An Architecture Description Language for Dynamic Sensor-Based Applications

Humberto Cervantes
Universidad Autonoma Metropolitana-Iztapalapa (UAM-I)
Iztapalapa. D.F., Mexico
hcm@xanum.uam.mx

Didier Donsez, Lionel Touseau
University of Grenoble-1
LIG Laboratory, ADELE team
Grenoble, France
Didier.Donsez@imag.fr, Lionel.Touseau@imag.fr

***Abstract*—This paper presents an approach to describe dynamic sensor-based applications using a declarative language called WADL. Dynamic sensor-based applications are characterized by the fact that measurement producers (sensors) and consumers are introduced or removed from an execution environment at run-time. Supporting this degree of dynamism is usually done programmatically, and the WADL intends to simplify this task and to provide developers with an explicit view of the system architecture, while supporting its dynamic evolution. The paper describes the WADL, its implementation on top of the OSGi WireAdmin Service, and some experimentation results.**

Keywords: architectural description language, dynamic sensor-based applications, OSGi

I. Introduction

The next wave of e-business will probably rely on the "Internet of Things" where data generated by many diverse devices will be collected by using a variety of sensors [1]. Sensor-based applications (SBAs) seek to acquire, collect, filter, aggregate, analyze and react to measurements gathered through a network of physical sensors that are spread in the physical world. This information should be integrated into different applications to support activities such as automation control (SCADA) or decision support (data analysis and monitoring). New business opportunities and models (pay-per-use, pay-as-you drive, etc) can be created from the online and offline exploitation of the information on the physical world. Examples of measurements that are obtained through sensors include RFID identifiers, GPS vehicle positions, room temperatures, smoke density in a lobby, blood glucose levels, etc.

Sensor-based applications can be nicely designed by using mainly the Producer-Consumer communication pattern [2] where sensors produce measurements and, data processing modules consume produced data. Connecting producers and consumers is a frequent activity in SBA. This pattern differs from the publish-subscribe communication pattern since it combines push and pull interactions. The producers push the data to the consumers when a new data is acquired, however consumers can force the production of a new value or retrieve the previous value. Moreover, various levels of quality of service can characterize a connection between a producer and a consumer. For instance, the dataflow control can limit the push until acquired data are significantly different.

Dynamic sensor-based applications are characterized by the fact that measurement producers and measurement consumers are introduced or removed from the application at run-time. For instance, a newly-installed smoke detector should be taken into account by a fire monitoring system without the need to restart it. Although there exist different middleware platforms and component models that can be used in the construction of sensor-based applications, they do not usually support the dynamic aspect in an explicit way, as dynamism usually has to be supported programmatically. Managing dynamism, which can be considered a non-functional requirement, through code is generally a complex task. Furthermore, this approach results in a mix of functional and non-functional code and it makes the architecture of the application difficult to understand and to modify as connection logic is buried inside the code.

This paper proposes an approach to describe dynamic sensor-based applications through the use of a declarative language called Wired Application Description Language (WADL). This language describes collections of connectors that bind measurement producers and measurement consumers. To support dynamism, a WADL descriptor is capable of expressing variable sets of connectors that can be created and destroyed dynamically. These descriptors are further used by an interpreter which is responsible for managing the connectors between measurement producers and consumers as they are introduced or removed dynamically from the execution environment.

The remainder of the paper is structured as follows. Section 2 introduces dynamic sensor-based applications, section 3 describes the WADL characteristics, section 4 presents an implementation of the execution environment based on the OSGi framework and some experimentation results. Section 5 discusses related work and finally section 6 exposes future work and concludes this paper.

II. Dynamic sensor-based applications

This section describes the concepts and issues associated to the introduction of dynamism in sensor-based applications.

