Abstract

The use of Internet as a general purpose communication system is growing very fast in all sectors of activity. Multiple forms of equipments are now connected to Internet and have gained increased embedded intelligence. This evolution leads to new interactive services that are embedded on these smart objects while interoperating with existing enterprise information systems. The objective of an operation support system (OSS) is twofold: reducing operation costs by automated administration of smart objects, retrieving valuable operational and business information from smart objects to create new sources of revenue. The paper describes the design of a flexible and scalable operation support system for networked smart objects that meets these requirements.

1. Introduction

The use of Internet as a general purpose communication system is growing very fast in all sectors of activity (banking, manufacturing, energy, healthcare, building/home automation, etc.). Multiple forms of devices and appliances are now connected to Internet and have gained increased embedded intelligence. This evolution leads to emerging interactive services that are embedded on smart objects while interoperating with existing enterprise information systems. Operation support for equipments and hosted services is a major issue for the various actors involved: end users, device manufacturers, service providers. The term “operation” refers here to the set of management functions involved in the object/service life-cycle: deployment, configuration, monitoring, maintenance and reconfiguration as necessary. The purpose of an operation support system (OSS for short) is twofold. On the one hand automated administration allows a reduction of operation costs associated with the management of a large number of heterogeneous smart objects. On the other hand, intelligent devices contain valuable operational and business information that can be retrieved in real-time to feed business applications such as remote supervision, billing, service promotion and customer relationship management, etc.

From the technical point of view, these objectives are achieved through the definition of an efficient and flexible operation support system that provides two types of services:

- **Service Provisioning** to install and configure software components on remote objects. Components implement both end-user services and system support functions.
- **Data Mediation** to collect and process usage data from smart objects (and hosted services) and to deliver them to business applications within the enterprise information system (e.g. monitoring, billing, CRM, etc.).

This paper focuses on the design of a distributed mediation infrastructure that meets the needs of operation support systems for networked smart objects. Mediation infrastructures are in the heart of IT systems as they provide an integration facility between a set of heterogeneous remote autonomous devices and business applications. They enable the seamless integration of usage data from devices with ongoing business activities in real-time, thus leading to faster reaction times, proactive customer services and improved operational efficiency.

Section 2 presents the main requirements for the management of distributed smart objects. Based on this analysis, section 3 describes the basic design choices to meet these requirements. Issues addressed are: architecture, middleware services, programming model, development and management tools. Section 4 describes the components of the ScalAgent mediation infrastructure that aims at providing an actual implementation of this design architecture. Current status and future work are presented in section 5 to conclude.

2. Requirements

The term “mediation” was born in the telecom area, where the mediation building-block is now recognized as a strategic element at the forefront of numerous business applications such as rating and billing, supervision and QoS management. Emerging applications involving smart objects - such as health care systems, building/home automation, energy systems, etc. - are today facing similar problems, but the solutions adopted so far in the telecom area are not totally suited for them because of specific issues involved in the management of networked smart objects. Two major requirements are identified.

- **Scalability** due to the very large number of heterogeneous objects involved. In addition, objects are joining and leaving the system frequently. Moreover, a large number of objects usually imply large volumes of usage data to be collected and processed. This leads to technical issues such as limited network bandwidth and computing power available to handle usage data.

- **Flexibility** to adapt the mediation infrastructure to the characteristics of the physical environment and to the user needs (end-user, device manufacturer, service provider). The infrastructure should also be able to evolve rapidly to address new requirements (e.g. new object and service, new billing strategy) and changing operational conditions (e.g. networking topology). The level of flexibility can be measured by the time (and the development cost) needed to adapt the existing infrastructure to evolving parameters. This obviously requires dynamic reconfiguration capabilities.

Additional features are required such as availability, reliability and security amongst others. Availability and reliability are addressed below. As far as the security issue is concerned, we think that existing internet-based solutions can be used for networked smart objects.
Mediation solutions currently available on the market have severe limitations that do not meet the scalability and flexibility criteria mentioned above. They usually take the form of a packaged product that integrates both data acquisition procedures and the business application code. This “integrated” approach lacks flexibility and configurability in the context of distributed heterogeneous environments. Moreover these solutions follow a centralized client-server architecture. Data are retrieved from remote equipment using basic protocols (e.g. FTP, HTTP, SNMP, etc.) and the whole data processing (i.e. aggregation, filtering, transformation, etc.) is concentrated on the application server. This approach does not meet the scalability requirement.

3. Design Choices

In this section we describe the main architecture choices that allow the requirements described above to be satisfied.

3.1. Architecture: distributed intelligence, i.e. the right function at the right place

It is recognized today that centralized architectures introduce bottlenecks both at the communication and computing levels. They also suffer from a lack of availability as they are subject to single-point failures. Replicated architectures (i.e. cluster systems or replicated data servers) bring a partial answer to this problem but these solutions are complex and expensive.

