated into the system, provided that they are connected to the communication service through its interfaces. In the following paragraph, the functions of the main interfaces are summed up.

`ServiceDiscovery` Interface is used by applications to find out a WSN able to perform their sensing task. **`Subscribe` Interface** contains service primitives: (i) to allow the application to submit a query or task (`SubscribeInterest`); (ii) to notify the application about sensor published data, in the case of asynchronous queries (`ReceiveResults`); (iii) to allow the application to submit an execution policy, which consists of information on the WSN behavior while executing the requested task (`SubmitExecPolicy`); and (iv) to allow the application to submit QoS parameters to be met during the execution of the requested task

(`SubmitQoSParameters`). **`QueryContext` Interface** allows applications to inspect the WSN state. **`AdaptationRequest` Interface** is used by applications to request the activation of an adaptation policy in order to recover the network from a state of QoS violation. **`Publish` Interface** allows sensor nodes to advertise their capabilities (`PublishConfiguration` primitive) and to publish their collected data (`PublishData`), and to allow service components to publish results of their processing tasks (`PublishResults` primitive). **`Notifier` Interface** contains primitives to notify framework services about data or application subscription arrivals. **`ConfigureRequest` Interface** is used by a service component to indicate how a sensor node may request its execution. **`DataRouting` Interface** contains service primitives used for exchanging data between the framework and the underlying network routing protocols. **`ServiceRequest` Interface** is used by sensor nodes to request a framework service that requires an explicit request to be activated. **`SystemKnobs` Interface** allows the framework services to interact with hardware devices, such as processor, sensor, radio transmitters and receivers. It contains service primitives that allow modifying the current state of such devices, enabling a fine-grained control over the network operation.

3.2 Physical Model

The proposed framework makes use of Web Services technologies in the specification and implementation of its components. The adoption of a Web service-based approach was motivated by the fact that such an approach relies on protocols and languages largely used on the Web, thus facilitating the interoperability among the system and client applications. Most of the WSN client applications need gather the data generated and pre-processed by the network as input for their own analysis and processing software. Such interaction characterizes an application-to-application interface. Web Services architecture supplies a feasible solution to enable that kind of application interoperability.

The use of Web Services technologies in both the physical model and the framework implementation implies that framework services are exposed as Web Services. The logical interfaces presented in Section 3.1 are physically described through WSDL documents [14] and XML schemas [5]. Messages exchanged among the external and internal components (in other words, the service primitives) are implemented as SOAP or XML messages.

Regarding the physical constraints to be hold in the framework design, sinks are devices more robust than sensors and represent the network access point. Therefore, they are fully designed following the Web Services architecture. Thus, the implementation of the communication service in sink nodes is based on the SOAP protocol. To avoid the overhead imposed by SOAP, the implementation of the communication service inside sensors is based on lightweight XML messages, and formatted accordingly to specific schemas, which aim to generate XML messages as compact as possible. Optionally, in order to further reduce the communication overhead, the XML binary format, WBXML [15], can be used inside the network. The framework service components are implemented as software modules. These components are described in the next sections.

3.2.1 Communication Module

This module is responsible for the communication service implementation. It includes a SOAP proxy responsible for the interaction with client applications and XML drivers for communicating with the underlying network protocols and devices. SOAP proxies are programs that translate function calls in the application programming language to SOAP messages so as to invoke respective operations of the network services. Conversely, SOAP reply messages are converted to data and function calls in the application programming language. SOAP proxies use primitives of the **`Subscribe`**, **`QueryContext`** and **`AdaptationRequest`** logical interfaces, which are implemented as operations of the communication module and are described in the WSN WSDL document.

Similarly to SOAP proxies, drivers for different protocols consist of software modules that convert functions calls, according to the programming language of a given communication protocol, in operation calls defined according to the framework API, and vice-versa. In the same way, drivers convert XML messages generated by the framework components in proprietary data formats of the communication protocol, and vice-versa.

To provide the application requested QoS and to enable the adaptation of the network behavior according to execution contexts, the framework components should be able to directly interact with hardware devices, such as sensors and radio transmitters. This requirement is met by specifying an API that abstracts the behavior of the WSN configurable parameters (communication protocols and devices). This API consists in the implementation of the **`SystemKnobs`** logical interface. The device manufacturers enable low-level interactions with devices by supplying a service API. Drivers for the conversion between framework and devices APIs can be built from the supplied specifications.