A. *Dynamism in sensor based applications*

Dynamic sensor-based applications are characterized by the fact that measurement producers and measurement consumers need to be introduced or removed from the application at run-time. Dynamism is highly desirable in a majority of sensor-based applications. Certain environments, such as medical monitoring systems, impose this type of constraint, as it is not possible to turn off the monitoring application in order to modify the sensor network topology or to add or remove data processing modules. In large scale sensor networks, such as the ones present in residential or office building automation [3,4], the addition or the replacement of sensors such as thermometers or smoke detectors by human operators must be done automatically without stopping the building's monitoring systems. In a similar way, dynamic changes in the quality of service offered by sensors could impact the topology of the application. Moreover, the configuration of a complex and dynamic topology is a real burden for human administrators. Automating this task can help reduce costs. Fig. 1 presents a portfolio of sensor-based applications commonly used in building automation. For instance, adding a presence detector or replacing a faulty one requires both the lighting control application topology and the burglar central application to be modified since they share concurrently those sensors. These operations are error-prone since the maintenance operator may not be the administrator for both applications.

B. *Modeling sensor-based applications using a service-oriented approach*

Traditional software architectures are usually modeled statically through the description of sets of components and connectors that bind the components using Architecture Description Languages (ADLs) [5]. Dynamic software architectures introduce a particular challenge, because they must support changes at the architectural level during execution. These changes may include the creation or removal of component instances, and connections between these instances at run-time.

Furthermore some applications with dynamic architectures have additional requirements with respect to the introduction or removal of components at runtime. For instance, components may not be available at the time the original application is composed. Supporting these requirements can be achieved by incorporating a discovery mechanism in the environment. In service-oriented architectures (SOA) [6,7], this discovery mechanism is usually some type of registry where components publish the services they provide. Clients can later query the registry or receive notifications about services that are published or removed from the registry at runtime. Once a client discovers a particular service, it can bind directly to the service provider and, in this way, the application architecture evolves continuously as new components are incorporated or removed from the execution environment. Moreover, with an SOA approach every component can be substituted by another one as long as they comply with the same contract (typically defined trough an interface) is provided. If applied to sensor-based applications this substitution mechanism strengthens the availability and robustness of components representing physical measurement producers.

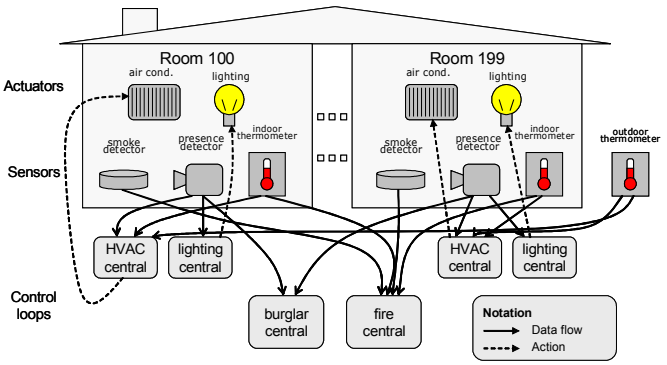

Figure 1. Sensor-based applications in residential or office building automation

The OSGi specification [8] proposes facilities to manage connections between producers and consumers through its WireAdmin service using an SOA approach. Producers and consumers are modeled as uniquely identified OSGi services (i.e. they are published in a service registry along with a set of properties). They are delivered in deployment units called bundles. At runtime the connectors, namely wires, are managed by the WireAdmin Service. This service allows wires to be created, deleted, retrieved and updated programmatically. Once connected, producers can either push data into consumers or provide data when they are polled through the wires. Wires are persistent entities that bind specific producers and consumers through unique identifiers.

C. *WireAdmin service limitations*

Although the WireAdmin mechanism supports the construction of dynamic sensor-based applications, it has several limitations. The first one is that wires are inextricably tied to specific consumers and producers via persistent and unique identifiers. The second one is that modifications of the topology must be realized programmatically. As a result, there is no explicit representation of the architecture, for it is hidden inside the code responsible for creating or destroying the wires. Furthermore, the life-cycle (i.e. activation and passivation) of a SBA depends generally on the presence or on the absence of mandatory producers or consumers. For instance, a HVAC central (see Fig. 1) may be stopped if no more thermometers are available. The code that manages the application life-cycle is also mixed with the code creating and destroying the wires. As a consequence, evolution and maintenance of such wired applications is complex and error-prone.

