Scalable Computing: Practice and Experience
Volume 12, Number 1, pp. 121–135. http://www.scpe.org
ISSN 1895-1767
© 2011 SCPE

SERVICE COMPONENT ARCHITECTURE EXTENSION FOR SENSOR NETWORKS

PAWEL BACHARA *AND KRZYSZTOF ZIELINSKI†

Abstract.
The role of small electronic devices has been growing over the past years. Many types of devices which allow for interaction with the environment can now be found in the market and there is large set of solutions for integration of mobile devices with enterprise systems. However, most such solutions focus only on providing base services for other SOA system elements. The paper presents an extension of the SCA programming model to include intelligent sensors and a concept of uniform programming model shared between sensor networks and standard computers. The architecture of such an extension is proposed and an initial prototype implementation is presented. The solution utilizes a proxy pattern, validated by a simple case study using Sun SPOTs and Sentilla. The advantages and limitations of the proposed extension are also discussed.

Key words: SOA, Sensor networks, Service Component Architecture, Sun SPOT, Sentilla, Proxy, Multiproxy

1. Introduction. The role of small electronic devices is constantly growing and they have already become an integral part of our lives. Their computational power is also on the rise despite their progressive miniaturization. This work focuses on intelligent sensors and sensor networks. Current generations of smart sensor devices can work either separately or collectively, providing important information about their environment. They may collect simple data such as temperature, acceleration or humidity, or measure more sophisticated data, e.g. concentration of different substances. Intelligent sensors can be embedded in everyday devices and applied to controlling business processes. Additional examples can be found in medicine, where sensors are being applied in patient care, to constantly collect data about various vital parameters. Another promising area is the industry domain where RFID sensors are used to monitor warehouses and production lines.

This paper extends the concept presented in [1], namely that of building SCA applications with use of intelligent sensors with inbuilt communication capabilities. Intensive work on emerging technologies, e.g. Sun SPOT [13] and Sentilla [15] points to the need for meeting business needs. The number of business scenarios which require intelligent sensor devices and their networks is constantly on the rise. While programming embedded devices used to require dedicated platforms, nowadays the computational and communication capabilities of these devices have increased and they can be fully integrated with enterprise applications as extensions of their respective programming models.

SOA appears to the most commonly accepted enterprise software programming paradigm. The goal of this paper is show how the SOA-related software platform called the Service Component Architecture (SCA) [9] can be used to provide a uniform way to access components deployed on standard computers and on intelligent sensor devices. The architecture of such an extension is described and a prototype implementation is validated in the context of a simple case study. The advantages and limitation of the proposed approach are discussed. The paper also compares the proposed solution to the Service Oriented Device Architecture (SODA) [2] and Mobile SOA (MSOA) [16]

The structure of the paper is as follows. Section 2 presents the problem of integrating sensor networks with systems built according to the SOA paradigm and discusses the most important assumptions. Related work is presented in Section 3. The SCA technology and the programming model it involves are briefly characterized in Section 4. In Section 5 two intelligent sensor network solutions are described. Section 6 introduces an architecture of the proposed extension of the SCA programming model. Details of the proposed prototype implementation are presented in Section 7. Section 8 presents the simplicity of using the proposed solution and the benefits it offers. A practical usage example is described in Section 9. The paper ends with conclusions.

2. Device integration problem in SOA. The SOA Solution Stack (S3) [4] proposed by IBM is often cited as a reference for classification of software platforms supporting service-oriented enterprise software [5]. It comprises the following nine layers: Operational System, Service Component, Services, Business Process, Consumer, Integration, QoS, Information Architecture, and Governance and Policies.

*University of Science and Technology, Department of Computer Science, Krakow Poland, Tel.: +48-12-6173902 (bachara@agh.edu.pl).
† University of Science and Technology, Department of Computer Science, Krakow Poland, Tel.: +48-12-6339406 (kz@agh.edu.pl).

Support for programming sensor network devices should be located in one of the presented layers. This decision determines the required level of software abstraction as well as the set of nonfunctional requirements that should be satisfied. Choosing different layers has been considered. Following detailed analysis of the SCA specification we have decided to represent an intelligent sensor as a software component [11]. This choice is justified by the fact that software components represent well-defined functionality exposed by services and also directly specify what is required from other components, enabling seamless integration.

Apart from representing devices as software components the following functional requirements are important [1]:

- Support for bidirectional communication between devices and the enterprise system.
- Dynamic discovery of devices and connection setup.
- Support for concurrent ad-hoc communication using many devices.

Other requirements, mostly nonfunctional in nature, even more strongly support the representation of small mobile devices as software components. They are as follows:

- Genericity - the representation should be usable by different applications.
- Ease of use - this is a key issue. The presented solution should effectively support most use cases rather than support all of them in a very complicated way.
- Scalability - The presented solution should be designed for use by a large set of devices.

There are also some nonfunctional requirements which are common for distributed applications, but should be mentioned for the record:

- Security - this issue affect all distributed systems.
- Reliability - this is a key aspect of in distributed systems where single points of failure must be avoided.

Taking these considerations into account the most suitable solution is to integrate small devices within the Service Component layer of the S3 model. Further analysis should therefore concentrate on the Service Component Architecture as the programming platform most commonly used in this layer [1].

3. Related work . The concept of integrating mobile devices with SOA systems is not new and many works directly address this topic. The most popular approach is SODA (Service Oriented Device Architecture) [2]. This architecture focuses on the layer separating the physical and digital realms. Integration of devices with the SOA runtime infrastructure is realized by the Enterprise Service Bus (ESB), exploiting the Integration Layer of the S3 model. Typically, external devices are connected to ESB via the Web Service Binding Component and their functionality is exposed as Web Services. SODA services allow programmers to use sensors and actuators as standard business services.