The processing of a data message arriving in a WSN node, and its subsequent forwarding to the framework services, are performed by SOAP-based components in sink nodes and by XML-based components in sensor nodes. The communication module also contains an ontology database and a reasoning engine, which comprise the semantics components of the framework.

SOAP-based Components. The communication module in sink nodes is composed of a SOAP engine and a set of handlers. The SOAP engine is responsible for coordinating the SOAP messages flow through the several handlers and for guaranteeing that the semantics of SOAP protocol is respected. Handlers represent the logic of message processing and act as dispatchers for the several services supplied by the framework. Handlers intercept SOAP messages, parse the message-header fields indicating the services that are to be executed over the data packet, and dispatch the packet for the components that implement such services.

XML-based Components. To avoid the overhead of the SOAP protocol, XML messages exchanged inside the network are formatted according to a specifically designed schema. Such schema is a lighter SOAP-like specification that generates more compact messages than the original SOAP protocol. The XML-based communication module is the counterpart, in sensor nodes, of the SOAP-based communication module in sinks. It is composed of a message dispatcher, which is a

lighter version of the SOAP engine, and a set of handlers, responsible for forwarding messages to the service components provided by sensor nodes.

Knowledge base. Corresponds to the ontology database. It contains the adopted ontology model, that is, the definitions of the classes and properties created for describing sensor features, execution contexts and policies, application queries and tasks. The full database is implemented only in sink nodes. Sensor nodes keep a sub-part of the ontology definitions needed for representing their own capabilities and information about execution contexts.

Reasoning engine. A software module responsible for reasoning with ontology knowledge, that is, static knowledge derived from the underlying ontology model. Its function is to decide whether WSN nodes can meet the requirements of a submitted application task.

3.2.2 Service Modules

The service interfaces offered by the framework to the external world (applications and other services) are described through WSDL documents. Therefore, a WSDL document defines the format of messages used to submit application interests and QoS requirements and to request inspection or adaptation of the network behavior (Figure 2). The framework also provides a WSDL document that describes the interfaces to be used by service developers to incorporate new services in the WSN architecture. Furthermore, XML schemas are provided which describe the interfaces to be used by network protocol developers. Primitives described in the logical interfaces correspond to definitions of service operations, which are invoked through SOAP or XML messages.

The external discovery module of the framework discovery service allows applications to find both the location of a potentially interesting WSN and the format of messages to interact with it. The use of SOAP and XML, both part of the Web Services architecture, makes UDDI [4] a natural choice as the discovery protocol to be used by applications.

Sink nodes implement a Web Service comprising all the functionalities offered by the WSN and they keep a repository containing the WSDL documents that describe the Web Service interfaces. To access a WSN, an application initially locates the WSN sink node through UDDI and then obtains the WSDL document to learn the message format to communicate with the network. Therefore, the external discovery module is composed of WSDL documents and of the necessary specifications for publishing the WSN Web Service in the UDDI registry.

The **configuration service** is implemented only in sink nodes, by the **decision module**. The **inspection and adaptation service** is implemented as two independent modules: (i) the **inspection module**, that allows the application to inspect the network behavior at runtime, supplying a representation of the current execution state; and (ii) the **monitoring and adaptation module**, responsible for monitoring the states of the network and application and for activating adaptation policies whenever it is necessary or requested by the application. The **monitoring and adaptation module** accesses the local ontology database and, similarly to the communication module, it contains a reasoning engine. This engine is responsible for reasoning with both ontology knowledge and contextual knowledge. Contextual

knowledge is a dynamic knowledge that is inferred from situational information reported by sensor nodes. Once that information is available, the module verifies if the WSN execution context at every given moment represents a valid state. Otherwise, a predefined adaptation policy is triggered to repair the network state.

