A Middleware Framework for the Internet of Things

Bruno Valente
Department of Informatics
LaSIGE & Faculty of Sciences, University of Lisbon
Portugal
bvalente@lasige.di.fc.ul.pt

Francisco Martins
Department of Informatics
LaSIGE & Faculty of Sciences, University of Lisbon
Portugal
fmartins@di.fc.ul.pt

***Abstract*—After the traditional Internet (with program-to-human communication), and after the Internet of Services (with program-to-program communication), the Internet of Things is a new paradigm of communication aiming at integrating the state of everyday things into the digital world. But things are everywhere, have different colours, come in different flavours, so, building reliable applications that depend on such things imposes great challenges and demand for new approaches to integrate heterogeneous devices smoothly. These new methods should use public and resilient networks, like the Internet, to secure and facilitate the access to things. This paper addresses such constraints and proposes a middleware framework to interact with the devices and their data, all this supported by the use of Web services. It is our goal to design and implement a generic framework where services represent functionalities of sensor networks and provide a dynamic way for high-level applications to interact and program such devices with heterogeneous hardware. By using Web services we benefit from the interoperable technology, cross platform, independency of programming languages, and available on the Web. Those attributes are ideal to combine heterogeneous systems like sensor networks. It is also our goal to create an event-driven system, where the user can subscribe the available services and receive notifications as network data is being processed.**

***Keywords*-middleware, Web services, sensor networks, internet of things, service-oriented architecture.**

I. Introduction

The Internet of Things (IoT) is a novel paradigm that aims at bridging the gap between the physical world and its representation within the digital world. The idea is to integrate the state of the "things" that form the world into software applications, making them benefit from world's context information.

There are various forms of capturing the state of things, ranging from simple bar codes identifying objects, to more sophisticated technologies involving radio-frequency identification (RFID), near field communication (NFC), or even more complex devices, such as sensor nodes, that come equipped with an internal memory, context awareness devices (e.g., GPS), and, specially, with computation capabilities [1]. These last mentioned devices are of special interest, since they allow things, named *smart objects*, to perform local computations and interact and collaborate among them.

Sensor networks [2] are a hot research topic in academia and an expansion business area in industry, with applications in many fields. These networks are composed of a set of nodes with the capability of sensing physical phenomena (e.g., luminosity, temperature, or humidity). Wireless Sensor Networks (WSNs) are a specialization of sensors networks, where communication among devices occurs via radio-frequency and nodes (usually) rely on a battery power supply. Those characteristics allow devices to operate remotely, but complicate the access and the process of sensed data.

Building high-level applications that exploit information from smart objects are of valuable interest, but they do not come without a cost. In fact, interacting with devices that run on top of different operating systems (e.g., TinyOS [3], Contiki [4]) or virtual machines (e.g., Squawk [5]), that use distinct programming languages (e.g., nesC [6]), and that use different communication protocols, makes it very difficult, and undesirable, to handle the complexity from the high-level application.

Our work focus on abstracting the interaction among applications and smart objects (e.g., nodes connected via a WSN), by both hiding the communication (and other hardware idiosyncrasies) and the programming capabilities of such objects. It is our intent to accommodate these differences in a middleware layer such that, from an application perspective, all smart objects are reprogrammable and present a common interface.

The programmability of smart objects depends, more often than not, on manufactures and whether they provide hardware specifications. As far as we are aware, there are not so many platforms that allows devices to be reprogrammed remotely, while running. In Callas [7] we presented a framework that supports reprogramming of nodes in a WSN. Here, we lift this idea to the middleware level by equipping it with reprogramming capabilities irrespectively of these capabilities being supported by the underlying infrastructure.

From the high-level application point of view deploying code into (a network of) smart objects is performed regardless of the ability of these devices to be programmed. Based on the configuration of smart objects, the middleware either installs the code in the devices or in itself and behaves as if the code was effectively deployed into the devices. For this approach we put together a Data-Flow engine that runs client deployed modules; these work upon data received from smart objects and are organised in dataflow chains acting as filters to process incoming data according to the client's requirements.

Another problematic area is the interoperability among observation centres and their clients. We follow the Sensor Web Enablement standard (SWE) [8] from the Open Geospatial Consortium. This standard defines a set of interoperability interfaces and metadata encodings that enable real-time integration of heterogeneous sensor webs. Our implementation respects two SWE standards: *Observations and Measurements* (O&M) [9][10], and *Sensor Observations Service* (SOS) [11].