The details of integration with physical devices in SODA are hidden behind higher-level abstractions. Service interfaces separate enterprise developers from internal hardware-specific issues and allow for easy modifications to service implementations when migrating to a new generation of sensors or actuators. Another advantage of the SODA approach are the well-defined interfaces that can be tested independently of any application and may be reused in various applications, reducing the overall cost of system development.

The three main components of SODA are:

- Device adapters talk to device interfaces, protocols and connections on one side, and present an abstract service model of the device on the other side.
- The bus adapter moves device data over network protocols by mapping a device service's abstract model to the specific SOA binding mechanism used by the enterprise.
- The device service registry provides for the discovery and access to SODA services.

SODA has become most popular solution in the industry. It is also important that SODA implementations are not limited to specific technologies and any new devices and SOA standards may be used. However, in contrast to the solution proposed in this paper, SODA focuses on integration support and does not expose any uniform programming model. SODA exposes device functionality to SOA systems but treats them as external system elements, as shown in Figure 3.1.

Another alternative to SODA is proposed by the iMobile framework [3]. The iMobile idea appears somewhat similar to the solution presented in this paper, compared to other existing solutions (such as e.g. Mobile SOA [16]). iMobile is a proxy-based platform that addresses the research issues in building mobile services. The key element of this solution is a message gateway which allows integration of mobile devices using various protocols on different access networks. The iMobile specification consists of three main elements:

- A devlet, i.e. a driver attached to a proxy. It receives and sends messages through a particular protocol.

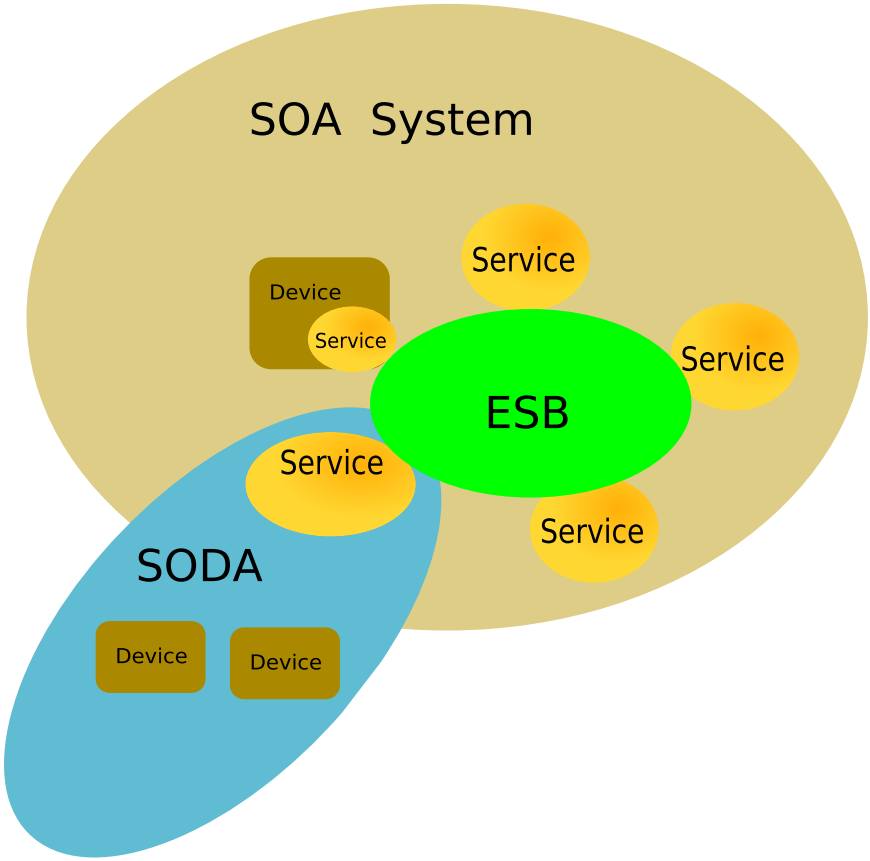

FIG. 3.1. *Device integration using SODA*

- An infolet hosted on iMobile, using an access method to provide an abstract view of the information space.
- An applet, implementing a service or application logic by processing information from various infolets.

The core of iMobile, the let engine, implements the basic framework for maintaining applets, devlets and infolets. It supports user and device profiles for personalization and transcoding, and invokes proper applets and infolets to serve requests from a devlet [3].

In summary, there is a marked difference between the presented solutions and the approach discussed in this paper. SODA aims to expose mobile devices as services and integrate them with external components, in compliance with the SOA paradigm, but it also limits devices to functions provided by Web Services. iMobile operates on a lower level and can provide better device utilization but it remains an experimental system with a relatively rigid architecture and it is not easily integrated with existing technologies [1].

4. Service Component Architecture. The Service Component Architecture (SCA) is a highly promising emerging standard for enterprise software development and integration of heterogeneous components. The specification of this technology involves a model for building applications and systems composed from services. In this sense, it remains very much in line with the Service-Oriented Architecture (SOA) [5]. The SCA programming model for applications and systems is based on the notion that business functions are provided as a series of services, assembled together to create solutions that serve a particular business need [9]. Applications can be constructed as composites which may contain new services along with business functions derived from existing systems and applications. This programming model aims to encompass a wide range of technologies for implementation of service components and to support communication and operation invocations between them [9] [10]. The SCA specification defines components as atoms used to build applications. Different program-