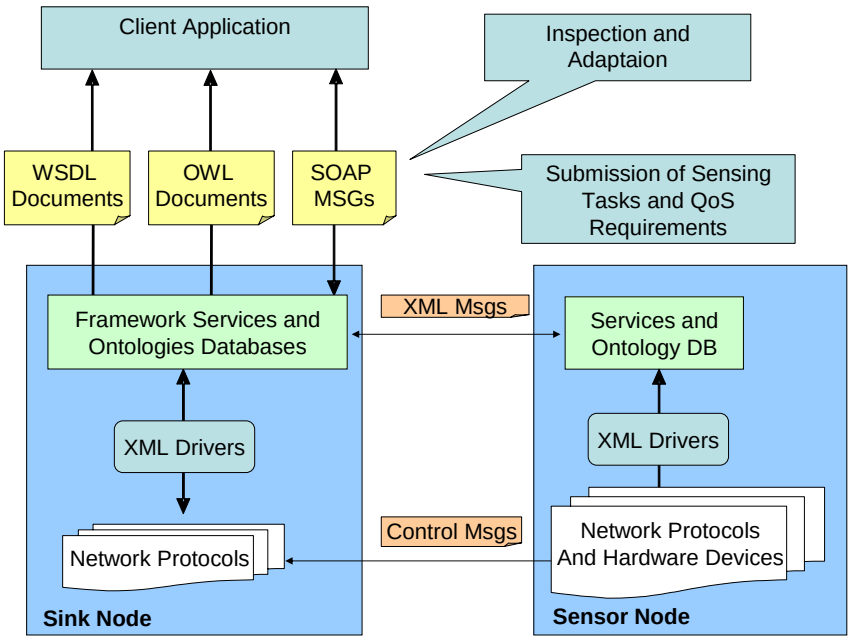

Figure 2: Communication among the framework components

The inspection module is implemented only in sink nodes and the monitoring and adaptation module is implemented both in sink and sensor nodes. The **resource and task management service** is also implemented as two independent modules: an **admission control module** and an **active node selection module**, both implemented only in sink nodes. Each generic service supplied by the framework is implemented as a separate module in the proposed system. Modules of generic services can be implemented and supplied by third parts. The use of XML-based APIs in the framework design allows services to be easily incorporated to the WSN system, provided that they implement the defined interfaces.

3.2.3 WSN Ontology

We detected three situations in which it would be worthwhile to add semantics in the context of accessing and using WSNs:

- To locate networks that potentially meet the interests of an application, given a high level description of the requested services. The goal here is to discover the address (URL) of the access point (sink) of such WSNs, through which applications are able to access and use the WSN services. In this case, the UDDI protocol, used for discovering WSNs, should be added with semantic capacities. Addressing such situation is out of the scope of our work.
- Once a specific WSN has been chosen, to determine if the sensing task requested by the application can be fully accomplished by such network, given the task detailed description, including QoS requirements.
- Once the task has been initiated, to share knowledge on the execution context, allowing (i) sensor nodes to send information about the network and the application current states; (ii) applications to monitor such state; and (ii) service components to verify if a given execution state is valid and the eventual need of triggering adaptation mechanisms in case of violation of QoS parameters.

We designed a WSN ontology to capture the most relevant features of sensors, execution context and application requirements for the purposes of service discovery and context monitoring (items 2 and 3 above). Therefore, we created classes and properties to describe concepts related to the descriptions of sensor node capabilities, application tasks and policies, and execution contexts. For purposes of service discovery, we defined: (i) three main classes for describing WSN features (*WSN*, *SensorNode* and *SensorField*); and (ii) four main classes for describing application requirements (*Task*, *Query*, *QoSParameters* and *SensorType*). For purposes of describing execution policies and contexts, and verifying if the current state fits in a valid policy, the main classes we created are: *ExecutionContext*, *ApplicationState*, *NetworkState*, *ExecutionPolicy* and *CurrentState*. The defined ontologies are concisely depicted in Figures 3 and 4. The framework reasoning engines have a set of rules that allows reasoning based on such ontologies.