III. WADL CHARACTERISTICS

The declarative description language for dynamic-sensor based applications (WADL) is based on three main requirements. First, it must allow producers and consumers to be introduced and removed at run-time. Second, it must support the binding of producers and consumers which may not have been available at the time the composition was described. Third, the application must be activated or passivated depending on the presence or absence of mandatory producers or consumers. This section describes the main characteristics of the language and presents an example of a fire detection application.

Identify applicable sponsor/s here. *(sponsors)*

A. Language meta-model

The main WADL language concepts and their relationships are represented in the meta-model in Fig. 2. These concepts include:

WireApp: A *wireapp* represents a wired application which is composed as dynamic sets of producers, consumers and their connections called wires. A *wireapp* defines the overall life-cycle according to the dynamic sets which are required to activate and deactivate the data-flow in the application. w*ireapp* life-cycle is discussed in more depth in the next section.

WireSet: As its name suggests, a *wireset* represents a dynamic set of wires that connect producers and consumers. To support flexibility in wire creation, *wiresets* are not defined in terms of specific producer and consumer identifiers but are rather characterized by two filters that constrain the selection of producers and consumers. These filters, which are based on the properties associated to the producers and consumers, allow producer and consumer selection to be narrowed or widened. A narrow selection can be achieved by filtering producers or consumers based on their unique persistent identifiers, whereas, a wide selection can be achieved by filtering them according to other properties such as the type of measurements that they produce or consume. In Fig. 4, both *wiresets* filters illustrate an intermediate selection where just the identity of the consumer matters. The mandatory attribute defines if the *wireset* is mandatory or optional for the *wireapp* life-cycle. Mandatory *wiresets* impose to have at least one producer-consumer connection to enable the *wireapp* activation. *Wiresets* also define a removal policy for the wires that are associated to them. The removal policy, which can take the values defined in the *RemovePolicy* enumeration, defines wire life-cycle policies. Filters and removal policies are discussed in more depth in the next section.

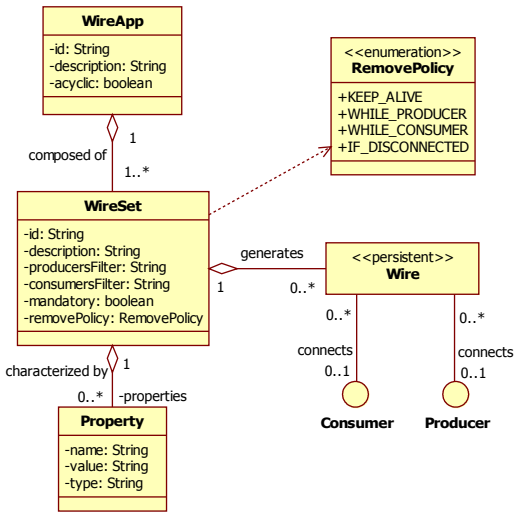

Figure 2. Wired Application Description Language meta-model

Property: Properties are specified QoS properties used by the wires and the Producers in order to control the dataflow and alleviate consumers load. Properties attached to *wiresets* are used to initialize the generated wires. A frequently-used property is a filter expression on produced data to push a new value only when the variation with the previous one is significant. For instance, the filter presented in the example of Fig. 4, forces the value to be refreshed at least every 2000 milliseconds.

B. Wired Application life-cycle

The overall activity of a SBA is usually constrained by the presence or the absence of some producers or consumers. This activity is mainly defined by the dataflow between producers and consumers in the application. Handling the application life-cycle (i.e. activation and passivation) consists in starting and stopping the dataflows. Since the WireAdmin specification does not define those operations on wires, the application activation consists in the creation of wires whereas passivation consists in the destruction of the previously created wires. In WADL, the *wireapp* cannot be activated until all mandatory *wiresets* do not match at least one producer with one consumer.

