SENDA: AN ALTERNATIVE TO OSGI FOR LARGE SCALE DOMOTICS

FRANCISCO MOYA
University of Castilla-La Mancha, Escuela Superior de Informática, E-13071 Ciudad Real, Spain
E-mail: Francisco.Moya@uclm.es

JUAN CARLOS LÓPEZ
University of Castilla-La Mancha, Escuela Superior de Informática, E-13071 Ciudad Real, Spain
E-mail: JuanCarlos.Lopez@uclm.es

In this paper we present an overview of SENDA, a novel architecture for device networking based on very simple abstractions, but powerful enough to overcome limitations of previous approaches. SENDA defines a set of simple device interfaces, a hierarchical composition mechanism for both device objects and event propagating objects, a set of interfaces for managing and initializing the network, and a set of conventions for easier development of services.

1 Introduction

The vast variety of technologies available for in-home appliance networking introduced the problem of interoperability among devices using distinct protocols. A number of companies realized how important this problem was for the full deployment of in-home networks, and joined efforts to develop a specification of a framework for next generation in-home services, the *Open Systems Gateway Initiative (OSGi* [1]*)*.

OSGi builds upon Jini [2], the Java-based infrastructure specified by Sun Microsystems for developing object-oriented distributed systems. Therefore OSGi inherits most of the facilities and services of Jini. Besides, OSGi provides additional APIs and services to ease remote service management.

In order to integrate non-Jini capable devices, Sun Microsystems propose several schemes using clusters of Java objects [3] which act as proxies to the real devices. This is essentially the same concept of a *residential gateway* as described in [1], the most commercially successful architecture for easy integration of heterogeneous networks and devices in the current home market.

Both, Jini and OSGi provide the means to offer a unified view of the devices connected to a network but neither of them enforce policies or conventions regarding those interfaces. OSGi focuses mainly on remote management, configuration and dynamic updates of services, leaving the device interfaces unspecified.

The decentralized architecture advocated by Jini contrasts with the mainly centralized architecture implemented by OSGi gateways. From our point of view, residential gateways should only be used as a transition mechanism for devices or networks unable to cooperate with the rest.

From the practical point of view OSGi imposes some other rather strong constraints. For example, there is a strong dependency on the Java programming language. The choice of the Java language is usually justified [4] as a way to achieve better platform independence and better security. Binary platform independence is only an issue when the devices allow remote software upgrading, which may not be the case when the devices are very small. And even in those cases there are other emerging options available (such as Microsoft `.net`). It is surprising to find such a restriction in the programming language for a system intended to deploy any number of heterogeneous networks, protocols and devices transparently.

While Jini, and thus OSGi, was supposed to be independent of the transport protocol, the actual implementation seems to be intended for Internet-enabled devices [5]. This is a significant burden for small devices [6] and a poor option for real-time applications given that Internet protocols do not provide deterministic message delivery times.

Finally, another drawback related to the previous one is the need for a feature-rich Java virtual machine. Even the embedded Java virtual machine developed by Sun Microsystems (KVM) is unable to run OSGi software bundles due to missing features used by Jini [5]. A full-blown Java Virtual Machine is far too expensive to fit pervasive tiny appliances such as lamps or motion sensors. It is frequently argued that computing power, memory and price will not be long-term issues, but it depends on what kind of smart home is envisioned for the near future.

In our opinion, a smart home should interconnect as much sensors and actuators as possible, and that means lowering the cost to the very minimum. Besides that, next generation services will probably be operated on a nationwide or even world-wide basis. Therefore scalability is also an important issue.

In this paper we show an alternative architecture for next-generation in-

home services focused on minimalism, scalability and standardized service and device interfaces, called SENDA. SENDA stands for *Services and Networks for Domotic Applications*, and tries to solve most of current OSGi problems while retaining the ability to interoperate with other architectures, including OSGi.

2 The SENDA framework

SENDA provides a network infrastructure specially suited to the fields of domotics and in-home services (hierarchical, language neutral, platform independent, network independent, etc.). It is based on a single abstraction (and interface) similar to the *box* described in [7]. SENDA current prototype is built using CORBA and some OMG standard services, as we discuss in section 3.

The SENDA network infrastructure groups its elements in three levels of abstraction:

- The lowest level provides basic control of each appliance or device abstracting the peculiarities of the actual technology used. At this level SENDA provides generic interfaces for sensors and actuators.
- The next level provides device location according to different criteria (physical position, functionality, etc.). At this level SENDA also provides activation and deactivation of systems and services and event propagation from producers to consumers.
- The highest level of abstraction provides methods and tools for developing services using formal models of computation. From simple deterministic finite state machines, with optimum formal properties, to complex SpecCharts diagrams or discrete event models. Of course, SENDA does not enforce these methods and tools. Services may be developed using the traditional software design flow.