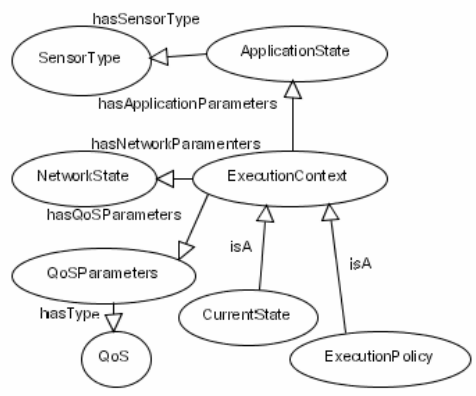

Figure 3: Main ontologies for execution policies and contexts

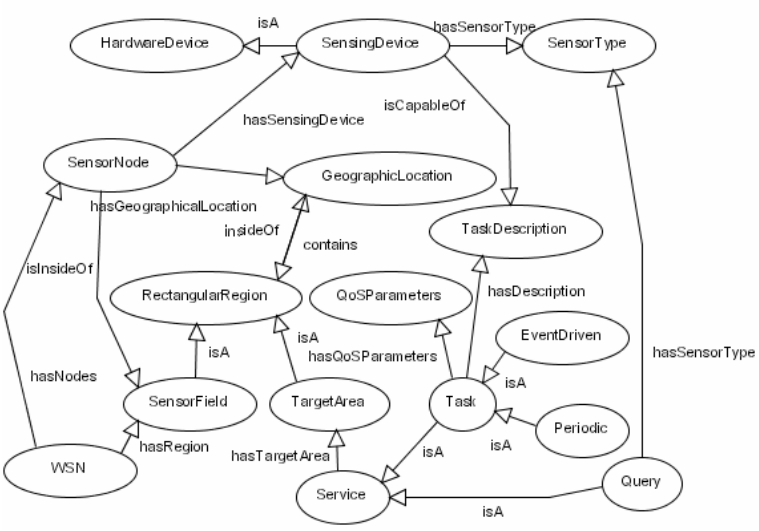

Figure 4: Main ontologies for tasks and WSN descriptions

The ontologies designed for the WSN domain were defined by using the OWL/RDF language [16]. The Web Ontology Language (OWL) is intended to provide a language to describe the classes and relations between them that are inherent in Web documents and applications. The OWL language can be used (i) to formalize a domain by defining classes and properties of those classes; (ii) to define individuals and assert properties about them, and (iii) to reason about these classes and individuals to the degree permitted by the formal semantics of the OWL language. We will not present the OWL files containing the complete description of

ontologies, for lacking of space. The OWL-DL [16] version of the language was used for representing the ontologies stored on the sink node knowledge base and the OWL-Lite [16] version for ontologies on the sensor node database.

4 Framework Prototype

As a proof of concept for the proposed framework, a prototype was constructed which implements its main building blocks. The goals of building the prototype were to validate the high-level interaction among applications and the WSN, according to the proposed programming model, and to establish a basis for evaluating the system requirements in terms of memory and processing power. From the implemented prototype it is possible to infer the feasibility of developing WSN applications based on the proposed framework with the currently existent hardware for sensor nodes.

The prototype was implemented in Java programming language. Since the hardware device features and, hence, the software components to be deployed in each device largely differ between sink and sensor nodes, two development platforms were used to implement each type of node.

The implementation of the prototype was divided into two phases. The goal of the first phase consisted of modeling and implementing the communication module in the sink nodes. The implementation of such module allowed to validate the high level interaction among applications and the WSN and to check the calls of operations supplied by the network Web Service. The goal of the second phase was to analyze the computational load of the communication module in sensor nodes. In that phase, the adoption of messages in both XML and WBXML formats was evaluated, in order to compare their performance, in terms of network traffic and memory consumption. A simple generic service component was also implemented to evaluate the amount of memory spent by sensor devices to receive XML messages, process and forward such messages through the different handlers and deliver them to the module that implements the requested service. The implemented service was data aggregation. Since data aggregation is a basic operation, required by all classes of current WSN applications, our work leverages the aggregation function as a first class operation supplied as one of the framework service components.

The prototype of sink nodes was executed in a Pentium 4 1.8 GHZ with 1.5 GB of RAM and 40 GB HD. The WSN Web Service and the classes representing the functionalities needed for the sink node were implemented using Apache Axis platform for Web Services [17] and J2SE 1.4.2_01. The document describing the WSN services was written in WSDL language. The previously described operations SubscribeInterest, PublishConfiguration and PublishData were provided by the implemented Web Service. Operations are invoked through SOAP messages.