To summarise, our main contributions are:

- the abstraction of code deployment, communications, and hardware of (networks of) smart objects, allowing for client applications to program heterogeneous devices either physically, by installing code into the devices, or logically, by creating a Data-Flow network of filters to process the data received from devices.
- the management of observations received from smart objects and its broadcast to client applications via Web services respecting the SWE specification;

The remainder of this paper is structured as follows: Section II presents state-of-the-art related work. Section III presents our approach to fulfil the aforementioned goals. In Section IV we detail the internals of out middleware, and, finally, the last section draws our conclusions and provides an overview of the intended future work.

II. Related work

Middleware frameworks are widely used to aggregate and manipulate sensor networks. Rakhi Motwani *et al.* [12] present a Service-Oriented Architecture (SOA) to coordinate observation centres and to share their information via Web services. The authors' motivation is that many observation centres are geographically dispersed and isolated, and it is difficult to share information among them. We adhere to this view and our middleware proposal, MufFIN (Middleware For the Internet of thiNgs), provides features to manage the data received from network devices and allow information to be registered and accessed via SWE standards.

Another open source implementation of SWE is the 52North Sensor Web community application [13]. Since the project seems to have achieved a mature state, we tried to build Muffin on top of it. However, it has revealed to be quite difficult to integrate a stand-alone application with our framework features. Instead, we used part of their specifications (e.g., database ER model) as the base to our SWE implementation.

The communication between heterogeneous services is possibly by using a known protocol on top of a common format, like XML. However, this kind of technologies needs more computational resources to marshal and unmarshal the contained information. Choon-Sung Nam *et al.* [14] propose an Event-Driven Architecture middleware equipped with a publish-subscribe paradigm, allowing client applications to subscribe topics of interest and to receive notifications when the server publishes information on such topics. This type of communication reduces the amount of messages exchanged among clients and servers, as opposed to pulling techniques used within synchronous scenarios.

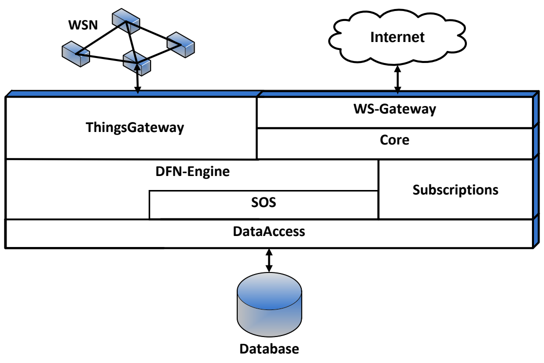

Figure 1. The bundles layer diagram illustrating the communication between bundles from the gateways to the database layer.

João Santos also implemented a middleware framework [15] that combines SOA and Event-Driven Architectures to achieve interoperability among WSNs and client applications via Web services. The framework supports the creation of pipelines (fixed dataflow chains), although it does not provide for dynamic composition of filters in dataflow chains, and its current implementation does not support the SWE specification.

Catello Di Martino *et al.* [16] present an adaptive and configurable architecture for accessing sensor networks based on their specifications. These specifications result in filters and focus of information, where clients connect to, in a fast way, and gather or receive notifications with the needed data.

Our middleware solution, like most of the aforementioned works, makes use of Web services, supports synchronous and asynchronous requests, complies with the SWE standards, and abstracts the sensor's hardware. New to MufFIN is the ability to transparently reprogram smart objects, being this our main scientific contribution.

III. Our Approach

This section presents our middleware design and the decisions made to address the requirements identified previously.

A. Architecture

We choose to design our middleware framework based on a Service-Oriented Architecture composed of a group of seven loosely coupled bundles. Figure 1 shows the framework layer diagram depicting the communication dependencies between components.

On top, we provide two communication gateways: the *thingsGateway* for communicating with smart objects (e.g., network sensors) and the *WS-Gateway* for communicating with high-level applications via Web services. The middleware supports both synchronous and asynchronous communications with clients. The *Core* bundle provides the Web interface implementation to invoke framework operations. *DFN-Engine* bundle manages client deployed modules as well as the dependency chains between them. It instantiates the client modules and creates the publish-subscribe connections for their communication. The details of the deploying operation is presented in Section III-B. As for the *Subscriptions* bundle it receives clients subscriptions, processes them, and saves the subscriber's information. This bundle publishes notifications for clients to the Web. Section III-C details the bundle's internals. The processing of XML documents for the Sensor Observation Service standard is delegated to the *SOS* bundle. All operations of this bundle must follow the OGC specifications. At the basis of our middleware lies the *DataAccess* bundle that provides a set of façades for other bundles to access the database layer. Database tables related to smart object observations are based on the O&M specification.