The next subsections will describe a functional decomposition of SENDA modules, the hierarchy support for module composition, and the mechanisms of event propagation and filtering.

2.1 SENDA modules

SENDA provides four kinds of modules: sensors, actuators, services and managers. They are briefly described below:

Sensors. They implement a minimal interface for notification of variations in physical magnitudes (events). The interface was designed to be

easily implemented in extremely simple devices, unable to implement even the TCP/IP stack (not to think in a full-blown Java Virtual Machine with RMI support as required by OSGi).

Some sensors, not necessarily all of them, may be *active sensors*. In this case they may be able to directly propagate events through an event consumer. Event propagation will be discussed in section 2.5.

Every sensor has an associated read-only value of the measured magnitude and a suitable type (binary, discrete, continuous, frame, etc.) for the vast majority of commercial sensors. SENDA also establish naming and functional conventions for future types of sensors in order to fit smoothly in the framework.

It is important to stress that the notion of a sensor as an event producer is not necessarily backed by an actual physical sensing device. The value of a given magnitude may be inferred from the values of other sensors, as we will describe later.

Actuators. They implement a simple interface complementary to the sensor interface allowing direct control over controllable devices. Minimalism is again the objective in order to be able to implement the interfaces in the devices themselves.

An actuator, just like a sensor, has an associated value of a given type. But this time it is a read-write value.

Actuators, like sensors, are also event producers. They may even participate in event propagation as described in section 2.5. This is the case for what we call *active actuators*.

An actuator does not require a physical device either. There may be virtual actuators able to react to any event in the system (even internal clock ticks) generating their own events.

Values associated to sensors and actuators are strongly typed. All sensor and actuator modules available in a system must implement one interface among a small set of core interfaces defined by SENDA (see table 1). A particular sensor may also refine those interfaces into additional more specific interfaces, but providing one of the core interfaces will ensure interoperability with modules unaware of the specifics of a given module. The set of core interfaces also allows hierarchical composition of them.

Services. SENDA is a service-oriented architecture. Services are just event consumers and/or producers. They act as relations among several event producers (sensors and/or actuators).

Usual services react to sensor stimuli (events), produce and propagate events to other services, and influence the environment through actuators.

Event filters constitute a special kind of services. They consume events

Module	type of *value*	container?	events?
Binary	*boolean*	no	no
Discrete	*integer*	no	no
Continuous	*float*	no	no
Frame	*byte sequence*	no	no
Object	*object ref*	no	no
BinaryActive	*boolean*	no	yes
DiscreteActive	*integer*	no	yes
ContinuousActive	*float*	no	yes
FrameActive	*byte sequence*	no	yes
ObjectActive	*object ref*	no	yes
BinaryComposite	*boolean*	yes	no
DiscreteComposite	*integer*	yes	no
ContinuousComposite	*float*	yes	no
FrameComposite	*byte sequence*	yes	no
ObjectComposite	*object ref*	yes	no
BinaryActiveComposite	*boolean*	yes	yes
DiscreteActiveComposite	*integer*	yes	yes
ContinuousActiveComposite	*float*	yes	yes
FrameActiveComposite	*byte sequence*	yes	yes
ObjectActiveComposite	*object ref*	yes	yes

Table 1. The current set of core SENDA actuators and sensors

from several sensors and generate, in a separate event channel, events derived from them. E. g. we may implement an intrusion detector filter from information gathered from many motion sensors, some cameras and a few perceptual computing algorithms. From this point of view the intrusion detector is perceived as a simple *active sensor*. Filters are intended to reduce the amount of information transmitted to upper levels of the hierarchy.

Managers. Some sensors, actuators and services are not able to initialize themselves. On the other side, some others will be preprogrammed in tiny microcontrollers. Those that require external initialization depend on a manager. The manager is also in charge of proxy activation if the physical device does not implement SENDA interfaces.

Besides, most services will probably be interested in publishing their object references or looking for other object references in an external service. This is also a manager responsibility, registering objects in a standard name service.

Finally, managers are also responsibles for monitoring the status of some sensors and actuators through *watchdog* processes. Whenever a failure is detected, a manager may apply preventive and corrective measures.

Some administrative procedures carried out by managers are also notified as events to the framework.

The manager interface always implements two methods (`start` and `stop`) for initialization and finalization of managed modules.

2.2 Composite modules

Every SENDA module may be composite. If this is the case they implement an object publishing interface. Each member of the composite must be published at construction time allowing a whole module hierarchy.