As it was previously described, applications access the WSN by using SOAP proxies. In the developed prototype, the client application was implemented in Java language. Therefore, the WSDL2Java tool [17] was used for generating the Java proxy for accessing the network. WSDL2Java tool is supplied along with the Axis package and it consists of a Java class which receives as input a WSDL document

representing a Web Service and it generates method calls in Java corresponding to the invocation of the respective service operations.

In order to validate the communication between client applications and the WSN, an event-driven application was implemented as a Java application that emits queries for the network and receives the results. The application, after obtaining the URL of the sink node through UDDI, gets a reference for the WSN Web Service and invokes the operation of interest submission, representing an asynchronous query. Sensing data meeting the application interests are reported by sensors through the PublishData operation and delivered to the application through the ReceiveResults operation.

In order to evaluate the feasibility of implementing the proposed framework in sensor nodes, we emulated the hardware of WINS NG 2.0 [18] nodes, which are endowed of SH-4 167 MHz processors and have 32 MBytes of RAM. The framework components in sensor nodes were implemented by using the J2ME (Java Micro-edition) platform, with CLDC configuration (Connected Limited Device Configuration) and MIDP profile (Mobile Information Device Profile). CLDC configuration defines the base set of application programming interfaces and a virtual machine for resource-constrained devices often connected through a wireless network. MIDP profile was designed to mobile and cellular phones and it offers a set of basic functionalities needed for mobile applications.

The *Wireless Toolkit* [19] was used for building the sensor node prototype. The Wireless Toolkit is a toolbox for developing wireless applications that are based on J2ME Connected Limited Device Configuration and Mobile Information Device Profile, and it was designed to run on cell phones, personal digital assistants, and other small mobile devices. The toolkit includes the emulation environments, performance optimization and tuning features. The kxml package [20] was used for implementing the communication module in sensor nodes. Kxml provides an XML parser and an XML writer light enough to run in the J2ME platform. Besides the communication module, an aggregation service module was implemented, which offers methods for accomplishing MAX, MIN and AVERAGE functions. The size of the ".jar" file (Java deployment format) composed of all classes needed for a sensor node running the proposed framework is 90kBytes, including all libraries. Such size is perfectly compatible with the memory resources of our target sensor nodes.

Table 1: Measurements performed using XML and WBXML formats (in bytes)

Method/Operation	XML Format		WBXML Format	
	Memory	Network traffic	Memory	Network traffic
PubContent (send)	10748	139	8152	47
PubContent (reception/forward)	10344	139	5964	47
AdvInterest (recep/forw)	33216	178	21140	57
PubData (send)	13136	42	9396	24
PubData (recep/forw)	6136	42	4016	24

To deal with the XML verbosity, there are binary versions suitable for resource-constrained environments. One widely used binary format is WBXML [15]. The kxml package includes support to handle WBXML messages. We implemented

XML and WBXML formats, in order to compare their performances in terms of number of bytes transmitted in the network and memory consumption on devices.

Wireless Toolkit includes tools for monitoring: (i) the frequency of use and execution time for every application method; (ii) the usage of memory while the application runs; and (iii) network data transmitted and received by the application. Such tools were used to perform measurements with the prototype. The methods of the prototype code were grouped according to the corresponding working stage of the network. Table 1 shows the results obtained by the memory and network monitors. In the table, PubContent refers to the stage of internal service discovery, in which sensors exchange messages describing their capabilities. AdvInterest refers to the stage of task submission for the application. PubData refers to the stage of data sending for the sensors. Results represent the measurements of one single node, that is, its individual memory consumption and the network traffic generated by it, when accomplishing each one of the mentioned operations. The performed measurements shown that the adoption of WBXML format resulted in a decrease of around 30% in the memory consumption and around 70% in the network traffic, in comparison to the XML format. Therefore, WBXML represents a better choice for representing message exchanging inside the WSN. It is worthwhile mentioning that message sizes in the WBXML format is lower even then message sizes of well known WSN protocols that adopt proprietary binary formats [21].