The following section describes how our framework handles devices' programming and data management.

B. Network Programming and Data Management

For network programming we allow clients to upload modules (filters) and to instantiate them. If the target smart object (network) allows for code installation, the middleware deploys the received module to it. Otherwise, the code is instantiated locally and runs directly at the middleware layer. For that, the client must also specify the module's dependencies. This specification is given in an XML document, like the example shown in Listing 1. Notice that the document also specifies whether the module will be made available as a Web service; the client may decide not to expose the module to the Web because it may be used to compute intermediate results. To discover module's information the client may invoke the Web service *getModulesInfo()*.

Listing 1. An example of XML file to instantiate a new modules

```
<deploy>
 <instance serviceId="TemperatureDiff" service="true">
  <dependency serviceId="LisbonTemperature"/>
  <dependency serviceId="PortoTemperature"/>
 </instance>
</deploy>
```

The created modules and the triggered events build a dependency chain as a publish-subscribe network, like it is shown in Figure 2. This chain results in a Data-Flow Network (DFN) where the raw data, coming from the smart objects (e.g., WSN nodes), is further processed to produce the information subscribed by the high-level applications.

Figure 2 illustrates the interaction with two networks of things, accessed through gateways *LisbonSink* and *PortoSink*. This modules receive raw observations directly from the networks. Whenever new data is available this modules

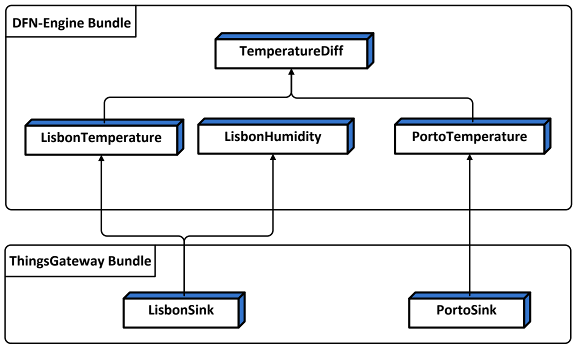

Figure 2. The modules dependencies chain.

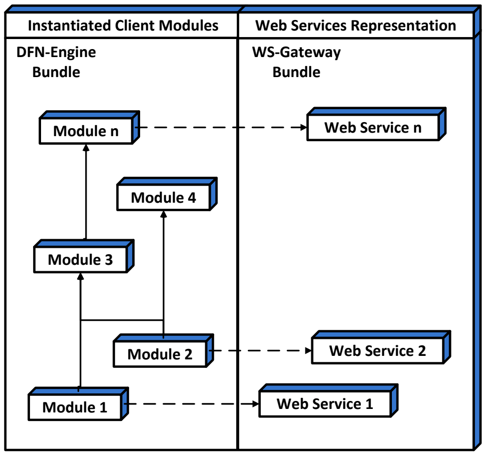

Figure 3. Conceptual view of the relation between module instances and their representation on the Web.

will store it and notify their subscribers, which will process the supplied data, store it, and further notify theirs subscribers (and so on). In case a module is available to the Web, its remote subscribers will be notified as well. For instance, the *LisbonSink* notifies modules *LisbonTemperature* and *LisbonHumidity*. In its turn, *LisbonTemperature* will inform *TemperatureDiff* that, dependent on a value received from the *PortoTemperature*, will compute its difference and make it available for its subscribers. If no value is available from the *PortoTemperature* module, *TemperatureDiff* will suspend until a value is obtained.

Each module stores its processed information, allowing for backdated queries.

C. Subscriptions

After instantiation, a module becomes available for subscription via its corresponding Web service. The *Subscriptions* bundle maintains information about subscribers and which services they subscribe. When the DFN processes information, it notifies the respective module and an event is sent to the subscriber.

Figure 3 represents the mapping between modules and

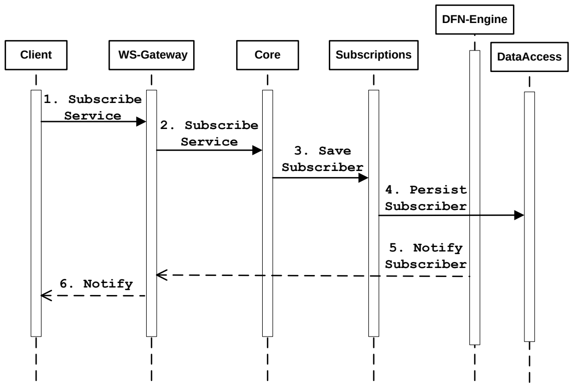

Figure 4. Sequence diagram with the invocations between modules from service subscription to client's notification.