Hierarchical module composition is not only useful for increased scalability but just to increase the level of abstraction of a set of sensors, actuators, or managers. E.g. a composite actuator may be used to switch off all the lamps of a building with a single command.

2.3 Publishing object references

Object location attending to any criteria (geographical position, functionality, logical connection, etc.) is delegated to the same publishing interface used for composite objects. SENDA specifies the hierarchical organization and registration conventions.

2.4 Legacy systems

From the point of view of device integrators, adding a new appliance to a SENDA network implies the activation of a module (sensor or actuator). There are three options:

1. Traditional non-SENDA-aware devices. In this case a low cost device act as a proxy between SENDA and legacy networks. This is the model assumed in OSGi based installations.

   As a representative of this choice, we co-developed a gateway-style device (Domobox@) able to operate X10 devices, Lonworks devices and TCP/IP based Ethernet networks[a] (see bottom of figure 1). This kind of intermediary devices constitute a good solution to integrate legacy networks with a minimum of configuration.

[a] This device was developed in collaboration with Telefonica I+D.

Figure 1. Different devices and interfaces used in a SENDA network.

2. Another posibility consists in developing a minimal version of the communications core specially suited to the kind of messages used in SENDA. An example of this approach was developed on a TINI device (see upper left corner of figure 1), from Dallas Semiconductors. A TINI is a very small system based on a low cost microcontroller implementing an embedded Java Virtual Machine. The embedded JVM is so limited that RMI is not available and, therefore, OSGi is not supported. Even with such limited resources it was enough to implement a reduced CORBA ORB specially suited to SENDA needs.

3. Some devices may be inmediately available for other SENDA modules. This makes easy to develop smart devices without no need of an associated software module. This kind of devices are being developed using very low-cost eight-bit low-end microcontrollers. It is not possible to include the full communications core, but there is enough room to implement those messages of a single actuator or sensor interface (just a handful of

GIOP messages). In a way these devices may be considered a hardware implementation of a SENDA module.

2.5 Event notification

A SENDA event is any change in the value associated with a magnitude of the system. Most events correspond to physical magnitudes (temperature, motion, light intensity, smoke, time, etc.) although we also consider virtual magnitudes, whose variations produce administrative events (sensor and actuator states, module activation and deactivation, etc.).

Events are propagated through logical event channels, including enough information to identify the source of the event and the new value of the associated magnitude.

Each SENDA service or manager may provide any number of event channels to propagate events coming from sensors or actuators. Each channel propagates homogeneous events (same type) which define the type of sensors and actuators that may be connected to it.

Composite services automatically export their internal objects, including references of event channels. These references may be used by upper level services or connected to other channels in a hierarchical fashion. Therefore lower level services will tolerate failures in upper level channels.

SENDA does not require sensors or actuators to propagate events. Pasive sensors and actuators will be polled periodically by consumer services and any event will also be propagated through an event channel.

3 A SENDA Prototype

The current prototype of the SENDA framework is built on top of CORBA [8] using standard services as the basic building blocks.

SENDA Sensors and Actuators are implemented as template specializations of an extremely simple interface consisting of a single attribute called *value*. Figure 2 shows a subset of SENDA core interfaces for sensors and actuators.

As shown in figure 2, a composite module must inherit from the standard naming service interface *CosNaming::NamingContext*. As explicitly stated in the CORBA specification [8], it is possible to implement specialized versions of the standard services for special purpose applications. SENDA only requires the standard *list* and *resolve* methods to obtain references of modules contained in the composite module. SENDA does not even require the *bind* method to associate a symbolic name to a component because all composite

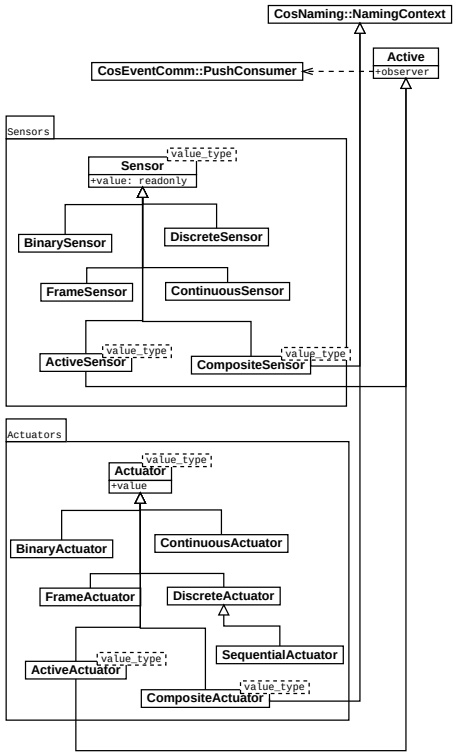

Figure 2. Simplified UML diagram showing SENDA core interfaces for Sensors and Actuators.

members may be required to register at initialization time.