5 Conclusions

The proposed framework has the goal of acting as the underpinning for developing a new, more flexible and easier to use architectural approach for WSNs. Such approach is suitable for building the envisioned autonomic WSNs of the near future. WSNs have historically been built with a strong coupling among applications and the underlying network infrastructure. Such architectural approach is justified by the need to achieve energy efficiency. However, not only are these solutions proprietary, but they generate rigid systems, with WSNs specifically designed to particular applications. This scenario is not desirable, considering the costs of the infrastructure deployment, the potentially long operational lifetime of the network and its capability of serving several classes of applications. Therefore, rather than being coupled to specific applications and often built by the same application development team, future WSNs should be designed with a flexible architecture, satisfying the demands of a broad range of applications from different groups of users.

Acknowledgements

This work was supported by the Brazilian Research Council, CNPq.

References

1. I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, Wireless sensor networks: a survey. *Computer Networks*, **38** (4):393–422 (2002).
2. L. Capra, W. Emmerich, C. Mascolo, Reflective Middleware Solutions for Context-Aware Applications in: Proceedings of the Reflection2001, Japan, pp. 126-133, 2001.
3. J. Heideman, F. Silva, and D. Estrin, Matching Data Dissemination Algorithms to Application Requirements. in: Proceedings of the the ACM SenSys, pp. 218-229, USA, Nov. 2003.
4. S. Graham, et al., *Building Web Services with Java: Making Sense of XML, SOAP, WSDL, and UDDI*. (Sams Publisher, 2002).
5. W3C Recommendation (February 4, 2004) "Extensible Markup Language (XML) 1.0 (Third Edition)"; http://www.w3.org/TR/REC-xml.
6. W3C Recommendation (June 4, 2003) "SOAP version 1.2."; http://www.w3.org/TR/soap12-part0/.
7. P. Bonnet, J. E. Gehrke, and P. Seshadri, Towards Sensor Database Systems. in: Proceedings of the 2nd International Conference on Mobile Data Management, Hong Kong, Jan. 2001.
8. Cougar Project (April 5, 2006); http://www.cs.cornell.edu/database/cougar/.
9. C. Shen, C. Srisathapornphat, C. Jaikaeo, Sensor Information Networking Architecture and Applications, *IEEE Personal Communications* v. 8, pp.52–59, Aug. 2001.
10. L. St. Ville, P. Dickman, Garnet: A Middleware Architecture for Distributing Data Streams Originating in Wireless Sensor Networks. in: Proceedings of the First International Workshop on Data Distribution for Real-Time Systems (DDRTS'03), Providence, Rhode Island, USA, May 2003, pp.235-240.
11. S. Avancha, C. Patel, and A. Joshi, Ontology-driven Adaptive Sensor Networks, in: Proceedings of the First Annual International Conference on Mobile and Ubiquitous Systems: Networking and Services (MobiQuitous'04), Boston, Massachusetts, August 2004, pp. 194-202.
12. F. Delicato, et al. Service Oriented Middleware for Wireless Sensor Networks (in Portuguese). Nucleo de Computação Eletrônica – Federal University of Rio de Janeiro Technical Report No.NCE04/04.
13. F. Delicato, et al. Application-Driven Node Management in Multihop Wireless Sensor Networks. in: Proceedings of the 4th IEEE International Conference on Networks (ICN 2005), Reunion Island, April 2005.
14. W3C Working Draft. Web services description language (WSDL) Version 2.0 Part 1: Core Language (March 27, 2006); http://www.w3.org/TR/wsdl12/.
15. W3C Note (June 24, 1999) "WAP Binary XML Content Format"; http://www.w3.org/TR/wbxml/.
16. W3C Recommendation. OWL Web Ontology Language (February 10, 2004); http://www.w3.org/TR/owl-guide/.
17. Apache Axis (April 5, 2006); http://ws.apache.org/axis/.
18. Sensoria WINS 3.0 Spec. (April 5, 2006); http://www.sensoria.com/products-wins30.htm.
19. J2ME Wireless Toolkit (April 5, 2006); http://java.sun.com/products/j2mewtoolkit/.
20. KXML Project (April 5, 2006); http://kxml.objectweb.org/.
21. C. Intanagonwiwat, R. Govindan, D. Estrin, Directed diffusion: a scalable and robust communication paradigm for sensor networks. in: Proceedings of the ACM/IEEE MobiCom 2000, USA, Aug. 2000.