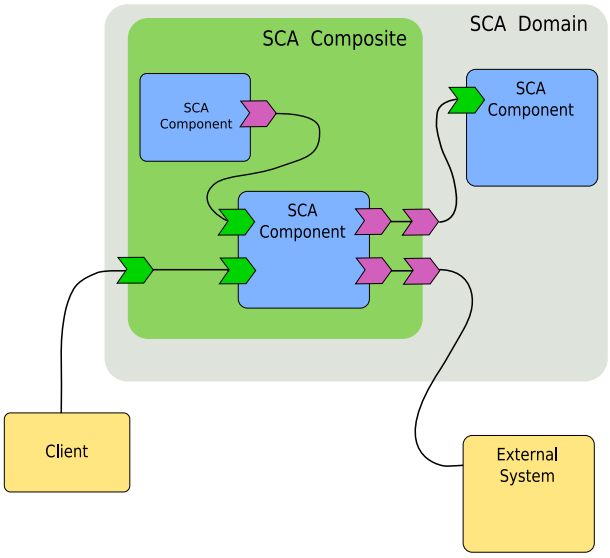

FIG. 4.1. *SCA application example*

ming languages can be used to create components. What is more, many popular frameworks and development environments associated with those languages are supported. Each component is an instance of an implementation configured using properties. The implementation of a component can be provided as a simple Java class but it may also assume the form of a Ruby script or even a BPEL process. In another paper [1] we demonstrate that it can also be represented as a mobile device, focusing on the properties of a sensor network. Important elements of SCA architecture include services and references. They allow components to communicate with one another and with external systems. References are responsible for defining the set of interfaces required by a component. The exposed functionality is described by services. Communication protocols used to access services or references are independent of their definitions.

An SCA application is built as a set of interconnected components. The basic unit of composition within an SCA system is the composite. It is an assembly of components, services, references and other interconnecting composites. All these elements can run on a single host or be distributed over a network; moreover, various technologies can be used in the scope of a single composite. The domain is the topmost element of SCA and aggregates all the above mentioned elements. It can span many computers and – thanks to proposed solution – involve the sensor network domain. An example of a simple SCA application illustrating the presented concepts is shown in Figure 4.1 .

An important advantage of the SCA architecture lies in its extensibility mechanisms defined in the specification. They describe how new bindings and implementations can be introduced. Developers who implement support for new technologies need to provide certain standard interfaces by which new components can cooperate with any other binding or implementation. For our research we chose Tuscany – the most widely used Java implementation of SCA, supported by the Apache Software Foundation [7] [12]. Its main benefits are as follows:

- open-source project,
- very good implementation of Java,
- wide range of component implementation technologies and binding types,
- large and active community.

Before we present our extension of SCA for sensor networks it is necessary to briefly discuss intelligent sensor devices.

5. Intelligent Sensors. Two relatively new technologies: Sun SPOTs [13] and Sentilla Perk [15] have been chosen as intelligent sensor samples. They have been selected because of their innovative potential and the large number of new applications they can support. They both represent a group of devices able to run a

FIG. 5.1. *Sun SPOT device [1]*

virtual machine but they differ on the conceptual level. Sentilla exposes a set of cooperating sensors as a unit (a sensor network), while Sun SPOT devices can work as a sensor networks, but also as standalone components.

5.1. Sun SPOT. A Sun SPOT (Small Portable Object Technology) device is a small, wireless, battery-powered experimental platform. It is programmed almost entirely in Java to allow regular programmers to create projects which used to require specialized embedded system development skills. The Sun SPOT includes a range of built-in sensors as well as the ability to easily interface external devices [13].

A free-range Sun SPOT device contains three main elements: a SPOT processor board, a sensor board and a battery. It is packaged in a plastic housing. The smaller basestation Sun SPOT consists of just the processor board in a plastic housing. Sun SPOT has a 180MHz 32-bit ARM920T core processor with 512K RAM and 4M Flash. The processor board features a 2.4GHz radio (IEEE 802.15.4 compliant) with an integrated antenna on board. Each processor board also provides a USB interface to connect to a PC. [13]. The basic Sensor Board includes the following components: a 3-axis accelerometer (with two range settings: 2G or 6G), a temperature sensor, a light sensor, 8 tri-color LEDs, 6 analog inputs readable by an ADC, 2 momentary switches, 5 general-purpose I/O pins and 4 high-current output pins. It is possible to replace the built-in sensor boards with custom devices. To provide communication with standard computers a Sun SPOT device can be connected to a USB port and work in a "basestation" mode. Owing to dedicated drivers, any Java application may access the functionality of a connected device and communicate with other nodes of the sensor network.

The Squawk Java Virtual Machine (JVM) works on Sun SPOTs. It is an open-source virtual machine for the Java language. Squawk is meant to be used in small, resource-constrained devices and has a small memory footprint. Reduction of memory usage on mobile devices has been achieved by performing many tasks (such as verification, optimization and conversion to an internal suite format) on the host machine during deployment. Suites are deployed onto Sun SPOT devices and are interpreted by the on-device VM. Squawk bases on the CLDC specification and introduces some important restrictions such as:

- java.io.Serializable cannot be used,
- Java Reflection API is not available,
- Remote Method Invocation (RMI)is not supported,
- User-defined class loaders have to be used.

Due to these restrictions using Sun SPOT devices with enterprise frameworks is much more complicated. For example, Tuscany uses the Reflection API to wrap managed components.

5.2. Sentilla Perk. The second type of device which has been selected for our work is Sentilla Perk. A standalone device and gateway used to communication with a PC are presented in Figure 5.2. Sentilla refers to elements of pervasive networks as motes or Jcreate pervasive computers. A single Sentilla mote is structurally

FIG. 5.2. *Sentilla devices and gateway [1]*

similar to SPOT devices but it is designed to reduce power consumption and operate at limited power. That's why it only provides an 8 MHz 16-bit Texas Instruments MSP430 microprocessor with 10 KB RAM. Flash memory capacity is limited to 1072 KB. Similar to Sun SPOT, wireless communication is built over IEEE 802.15.4. Each device also contains the following built-in sensor/effectors:

- a 3-axis accelerometer,
- a temperature sensor,
- 8 LEDs,
- two standard connectors (analog inputs),
- an expansion connector.