Web services. In the present case, only modules 1, 2, and n are available for subscription. Modules 3 and 4 just compute intermediate information that is further refined and made available. *Module 4* exemplifies a kind of extension point that can be refined later.

The framework provides two methods of subscribing services:

- *Direct subscription:* by specifying the service to be subscribed, given its identification.
- *Discovery subscription:* by querying specific service characteristics. This allows for a client to discover (and subscribe) services based uniquely on the partial matching of the client's topics of interest with the module's characteristics. Our approach is to classify modules in ontologies (upon module's definition) and allows for querying these ontologies for finding relevant services.

To enable notifications it is necessary a client Web-service endpoint where the events will be delivered. The endpoint is correlated with the client at subscription time. Figure 4 shows the sequence diagram corresponding to the actions from a client subscription until its notification.

D. Framework Instances Integration

This framework was designed with interoperability in mind. Not only to integrate with different smart objects, but also to be integrated with other frameworks. This is particularly useful when a client wants to access to various observation centres. He may do it in two ways: (1) access each observation centre per se; or (2) if possible, some observation centres acts as a subscriber for the others and the client accesses only one centre that aggregates all the data. Notice that (ideally) all communications between middleware instances must be invisible to the client. The integration with other frameworks that are not compliant with SOS standard have to be tackled, not surprisingly, case by case, but it amounts to develop specific things gateway adapters for each case.

IV. Implementation

This section presents our decisions about MufFIN implementation and gives an overview of the two SWE standards we implement.

MufFIN is implemented in the Java programming language and runs on top of Fuse Enterprise Service Bus [17], based on Apache ServiceMix [18]. Fuse ESB implements the OSGi functionalities, allowing the complete and dynamic decoupling of the system components. The ESB is an architecture that facilitates the integration between services [12], [19], [20] and allows the creation of dynamic flows between system components.

The middleware introduced in this paper implements two types of communication with clients:

- *Synchronous:* allows clients to filter and receive observations already stored in the middleware. This communication respects the OGC standard, and returns the observations that follow the received constraints.
- *Asynchronous:* allows the middleware services subscription to receive observations that will occur in the future. When some action related with the service happens, it triggers an event that sends to the client the observations in OGC standard format.

Both communication types respects the following SWE standards that we present a brief overview.

Observations and Measurements (O&M)

Standard models and XML schemas for encoding observations and measurements from a sensor. An observation is defined as an act of measuring a property or phenomenon, with the goal of producing an estimate of its value.

The observation has the following mandatory fields:

- *Sampling Time:* time when the measurement was made.
- *Procedure:* process used to generate a result. Could be a sensor observation, an algorithm, a computation, or a complex processing chain.
- *Phenomenon:* defines the environmental characteristic to be read, and its unit.
- *Feature of Interest:* the observation target; the real-world object on which the observation is made.

O&M aggregates observations that have common characteristics; in particular, observations that have a similar *Feature of Interest* and observe the same *Phenomenons* should be related by an *offering*. *Offerings* define what is provided by the system and it is the base property for all observation requests.

Sensor Observations Service (SOS)

Web services interface for requesting, filtering, and retrieving observations and sensor system information. The

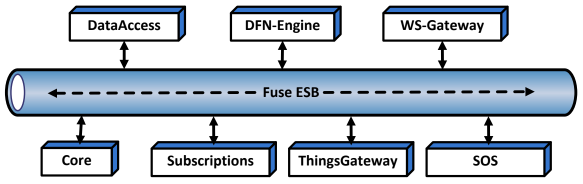

Figure 5. Main bundles that compose MufFIN integrated on Fuse ESB.

goal of SOS is to provide access to observations from sensors and sensor systems in a standard way that is consistent for all sensor systems including remote, in situ, fixed, and mobile sensors. The filtering arguments allow the client to specify the time, the location, the observed phenomenon, and the feature of interest of each sensor observation. All these arguments are dependent on an offering that is going to be used as the primary search criteria.