WADL proposes four different behaviour policies when a consumer or a producer is removed from the running application. The default policy, called IF_DISCONNECTED, destroys the wire if either the consumer or the producer are removed. The WHILE_PRODUCER and WHILE_CONSUMER policies result in the destruction of the wire only if the producer or the consumer are removed respectively. Those two policies prevent inefficient wire destructions when the producers or the consumers are used to disappearing temporally. Finally, the KEEP_ALIVE policy results in wires that are persistent once they are created and that must be removed programmatically. This policy is tied to the WireAdmin Service specification which requires the wire persistence.

Finally, the *wireapp* is passivated when the last wire of a mandatory *wireset* is removed. As a consequence, all the wires in the *wiresets* of the *wireapp* are removed, including those created with the KEEP_ALIVE policies.

C. Describing a fire central wired application

In WADL, applications are described declaratively in an XML descriptor where the *wireapp* element is at the root. As a consequence, WADL descriptors contain one *wireapp* which is itself composed of one or more *wiresets*. Inside *wiresets*, filters are described using an LDAP syntax.

Fig. 3 presents the components of a simple fire detection application similar to the fire central module included in Fig. 1. This module displays alert messages when abnormal temperatures (expressed in Kelvin) or smoke levels are detected in any room of the building. The topology of this SBA is described in the descriptor shown in Fig. 4. In this example, the *wireapp* is composed of two different *wiresets*. The first *wireset* ties a specific consumer (the fire central), filtered through its unique identifier, to any producers of temperature whose type can be either Measure (javax.measure.Measure) or Measurement (org.osgi.utile.measurement.Measurement). The second *wireset* ties any smoke sensor that produces a SmokeLevel to a specific consumer, the fire detection central.

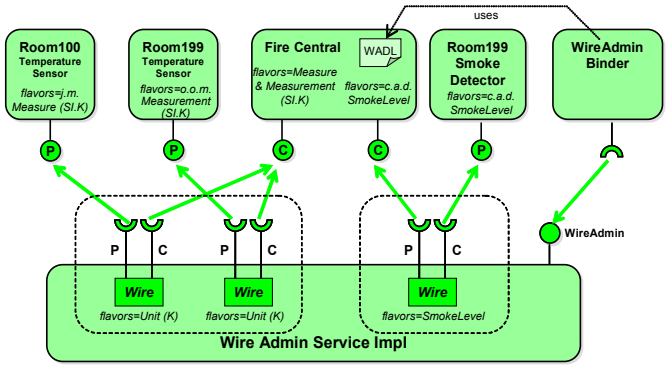

Figure 3. A fire detection wired application

The "smoke2central" *wireset* is the only one mandatory for the application activation. So the wires are effectively created when at least one smoke level producer can be connected to the fire central component. In this fire detection application, two different wire removal policies are used. The WHILE_CONSUMER policy will not destroy the wires until the fire central consumer becomes unavailable, even if smoke detectors components were to be removed.

IV. Execution environment and Validation

This section presents the WireAdminBinder, an engine that interprets the WADL descriptors and that manages sets of wires and their life-cycle. It also presents a validation of the WireAdminBinder built on top of the Felix OSGi implementation [9].

A. WireAdminBinder and application design

The WireAdminBinder is the engine that interprets WADL descriptors. It is implemented on top of the OSGi framework and delivered in a separate bundle. It relies on the WireAdmin Service to create persistent wires between consumers and producers according to the filters described in the *wiresets*. As producers and consumers are deployed or removed to/from the execution platform, the WireAdminBinder is notified and reacts by creating or removing wires according to the policies defined in the descriptor.

Two wired application designs are conceivable by the application architect. A first one where the *WireAdminBinder* acts as a global orchestrator of all its *wiresets.* Another design considers producers and consumers as autonomous components managing their own *wiresets*. However, this latter choice has some drawbacks. First, the *wiresets* managed by the independent components cannot be passivated according to the state of the other independent *wiresets*. Second, the lack of a global architecture orchestrator can introduce some issues such as the difficulty of preventing circular dependencies.