A USB gateway device provides communication between mobile devices and the host PC. It is also possible to install new applications thorough this wireless connection. Sentilla VM runs on Jcreate pervasive computers. Similarly to SPOTs, it is also based on CLDC 1.1 (with additional restrictions). RAM is limited to 2KB for user applications.

An important difference between SPOTs and Sentilla lies in the application deployment process. In Sun SPOT sensor networks each device requires separate deployment and different nodes can run different programs. In a Sentilla sensor network the application is deployed to the network and all participating devices must share the same code.

6. Architecture - small device Proxy pattern. To satisfy the requirements presented in section 2, extensions of the SCA platform have been designed and validated. This section presents the architecture and describes two different variants of its realization.

6.1. Mobile Proxy. The SCA specification provides two ways to introduce new elements (i) a specific binding which defines the protocol of communication with interface implementations, or (ii) creating a custom component implementation. The former solution requires low-level programming which is device-specific. The latter option is more general as it allows developers to introduce components implemented in different programming languages and frameworks, hiding all hardware dependencies [1].

This paper presents the concept of extending SCA over the sensor network domain by creating components which represent a device in the SCA application. Users can access the functionality provided by sensors or effectors and do not have to build code which would be deployed to a mobile device: they can simply refer to a new component called a mobile proxy instead. Implementations for SpotProxy and SentillaProxy have been created. The concept of our solution is presented in Figure 6.1 and a more detailed architecture can be seen in Figure 6.2.

To expose a service (green arrow) and provide access to device functions a dedicated component has been created. All communication details are hidden by this element. The out component has two main parts: the first one responsible for integration with SCA and providing communication on the host side, while the second one operating on mobile devices and responsible for precessing requests. When a device function is called using the exposed interface all request parameters are marshaled and sent to the assigned device (or devices). The remote device unpacks these parameters and method descriptions, and then invokes the selected action. Any obtained result is marshaled and sent back to the server. Finally, the component method returns the result

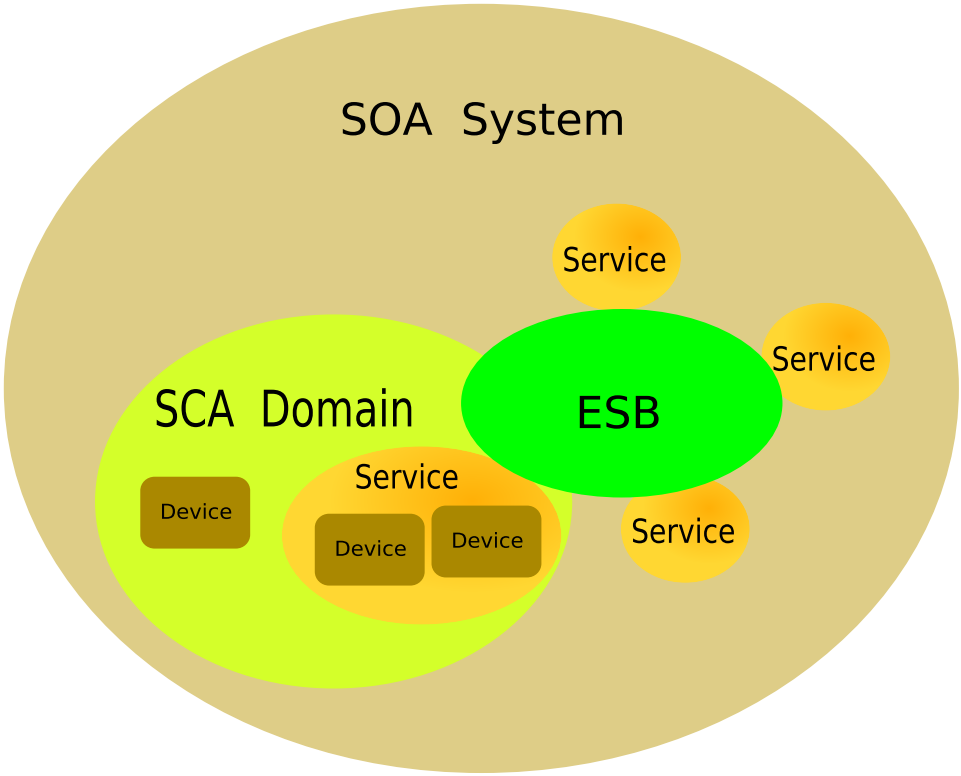

FIG. 6.1. *Device integration using SCA* .

value [1]. The presented solution simplifies access to sensor network devices. Its operation can be described in several simple steps:

1. Deploy the mobile part of the proxy component to sensor network devices (Our implementation provides an option to do it automatically).
2. Connect the basestation or gateway to a host computer.
3. Add a new component (built as a SpotProxy or SentillaProxy implementation) to SCA configuration files.
4. Configure and implement components which will use our sensor network base service.

A detailed description of how the application is constructed can be found in Section 8.

Based on the presented architecture, a mobile proxy implementation has been developed. In its simplest version the proxy represents one specific device. In our case it is identified by a MAC address provided by the end user. This version can be used for static sensor networks where we know which device will provide data of interest to the user. Sometimes this solution could prove sufficient, but in most scenarios we will lose most of the benefits of using sensor networks. Moreover, it is hard to utilize this version for a large set of devices. This is why a more complex alternative – the Mobile Multiproxy – has been designed.