Although we require a minimal subset of the standard services to interoperate within the SENDA framework it is still valuable to use standard CORBA Common Services for better interoperability and scalability. At the device level, a small residential gateway or a microcontroller based module may implement a simplified interface. On the other hand, at the regional level or even at the nation-wide level service providers may use COTS based mature implementations of CORBA services for increased fault tolerance, parallelization, replication, easier management, etc.

3.1 SENDA event channels

Similarly, event propagation is delegated to standard CORBA *CosEventChannelAdmin::EventChannel* service. Again, at the device level we provide a simplified implementation with just the push style methods required by SENDA. At a higher level, a service provider may use a COTS based implementation of an EventChannel able to handle millions of events per second.

A set of wrapper classes adapt passive modules to be *PushSuppliers* of an EventChannel, using private polling procedures, while active modules may be connected directly to an event channel.

In order to keep the event generation rate to a manageable level the service provider may organize a hierarchy of event channels into the composite modules. For example, a provider of a security and vigilance service is not actually interested in every activation and deactivation of a set of motion sensors, a single *intrusion* event would suffice. Therefore the service provider will set up an *intrusion detection* filter (*PushConsumer* of a channel and *PushSupplier* of another channel) as part of the basic infrastructure needed by the customers.

3.2 Building services and managing a SENDA network

Given that SENDA defines a limited set of core interfaces to access all devices in the network (summarized in figure 2), and due to the pervasive usage of standard CORBA services, it is possible to build generic managing tools and services.

Any specialized module must implement a set of core interfaces in order to expose all the available state and control features of the device. For example, a camera object must export all the available settings (zoom, pan, tilt, focus, configuration of the video stream, etc.) through a set of core SENDA modules registered into a composite camera module. A generic managing tool or service will be able to change any setting as if they were independent devices, while a specialized user interface may provide tighter control over related settings.

3.3 Transactions and sessions

Some devices such as cameras require the notion of session in order to grab the video stream, and the notion of atomic transaction in order to be confident on the current state of the device. Both are easily implemented using the SENDA model without additional interfaces.

A session or a transaction is just a special kind of actuator whose *value* attribute is another SENDA module. Session and transaction objects create

a new SENDA module each time their *value* attribute is read. Therefore, each client module works with a private session or transaction object. These private objects implement the *BinaryActuator* interface (an actuator with a boolean state) in order to control whether the transaction is commited or rolled back, or the session is renewed or cancelled.

3.4 Privacy and authentication

SENDA considers two strategies to allow secure communication among devices transparently:

- A secure transport protocol. Many CORBA ORBs already support SSL (*Secure Socket Layer*) as a transport protocol.
- Using CORBA interceptors. That is, registering procedures which are called before reaching the communication channel and before reaching the destination object.

4 Conclusions

In this article we have briefly described the SENDA architecture highligting the advantages over OSGi. SENDA exploits simple abstractions such as sensors and actuators to export functionality in a much more intuitive way than OSGi. In spite of the minimalistic interfaces, our prototype shows that those abstractions are perfectly capable of modeling more elaborated interactive modes, such as sessions, transactions, per usage objects, leased objects, etc.

SENDA is also a proof of the suitability of CORBA for low-end devices, contrary to what is stated in e.g. [5].

References

1. Open Services Gateway Initiative, Available online at http://www.osgi.org/. *OSGi Service Platform, 2.0 edition, October 2001.*
2. *Sun Microsystems, Available online at http://www.sun.com/.* Jini Architecture Specification, 1.2 edition, December 2001.
3. Sun Microsystems, Available online at http://www.sun.com/. *Jini Device Architecture Specification, 1.2 edition, December 2001.*
4. *Li Gong. A Software Architecture for Open Service Gateways.* IEEE Internet Computing, January-February 2001.

5. Meredith Beveridge. Jini on the Control Area Network (CAN): A Case Study in Portability Failure. Master's thesis, Dept. of Electrical and Computer Engineering, Carnegie Mellon University, March 2001.
6. Yoshito Tobe, Hiroto Aida, and Hideyuki Tokuda. A Lightweight Transport Protocol for Home Networks. In *IEEE Proc. 4th International Workshop on Networked Appliances*, 2002.
7. Francisco J Ballesteros, Fabio Kon, and Roy Campbell. Off++: The Network in a Box. In *ECOOP Workshop on Object Orientation in Operating Systems*, 2000.
8. Object Management Group. *The Common Object Request Broker: Architecture and Specification, 2.3 edition, June 1999. Available online at http://www.omg.org/.*