A. The Middleware Bundles

Figure 5 shows the middleware overview with the main bundles deployed on the Fuse ESB. The middleware is structured in decoupled bundles to facilitate code maintenance. Below, we present the bundles implementation details:

- *ThingsGateway:* allows the dynamic loading of adapters that bridge the communication between smart objects and our framework. These adapters are the root nodes of the Data Flow Network tree and are responsible for deploying new code into devices. Our framework is ready to send code to devices that run the Callas virtual machine, and can be customized to other languages with the deploying of new network adapters. This is possible because the framework receives byte-code, which facilitates the process since it is difficult to create a framework built-in with a large range of source language adapters.
- *WS-Gateway:* responsible for presenting the framework features as Web services. We follow the Web Service Notification protocol (WS-N) [21] to notify remote subscribers. When a high-level application subscribes a service, it must send an endpoint reference to where the notifications can be published. When a notification is triggered, our middleware dynamically connects to the client endpoint and sends the subscribed data.
- *DFN-Engine:* we use the *ActiveMQ*—a Fuse ESB built-in broker that implements the Java Message Service and provides reliable communication—to implement the following communications: *Queues*, to implement one-to-one communications among bundles; and *Topics*, to implement one-to-many communication among dynamically deployed DFN filters.
  The deployed services follow the command pattern [22], extending a class *Service* and overriding the method *doAction()*. This method is invoked whenever a notification arrives from a module, on which the self depends. The class Service also offers methods to get and set the observation properties specified in the O&M standard. After processing an incoming event a module publishes data via the *send()* method.
- *SOS:* responsible for the processing of the SOS standard XML documents. We implemented the XML parser on top of Apache XML Beans [23].
- *DataAccess:* provides the interfaces to access the database layer. All system data (e.g., observations, subscribers data) are stored in a MySQL database where the access is performed using the Java Persistence API (JPA) to decouple the database implementation.

V. Conclusion and Future Work

In this paper we proposed MufFIN—a generic middleware framework—that allows for managing and programming Internet of Things smart objects.

The IoT aims at bridging the gap between physical and digital worlds, by integrating world's context information, described by the state of the "things", into software applications, making them context aware. An important topic in this area is how to manage the heterogeneity among smart objects that equip these networks.

To improve the interoperability among observation centres, our framework conforms with Web service schemas that follow two Sensor Web Enablement standards from the Open Geospatial Consortium, namely the Observations and Measurements and the Sensor Observations Service. This allow for high-level applications and observation centres to get smart object data via the Internet using Web services.

The main goals of our framework are to manage the data received from WSNs and to create a generic framework to program their devices in two transparent methods: by installing code directly into devices, and by creating a Data Flow Network in the framework. The DFN is created with modules received from the clients and installed in the framework. This provides, to high-level applications, the perspective that the code was deployed into the sensors, even though the code runs at the middleware side. The DFN filters the data received from the WSN and returns the data that clients are expecting.

In the near future, we plan to pursue four major areas: (1) improve the discovery subscription of services, with the use of ontologies to bind directly the services with their *Feature of Interest* and the network device locations; (2) software testing and validation, in particular, load testing; (3) support for SensorML [24], a SWE standard to specify sensor devices and their characteristics; (4) field-testing, MufFIN is planned for being deployed in a real use case scenario, where all the presented features will be tested. The scenario involves collecting environmental values (temperature and moisture) from farms in the Azores archipelago, in order to establish the conditions wherewith a fungus that causes light skin hypersensitive in cattle may appear. The project has both financial and public health impact, since an accurate assessment of the fungus appearance symptoms will reduce causalities and prevent overmedication among animals.

Acknowledgment

The authors are partially supported by projects MACAW and PATI of the Fundação para a Ciência e Tecnologia (contracts PTDC/EIA-EIA/115730/2009 and PTDC/EIA-EIA/103751/2008).

References

[1] L. Atzori, A. Iera, and G. Morabito, "The internet of things: A survey," *Computer Networks*, vol. 54, no. 15, pp. 2787–2805, 2010.

[2] I. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, "A survey on sensor networks," *Communications Magazine, IEEE*, vol. 40, no. 8, pp. 102–114, 2002.

[3] "The TinyOS Documentation Project," Page visited on May 30th, 2011. [Online]. Available: http://www.tinyos.net/

[4] A. Dunkels, B. Grönvall, and T. Voigt, "Contiki - a Lightweight and Flexible Operating System for Tiny Networked Sensors," in *Proceedings of the First IEEE Workshop on Embedded Networked Sensors (Emnets-I)*, 2004.