At the opposite, bringing the processing functions close to the data sources provides two major advantages: i) raw data being processed locally, only pertinent information is reported to the management centre thus saving network bandwidth. ii) the overall computing load can be balanced between the various nodes in a configurable way (smart object itself, intermediate consolidation node, central management node).

Implementing this distributed intelligence approach can be achieved in the following way:

 ✓ Use a component way of programming to structure the management intelligence in fine-grained computing units and deploy them on the right node based on application requirements. Software components and architecture definition languages (ADL) are used to achieve this goal.
 ✓ Adopt the Java language to be independent of a given host environment.

3.2. Middleware services: asynchronous communication to meet availability, reliability and scalability requirements

Asynchronous communication systems (so-called message bus) have numerous advantages on client-server approaches (based for example on HTTP) [1]. Asynchronous communication is better suited to deal with disconnected mode encountered in ubiquitous environments. Reliable communication is achieved by persistent message queuing facilities. In addition, some message buses provide also checkpoint and recovery mechanisms that help designers to build full reliable applications. Scalability is achieved through a distributed implementation of the message bus (i.e. “snowflake” architecture). This approach leads to configurable topologies that can be adapted to the physical constraints (network and nodes capabilities) of a given environment.

Additional features are expected such as:

 ✓ Availability of lightweight versions of the message bus to be embedded in smart objects with limited resources (memory foot-print, computing power, .).
 ✓ Compliance with widespread e-business standards so that information transported by the communication system could be easily delivered to business applications.

The choice of a componentized architecture has been argued above. Following this approach an environment for smart objects can be viewed as a collection of distributed interacting software components. Design and management tools are required to maintain the consistency of the overall application despite the changes involved in the application lifetime. Tools are described in 3.4 below. They rely on a set of middleware services for the deployment, monitoring and control of individual software components. The description of these services is available in [2].

3.3. Programming Model: distributed components

Most of asynchronous communication systems available today provide a proprietary application programming interface (API) that takes the form of a function library (available through various language mappings). The JMS specification (Java Messaging Service) is a first step forward as it provides a Java API independent of a given bus implementation [3]. However, the use of APIs is still cumbersome and error prone. We advocate that better support should be given to application programmers. This can take the form of a simple programming model adapted to the description of mediation functions. From the functional point of view, the actor paradigm [4] is a good basis for such a model as it was designed for an asynchronous world. The model should also provide features for the control of non-functional properties such as atomicity, persistency, security, mobility, etc. This can be achieved through the component approach that clearly separates the functional part of the component (described by its interface) and the non-functional part, described in a so-called “container” [5].

In addition, the component structure can be used to encapsulate external software packages such as operation code on smart objects and legacy applications on the management node. Interoperability with this special type of components is achieved through specific gateway components (called connectors). Therefore, the component paradigm provides a uniform view of the overall distributed application, which is exploited by a set of tools as explained below.

3.4. Development and administration tools: ADL (Architecture Description Languages)

The development of distributed applications is still difficult and time-consuming. Therefore we think that the component-based approach combined with appropriate development and administration tools should greatly help users in the design, deployment and management of distributed applications. Experience drawn from many years of research in this area has convinced us to use an ADL (Architecture Description Languages) to describe an application as an assembling of interacting software components. ADL allow components, their interfaces and properties to be specified, as well as links between them [6]. This overall description is used during the whole application life-cycle to provide a complete and consistent view of what the distributed application is. It is
thus possible to exploit this formal representation to pilot the deployment process, to monitor the application components and later-on to reconfigure part of the application in order to adapt it to evolving user requirements and to changing run-time conditions. This configuration/adaptation capability is a major advantage for users as applications can be tailored according to his needs or those of his own customers.

It should be noted that ADL-based development tools are complementary to (and can cooperate with) modelling tools, especially those based on the UML formalism.

4. Implementation

The ScalAgent Mediation infrastructure is an attempt to provide an actual implementation following the design guidelines described in the previous section. This work has been initiated five years ago as a joint effort between a research laboratory and an industrial team (in the framework of the GIE Dyade). The prototype built so far is now industrialized and supported by ScalAgent Distributed Technologies, a new company, spin-off from Bull and INRIA. The main features of this infrastructure are described below.

4.1. Middleware

The Scalagent MOM (Message-Oriented Middleware) provides an asynchronous messaging service that guarantees messages delivery and causal ordering of messages. The constraints imposed by the scalability issue have greatly inspired the internal design of the Scalagent MOM. The MOM ensures reliable communication paths between components in Internet-based wide-area networks.

The Scalagent MOM is 100% Java, so that it can run on a wide spectrum of stations, servers and Internet appliances. Therefore various classes of distributed applications are supported including those that require communications between smart objects and application and data servers. From the functional and performance points of view the ScalAgent MOM can be favourably compared to other Java-based message buses available on the market (e.g. SonicMQ [7], Tibco Rendezvous[8], etc.).