6.2. Mobile Multiproxy. Due to the limitations of the simple version of Mobile Proxy, another solution has been proposed, utilizing the so-called Multiproxy which represents more than one small device. This component can represent a dynamic set of devices, reflecting the changes in the sensor network. The architecture of a Multiproxy is depicted in Figure 6.3. The concept of a context is used to define which devices should be represented by the Multiproxy. User are able to configure a set of properties applied to each sensor. Different options for defining sets of devices are provided:

- Address base:
  - simple device address
  - list of device addresses

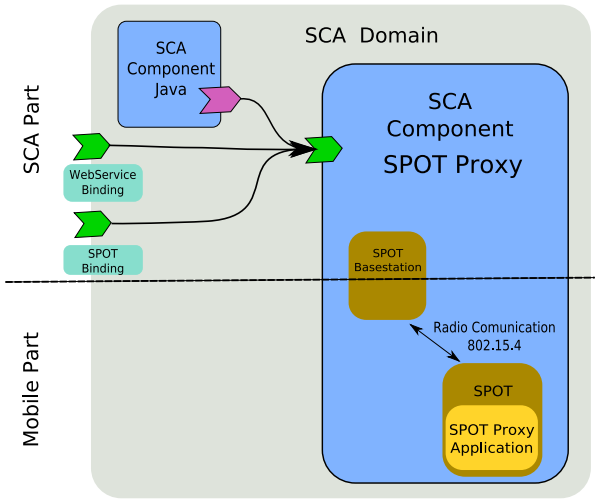

FIG. 6.2. *Spot proxy Architecture*

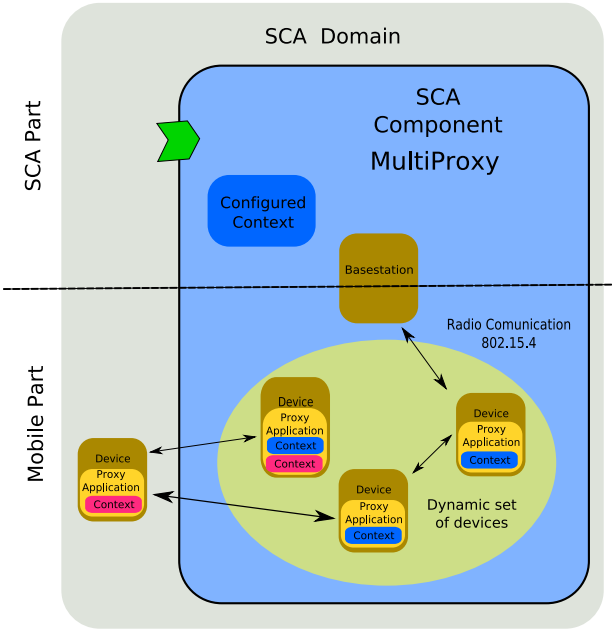

FIG. 6.3. *Multiproxy Architecture*

  - regular expression to match device addresses
- Parameter base:
  - simple text
  - list of texts
  - regular expression to match configured texts
- Dynamic:
  - hop number in the network
  - radio signal strength
  - sensor status

It is important to note that a single physical sensor may be utilized by different components at the same time. To make both implementations of the mobile proxy interchangeable the interface exposed by a multiproxy component is identical to the one exposed by the standard version. Unfortunately, this approach requires the Multiproxy to aggregate results of operations invoked on many devices. Following analysis of typical scenarios, several policies can be proposed:

- callAny - Choose any device and invoke an operation on it. The operation can be invoked on many devices but only one result is received (the first one). Another option is to send a request only to the device which (usually) returns the result in the least amount of time.
- callAnyRR - Similar to the previous one. The difference is that the system may try to use a different device for each invocation.
- callAll - Invoke the operation on all devices assigned to a component. In this case, if the user wishes to receive results, an aggregation should be chosen:
  - Maximum value
  - Minimum value
  - Logical AND or OR
- callMost - Similar to the previous one, but to provide results faster it only waits until a certain percentage of results is obtained.
  - Median
  - Average

The Multiproxy component enables better exploitation of the integration of dynamic sensor networks with SOA applications. Key advantages of this concept are as follows:

- support for dynamic sets of devices;
- components utilize available resources in an automatic way;
- communication with mobile devices is completely transparent to the user;
- users do not need to provide any code for the mobile devices;
- usage of mobile device functions is very easy;
- more effective management of SCA configuration due to limited number of components.

The existing implementation of the Multiproxy component constantly monitors the sensor network and maintains a list devices assigned to it. As part of our future work we want to implement and validate a more dynamic solution where devices execute operations only upon making an autonomous decision on being assigned to the component which has sent the request.

7. SPOT proxy implementation. The implementation of the proposed solution is presented on the example of a Mobile SpotProxy. The most important element of the implementation (from the developer's perspective) is its interface, exposed by the SpotProxy component and presented in Figure 7.1. In this example the user has access to all built-in SPOT sensors and can use LEDs and input/output connectors [1].

Another important element is the configuration of the SCA composite, defined as an XML file:

```
...
<element name="implementation.spot" type="e:SpotProxyImplementation"/>
<complexType name="SpotProxyImplementation">
   <complexContent>
     <extension base="sca:Implementation">
      <sequence>
        <element name="device" type="e:Device"
                  minOccurs="1" maxOccurs="1"/>
        <element name="spotProperties" type="e:spotProperties"
                  minOccurs="0" maxOccurs="1"/>
      </sequence>
      <attribute name="lazy" type="boolean" default="true" />
     </extension>
   </complexContent>
</complexType>

<complexType name="Device">
```

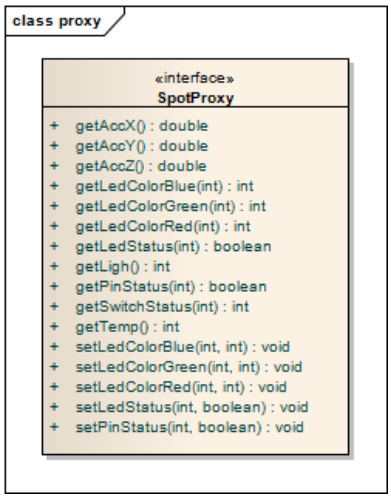

FIG. 7.1. *SpotProxy Interface [1]*

```
   <attribute name="spotPort" type="int" />
   <attribute name="spotAddress" type="anyURI" />
   <attribute name="spotName" type="QName" />
</complexType>

<complexType name="spotProperties">
   <attribute name="timeToLive" type="int" default="10"/>
   <attribute name="maxBroadcastHops" type="int" />
   <attribute name="chanelNumber" type="int"/>
   <attribute name="outputPower" type="int"/>
   <anyAttribute />
</complexType>
...
```