Moreover, most of SBA are designed as a sequence of stages processing measurement flows. The first stage is generally a set of sensors producing raw measurements and the last stage is a set of reporting tools consuming consolidated measurements. The intermediary stages can be components that consume measurements, process them and then produce measurements. When the produced measurements have the type (i.e. flavor in the WireAdmin terminology) of the consumed one, the architect has to take care of the *wiresets* definition in order to avoid cycles in the wire topology. The cycle prevention should be controlled at the *wireapp* level when the attribute acyclic is set to 'true'. By default, the wire creation is not controlled in order to let the architect design applications use feedback loop in the architecture.

B. Experimentation and validation

WADL and WireAdminBinder were experimented and validated in the context of the PISE project. This project was led by Schneider Electric, an electric-protection equipments manufacturer. The PISE project aimed to provide a component model for the development of dynamic sensor-based applications (SBAs). These applications are designed by domain analysts and experts by assembling and by configuring components selected from a domain-specific library.

This component model, called SensorBean [10], offers three message exchange patterns to the developer: request-response, publish-subscribe events and dataflows. The latter is implemented by producer-consumer interactions. The producer components represent electric sensors that acquire electric measurements such as power or voltage. The consumer components represent reporting tools, online dashboards and actuators such as circuit breakers.

The wire topology between components is described using the WADL formalism. Furthermore, these SBAs are dynamically deployed on industrial gateways installed inside factories networks. A gateway can simultaneously run several sensor-based applications which may share sensors.

```
<?xml version="1.0" encoding="UTF-8"?>
<wireapp id="building.FireCentral"
         description="A Fire central wired application"
         acyclic="true">
   <!-- a many-to-one wireset without wire properties -->
   <!-- connects temperature sensors to the fire central -->
   <!-- + keepAlive remove policy -->
   <wireset
     id="temperature2central"
     description="temperatures consumed by the fire central"
     producers-filter="(&(|(wireadmin.producer.flavors=
                     *org.osgi.util.measurement.Measurement)
                     (wireadmin.producer.flavors=
                     *javax.measure.Measure))(unit=SI.K))"
     consumers-filter="(service.pid=building.firecentral.temperature)"
     mandatory="true"
     removepolicy="KEEP_ALIVE"
   />
   <!-- current rooms smoke level to the fire central -->
   <!-- + whileConsumer remove policy -->
   <wireset
     id="smoke2central"
     description="smoke level producers consumed by the fire central"
     producers-filter="(wireadmin.producer.flavors=
                      *com.acme.data.SmokeLevel)"
     consumers-filter="(service.pid=building.firecentral.smoke)"
     removepolicy="whileConsumer"
     mandatory="true"
   />
     <property
       name="wireadmin.filter"
       value="(wirevalue.elapsed>=2000)"
       type="java.lang.String"
     />
   </wireset>
</wireapp>
```

Figure 4. Wireapp describing a fire central application

V. Related work

Sensor-based applications are a core element of the so-called "Internet of Things". Architects and developers of such applications require middleware support to tackle the complexity of sensor infrastructures. These infrastructures are composed of distributed nodes with various capabilities (sensors, gateways, intermediate servers, corporate servers, etc) on various protocols. These middlewares [11] can provide programming paradigms to query the sensors network as a fully distributed database [12], to publish events triggered on threshold, or to push periodically measurements such as OMG Data Distribution Service, IEEE/NIST 1451.x or OSGi WireAdmin. They can enforce Quality of Service requirements such as communication latency or throughput, and provide means to discover and manage the nodes. Most of them are designed to meet the challenges of wireless sensors, focusing on the energy-efficient computing. But unlike the WireAdminBinder none of them provide a convenient way to build the dynamic bindings that occur between nodes cooperating in an application at runtime.

Component models such as SOFA 2.0 [13] and O3MiSCID [14] provide dynamically reconfigurable dataflow connectors. Nevertheless, connections are set between identified components and the application life-cycle cannot be driven automatically by the presence of producers and consumers. ServiceBinder [15] propose to automate binding and life-cycle controls for the OSGi platform but it addresses only client-server interactions between services and does not fit for the SBA design.