[5] D. Simon, C. Cifuentes, D. Cleal, J. Daniels, and D. White, "Java on the Bare Metal of Wireless Sensor Devices – The Squawk Java Virtual Machine," in *Proceedings of the 2006 ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments (VEE)*. ACM Press, 2006.

[6] D. Gay, P. Levis, R. von Behren, M. Welsh, E. Brewer, and D. Culler, "The nesC Language: A Holistic Approach to Network Embedded Systems," in *ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*. ACM Press, 2003, pp. 1–11.

[7] F. Martins, L. Lopes, and J. Barros, "Towards the safe programming of wireless sensor networks," in *Proceedings of Programming Language Approaches to Concurrency and Communication-cEntric Software (PLACES)*, ser. EPTCS, vol. 17, 2010, pp. 49–62.

[8] M. Botts, G. Percivall, C. Reed, and J. Davidson, "OGC Sensor Web Enablement: Overview And High-Level Architecture," 2007, Page visited on October 15th, 2010. [Online]. Available: http://portal.opengeospatial.org/files/?artifact_id=25562

[9] S. Cox, "Observations and measurements part 1 - observation schema," 2007, Page visited on March 20th, 2011. [Online]. Available: http://portal.opengeospatial.org/files/?artifact_id=22466

[10] ——, "Observations and measurements part 2 - sampling features," 2007, Page visited on March 20th, 2011. [Online]. Available: http://portal.opengeospatial.org/files/?artifact_id=22467

[11] M. P. Arthur Na, "Sensor observation service," 2007, Page visited on March 20th, 2011. [Online]. Available: http://portal.opengeospatial.org/files/?artifact_id=26667

[12] R. Motwani, M. Motwani, F. Harris, and S. Dascalu, "Towards a scalable and interoperable global environmental sensor network using service-oriented architecture," in *Proceedings of the Sixth International Conference on Intelligent Sensors, Sensor Networks, and Information Processing (ISSNIP)*, 2010, pp. 151–156.

[13] A. Broering, T. Foerster, S. Jirka, and C. Priess, "Sensor Bus: An Intermediary Layer for Linking Geosensor Networks and the Sensor Web," in *Proceedings of the 1st International Conference on Computing for Geospatial Research and Applications (COM. Geo)*, 2010.

[14] C.-S. Nam, H.-J. Jeong, and D.-R. Shin, "Design of wireless sensor networks middleware using the publish/subscribe paradigm," in *Proceedings of the IEEE International Conference on Service Operations and Logistics, and Informatics (SOLI)*, vol. 1, 2008, pp. 559–563.

[15] J. Santos, "Um *middleware* para acesso e gestão de redes de sensores em ambientes Web," Master's thesis, Universidade Nova de Lisboa, Faculdade de Ciências e Tecnologia, 2009.

[16] C. D. Martino, G. D'Avino, and A. Testa, "icaas: An interoperable and configurable architecture for accessing sensor networks," *IJARAS*, vol. 1, no. 2, pp. 30–45, 2010.

[17] FuseSource Open Source Community, "Fuse ESB / Apache servicemix 4.3," Page visited on March 24th, 2011. [Online]. Available: http://fusesource.com/products/enterprise-servicemix/

[18] Apache Software Foundation, "Apache servicemix 4.3," Page visited on March 24th, 2011. [Online]. Available: http://servicemix.apache.org/

[19] H. J. La, J. S. Bae, S. H. Chang, and S. D. Kim, "Practical methods for adapting services using enterprise service bus," in *Proceedings of the 7th international conference on Web engineering (ICWE)*. Springer, 2007, pp. 53–58.

[20] X. Tang, S. Sun, X. Yuan, and D. Chen, "Automated web service composition system on enterprise service bus," in *Proceedings of the 3rd IEEE International Conference on Secure Software Integration and Reliability Improvement (SSIRI)*, 2009, pp. 9–13.

[21] Y. Huang and D. Gannon, "A comparative study of web services-based event notification specifications," in *Proceedings of the International Conference on Parallel Processing (ICPP)*, 2006, pp. 8–14.

[22] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns*. Addison-Wesley, 1995.

[23] Apache Software Foundation, "XML Beans project," Page visited on May 30th, 2011. [Online]. Available: http://xmlbeans.apache.org/

[24] M. Botts and A. Robin, "Sensor model language - implementation specification," 2007, Page visited on March 20th, 2011. [Online]. Available: http://portal.opengeospatial.org/files/?artifact_id=21273