The implementation.spot element has the following attributes and subnodes:

- **implementation.spot** - This is the generic SPOT implementation type. The type is extensible so it is possible to add additional domain-specific attributes and elements. It extends the sca:Implementation type.
- **implementation.spot/Device** - Identifies the assigned device.
- **implementation.spot/Device/@name** - The name of the destination to which the proxy is connected.
- **implementation.spot/Device/@spotPort** - The port which will be used to establish the connection.
- **implementation.spot/Device/@spotUrl** - The MAC address of the assigned device.
- **implementation.spot/spotProperties** - The element which facilitates connection configuration.
- **implementation.spot/spotProperties/ @timeToLive** - Timeout value when waiting for a response. Default is 10s.
- **implementation.spot/spotProperties/ @maxBroadcastHops** - Used for datagram broadcast connections; limits broadcast range.
- **implementation.spot/spotProperties/ @channelNumber** - Forces a specific channel number.
- **implementation.spot/spotProperties/ @outputPower** - Sets the output power of a radio device. Default is maximum power.

The presented configuration is used to create an instance of the Mobile Proxy component in the SCA application. Key classes are presented in Figure 7.2. Instances of the SpotProxyImplementationImpl class store parameters configured in the domain definition file and they are created by SpotProxyImplementationFactoryImpl. SpotProxyProcessor is used to parse the XML configuration. Based on its configuration, the component can send and receive messages using SpotProxyExecutor whenever SpotProxyImplementationInvoker receives a

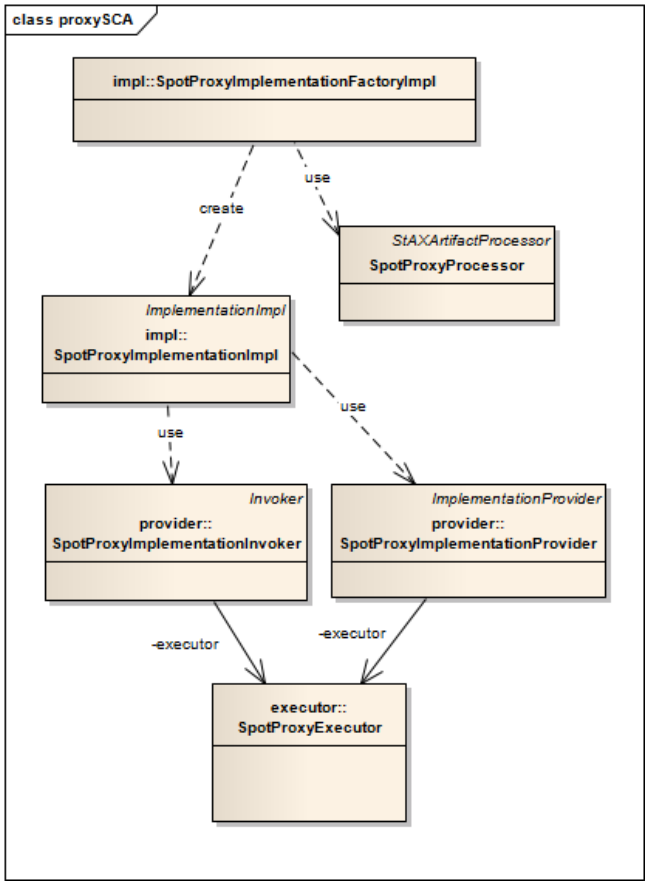

FIG. 7.2. *Main classes of Mobile SpotProxy [1]*

request to call some operation.

8. Application development workflow. Creating an application which utilizes sensor networks to provide data for large-scale enterprise SOA systems is a challenging task which requires in-depth knowledge of both domains. The proposed extension of the SCA architecture is designed to simplify it as much as possible. This section describe the steps necessary to create a simple SOA service. The application can turn a light on and off using Web Service access. It also monitors system temperature and disables the light when the temperature exceeds a specific safety threshold. For manual control, another device connected by the spot binding can be used. The entire system is depicted in Figure 8.1.

1. The first step is to define external interfaces of the exposed service.
2. The second step is to implement a class which provides the required functionality and implements the prepared interface. Access to the physical device is hidden by the SpotProxy interface presented in the previous section. For example, to turn the light off or on, the status of one of the output pins needs to be changed:
   ```
   public void setLightStatus(boolean status) {
   spotProxy.setPinStatus(PIN_NUMBER, !status);
   }
   ```
   To read the current temperature, another method has to be called:
   ```
   public int getCurrentTemp() {
   int temp = spotProxy.getTemp();
   return temp;
   }
   ```
   A reference to the proxy object is set by the SCA framework based on the domain configuration files.