Architectural Description Languages or ADLs are modeling notations that allow the architecture of a system to be described, mainly in terms of components, connectors and configurations. The majority of existing ADLs deal with static composition, although ADLs such as Darwin support a degree of dynamism [5]. The WADL is different from an ADL in the sense that it does not describe components but dynamic sets of components. However, in ADL terms, *wiresets* could be regarded as collections of connectors and *wireapps* as configurations.

VI. Conclusions

This paper has presented a description language to facilitate the construction of dynamic sensor-based applications built following the OSGi WireAdmin model. An interpreter for this language, called WireAdminBinder has also been implemented on top of the OSGi framework. Applications that are built using the WADL language support the introduction and removal of measurement producers and consumers through the dynamic creation of wires that connect these two entities. WADL has been successfully used in a research project led by an industrial partner. It must be noted that although the work presented here is implemented on top of the OSGi framework and the WireAdmin Service, its concepts can easily be ported to any dynamic service platform. One area that could be explored in the future is the use of the properties associated to the *wiresets* to describe more complex quality of service properties.

References

[1] International Telecommunication Union, "The Internet of Things", Executive Summary, ITU Internet Reports 2005, November 2005

[2] N. Nillson, "Connecting Producers and Consumers", position paper at OOPSLA Worshop on References Architectures and Patterns for Pervasive Computing, 27 October 2003, Anaheim, CA, USA http://jeckstein.com/oopsla/pervasive-computing.

[3] D. Marples, S. Moyer, "Home Networking and Appliances", in Diane Cook, Sajal Das, Smart Environments: Technologies, Protocols and Applications, Wiley, 2004

[4] D. Snoonian, "Smart Building", IEEE Spectrum, August 2003.

[5] N. Medvidovic, R.N. Taylor, "A Classification and Comparison Framework for Software Architecture Description Languages", IEEE Transactions on Software Engineering, Vol. 26, No. 1, (pp. 70-96), January 2000.

[6] G. Bieber, J. Carpenter, "Introduction to Service-Oriented Programming", OpenWings whitepaper, 2001, http://www.openwings.org/

[7] H. Cervantes and R. S. Hall: "Chapter I: Service Oriented Concepts and Technologies", in the book "Service-Oriented Software System Engineering: Challenges and Practices" (ISBN 1-59140-426-6) edited by Zoran Stojanovic and Ajantha Dahanayake, Idea Group Publishing, 2005.

[8] Open Services Gateway Alliance, "OSGi Service Platform Specification, Release 4", Available online at http://www.osgi.org

[9] Apache Felix : http://cwiki.apache.org/FELIX/index.html

[10] C. Marin, M. Desertot, "SensorBean: A Component Platform for Sensor-Based Services", proceedings of the International Worshop of Middleware for Pervasive and Ad-Hoc Compouting (MPAC), Grenoble, France, 28-29 November 2005

[11] W.B. Heinzelman, A.L. Murphy, H.S. Carvalho, M.A. Perillo, "Middleware to support sensor network applications", IEEE Network, Volume18, Number 1, Jan/Feb 2004, pp. 6- 14

[12] P. Bonnet, J. Gehrke, P. Seshadri, "Querying the physical world", IEEE Personal Communication, Volume 7, October 2000, pp pp. 10-15

[13] T. Bures, P. Hnetynka, F. Plasil, "SOFA 2.0: Balancing Advanced Features in a Hierarchical Component Model". SERA 2006: 40-48

[14] R. Emonet, D. Vaufreydaz, P. Reignier, J. Letessier, "O3MiSCID: an Object Oriented Opensource Middleware for Service Connection, Introspection and Discovery", 1st IEEE International Workshop on Services Integration in Pervasive Environments - June 2006

[15] H. Cervantes, R.S. Hall, "Automating Service Dependency Management in a Service-Oriented Component Model", 6th International Symposium on Component-Based Software Engineering (CBSE), Portland, OR, 2003