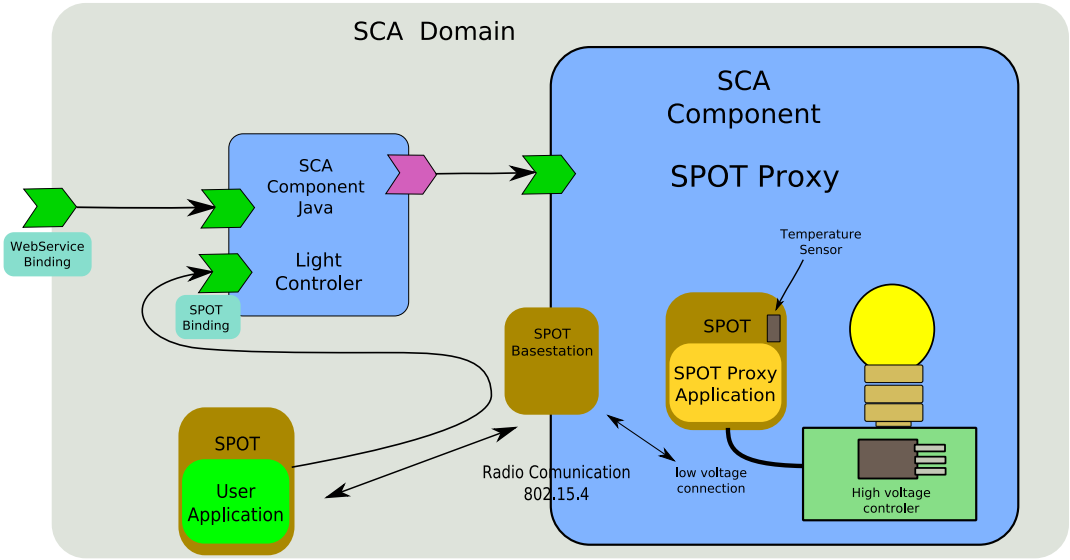

FIG. 8.1. *Light Controller application built using the Spot Proxy component.*

3. The next step is to prepare the physical device. Selected devices should have the SpotProxy application installed. This can be done by using a single **ant** command. In our solution a power module should be connected to the device. This module provides control over higher voltages than the internal SPOT components allow. In our case, a 230 Volt circuit can be controlled.
4. To integrate all of the described software components and sensor network devices, an SCA configuration file need to be created. A complete file is quoted here to show the simplicity of this step:

```
<?xml version="1.0" encoding="UTF-8"?>

<composite  xmlns="http://www.osoa.org/xmlns/sca/1.0"
  targetNamespace="http://spotsca" xmlns:cb="http://spotsca"
  xmlns:e="http://galaxy.agh.edu.pl/bachara" name="spotsca">

 <component name="SpotServiceComponent">
  <e:implementation.spot lazy="true">
   <e:device spotPort="114" spotAddress="0014.4F01.0000.13E4" spotName="testSpot">
   </e:device>
  </e:implementation.spot>
 </component>

 <service name="SpotProxyWeb" promote="SpotServiceComponent">
   <interface.java interface="pl.edu.agh.bachara.spotsca.proxy.SpotProxy"/>
   <binding.ws uri="http://localhost:8086/SpotProxy"/>
 </service>

 <component name="LightManagerComponent">
  <implementation.java
    class="pl.edu.agh.bachara.spotsca.proxy.demo.LightManagerImpl"/>

  <service name="LightManager">
   <interface.java interface="pl.edu.agh.bachara.spotsca.proxy.demo.LightManager" />
   <binding.ws uri="http://localhost:8086/LightManager"/>
   <e:binding.spot >
```

TABLE 9.1
Comparison of startup time and memory allocation.

Application	Startup time [s]	Memory allocated [MB]	Average request time[ms]
SPOT simple	1	9	59
SPOT proxy	6	40	68
Sentilla simple	0.5	7	1259
Sentilla proxy	5	37	1276

```
    <e:destination name="serDest" spotPort="103"
          spotUrl="radiogram://0014.4F01.0000.14D7" spotName="serDestSpot"/>
    <e:response>
      <e:destination name="serReqDest" spotPort="104"
          spotUrl="radiogram://0014.4F01.0000.14D7" spotName="serReqDestSpot"/>
    </e:response>
   </e:binding.spot>
  </service>
  <reference name="spotProxy" target="SpotServiceComponent" />
 </component>

</composite>
```

Let us briefly describe this configuration file. The first component, called SpotServiceComponent, is an instance of the mobile proxy. The assigned device address and communication channel is defined in the configuration. The next step involves exposing the service by using a Web Service binding which allows direct access to SPOT functions thorough the WS protocol. The LightManager component is subsequently defined using a SpotProxy reference and exposes a LightManager interface thorough Web Service and SPOT bindings.

5. To obtain a working application, runtime configuration is necessary. A class with a main method has to be created to spawn the SCA Domain using the prepared configuration:
`scaDomain = SCADomain.newInstance("proxy.composite");`
Required libraries should be added to the classpath and, finally, options required by SPOT libraries should be defined.

9. Performance study . The performance of the presented implementations is validated in this section. To measure the overhead generated by our solution an application devoid of SCA extensions has been prepared. Test results are summarized in Table 9.1. The first column presents a comparison of startup time, measured between application startup and reception of the result of the first remote call. The second column contains the allocated memory volume (measured one minute after application startup). The last column presents the average operation execution time. All tests are performed for a simple network where the device operates in a basestation (or gateway, in the case of Sentilla) radio range. Time is measured between sending a temperature measurement request and receiving a response.

The presented results show that the SCA base programming model for sensor network integration introduces approximately 15% overhead for a single operation call compared to raw implementation. This could be explained by execution of additional framework code in the course of communication and the relative complexity of the protocol used to send requests and receive responses. In some rare cases, the enlarged memory footprint and longer startup time may also become important [1].

To validate the Multiproxy implementation a more complicated sensor network has been created. This network is presented in Figure 9.1. The first test was conducted using a simple SpotProxy configured to use device 4. Because sensors number 3 and 4 are outside the basestation range, all communications have to be routed through devices 1 or 2. The second test validated support for more then one device. Here, Multiproxy was configured to use devices 3 and 4. In the final test, simulating a dynamic sensor network, all mobile devices were configured to be disabled 25% of the time at random intervals (the average duration of an active

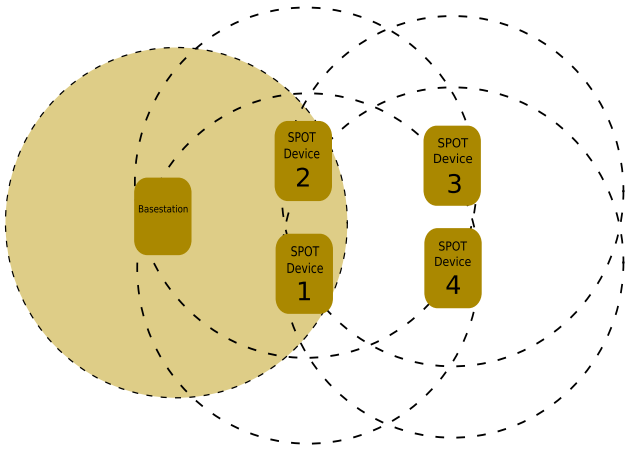

FIG. 9.1. *SpotProxy test configuration.*

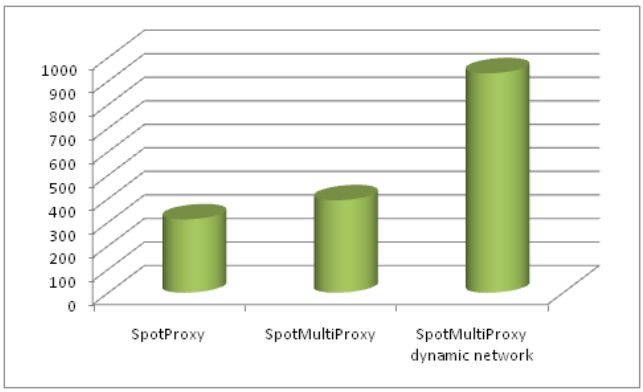

FIG. 9.2. *Average request time in ms.*

session is approximately 30s). Results are presented in Figure 9.2. Multiproxies exhibit longer average request times compared to a standalone implementation. This could be explained by the need to transmit additional monitoring messages concurrent with user requests. Due to the dynamic nature of the network used in the final test, request processing is slowed down by a factor of approximately 3. More detailed dynamic Multiproxy tests have been conducted in relation to the use of ad-hoc protocols but this issue falls beyond the scope of the presented research.

10. Conclusions. Extending the SCA architecture with new components which represent sensor network devices seems to be a very effective solution. Abstracting mobile devices as components enables access to their features consistent with the platform used for system implementation. It also provides a suitable level of decomposition of complex functionality into reusable elements. Thus, the SCA domain can be seamlessly expanded to cover the mobile domain, in addition to standalone computers. The SCA technology seems to be a suitable development platform for integration of both worlds. This work presents a practical verification of this approach on the Service Components Layer but it may also be applied to other layers of the S3 model such as the Services Layer used in SODA.

The proposed mobile proxy pattern used as an internal mechanism of the presented extension is flexible enough to provide the requested level of transparency for single device. The Multiproxy performs the same function for an entire set of devices and simplifies management of the sensor network. The proposed solution requires further evaluation for larger sets of devices and real sensor networks, but preliminary studies look very promising.

11. Acknowledgment. The research presented in this paper was partially supported by the European Union in the scope of the European Regional Development Fund program no. POIG.01.03.01-00-008/08.

REFERENCES

[1] Pawel Bachara; Krzysztof Zieliñski, SOA-compliant programming model for intelligent sensor networks SCA-based solution, SYNASC 2010 : 12th international symposium on Symbolic and Numeric Algorithms for Scientific Computing : 23–26 September 2010 Timisoara Romania, ISBN 978-0-7695-4324-6, pp. 471–478.
[2] Scott de Deugd; Randy Carroll; Kevin Kelly; Bill Millett; Jeffrey Ricker, SODA: Service Oriented Device Architecture,IEEE Pervasive Computing, vol. 5, no. 3, pp. 94-96, July-Sept. 2006
[3] Chung-Hwa Herman Rao, Yih-Fam Robin Chen, Di-Fa Chang, Ming-Feng Chen, iMobile: A Proxy-Based Platform for Mobile Services, Workshop on Wireless Mobile Intemet 7/01 Rome, Italy 2001
[4] Arsanjani, A.; Liang-Jie Zhang; Ellis, M.; Allam, A.; Channabasavaiah, K., S3: A Service-Oriented Reference Architecture IT Professional Volume 9, Issue 3, May-June 2007 Page(s):10–17
[5] Thomas Erl, "Service-oriented Architecture: Concepts, Technology, and Design" Upper Saddle River: Prentice Hall PTR. ISBN 0-13-185858-0.
[6] Dave Chappell, "Enterprise Service Bus" , O'Reilly: June 2004, ISBN 0-596-00675-6
[7] Apache Tuscany Project. http://tuscany.apache.org/
[8] JSR-000139 Connected Limited Device Configuration 1.1
[9] OSOA - Service Component Architecture Home. http://www.osoa.org/display/Main/Service+Component+Architecture+Home
[10] IBM Service Component Architecture. http://www.ibm.com/developerworks/ library/specification/ws-sca/
[11] SCA Service Component Architecture, Assembly Model Specification. http://www.osoa.org/download/attachments/35/SCA_AssemblyModel_V100.pdf
[12] SCA Java Component Implementation V1.0, Assembly Model Specification http://www.osoa.org/download/attachments/35/ SCA_JavaComponentImplementation_V100.pdf
[13] Project SUN SPOT. http://www.sunspotworld.com/
[14] The Squawk Project. http://research.sun.com/projects/squawk/
[15] Project Sentilla Perk: Pervasive Computing Kit. http://www.sentilla.com/developer.html
[16] Ari Shapiro, Andreas Frank , Mobility Service Oriented Architecture Extending SOA to Mobile Devices http://developers.sun.com/learning/ javaoneonline/ 2007/pdf/TS-5639.pdf

Edited by: Dana Petcu and Alex Galis
Received: March 1, 2011
Accepted: March 31, 2011