This MOM can be accessed in two ways using complementary application programmatic interfaces. One is compliant to the JMS™ standard (Java Messaging Service). A more sophisticated API, based on the component paradigm, is also provided to application designers (cf. 4.2.). The Scalagent MOM and its JMS API are available today as an open-source package (called JORAM) on the ObjectWeb platform [9].

4.2. Programming model

The ScalAgent component-based programming model has been designed to be integrated with a MOM and to be bound with ADL tools. ScalAgent components are distributed Java objects that communicate by sending and receiving messages. A message is a Java class that extends a pre-defined class of the run-time system. A component is made of two parts: a functional part (or SCBean) and a container part (or SContainer). A hierarchical construction of components is achieved through the use of the Olan ADL [10].

The functional interface of a SCBean distinguishes provided interfaces from required interfaces. A required interface of a component can be connected to a provided interface of another component if the former conforms to the latter. Connections between provided interfaces and required interfaces are described by ADL statements. The SCBean execution model extends the traditional event/reaction model with additional features such as atomic execution.

A SContainer encapsulates a SCBean. It manages configurable non-functional properties (e.g. persistency, atomicity, security, deployment, monitoring) and implements the mapping with the underlying middleware services.

4.3. Run-time support

Mediation components implement the mediation processing logic. This includes the following operations: collection of usage data generated by smart objects, parsing, filtering, aggregation, as well as correlation, merging and fusion of various data sources, and finally the transformation to different formats. Mediation servers are in charge of hosting and operating mediation components, ensuring secure communications and guaranteed delivery of usage data despite transient network failures and disconnected mode. Servers can operate on any Java enabled host environment. Depending on the type of hardware, two types of mediation servers can be used.

✓ **Mediation Servers** are the full version of the Mediation runtime support. They are configured to ensure part or all of the following properties: persistency of smart agents, guaranteed delivery of mediation data, global ordering of messages, secure communication channels, web-based management, JMX™ enabled management.

✓ **Mediation eServers** are a lightweight version of the Mediation runtime support that provide only a subset of server properties according to hardware capabilities and application needs. Mediation eServers have been developed for various types of devices and smart objects such as: industrial/home gateways (e.g. an OSGI gateway), industrial automatons, Java smart cards, PDAs, SmartPhones, etc.

In addition, specific “business connector” components provide gateways to Enterprise Application Integration middleware standards, such as: JMS™, EJB™ (RMI/IIOP), CORBA (IIOP), …

4.4. Development and administration tools

ADL-based tools assist the application programmer in the design and customization of the mediation solution. They also provide support for the remote control of the installation, configuration and monitoring of software components whatever the complexity of the distributed solution could be. These tools are used through a friendly graphical user interface. The tools currently available include:

- **Component description Tool**, to specify the application architecture: basic components as well as the composition and cooperation rules between components.
- **Component Assembling Tool (or Configuration Tool)**, to define and customize a particular instance of an application: instantiation and configuration of a set of appropriate application components.
- **Application Deployment tool** to install the actual components into their target environment and to set up the links between them. At the end of the deployment stage, the application is ready for execution.
- **Monitoring tool** to control the distributed execution of components.
The description of the entire distributed application is stored in a repository. This description can be modified in two ways: statically using the ADL tools or dynamically using an API. The latter allows new service components to be added as a result of a discovery process.

5. Conclusion

The use of networked smart objects is growing very fast, thus leading to large-scale distributed application scenarios. The operational control of objects and hosted services requires a dedicated operation support system. An OSS provides the integration infrastructure between a set of remote autonomous smart objects and business-oriented applications such as: supervision and QoS management, billing, customer relationship management, etc. OSS infrastructures built for the telecom world are not appropriate because they are lacking features for specific requirements encountered in the management of networked smart objects. On the one hand, applications are evolving very fast, as new types of objects are introduced, network configurations and user requirements are subject to frequent changes. Flexibility is thus a key issue in the design of an OSS. On the other hand scalability is another major feature as the number of smart objects operated can be very large. Flexibility and scalability are not well addressed by today infrastructures, and new technologies are required.

This paper has described the design of a flexible and scalable mediation infrastructure that aims at providing the basic building blocks to implement a scalable and flexible OSS for networked smart objects. The main features of such an infrastructure are: i) distributed intelligence embedded in components running both on smart objects and a set of servers, ii) middleware layer based on a reliable asynchronous communication system, iii) construction and deployment tools based on ADL principles.

A prototype of such a system has been built and experimented in the Internet/Java world. This prototype is being industrialized by a new company ScalAgent Distributed Technologies (see http://www.scalagent.com/).

The following example illustrates the use of the ScalAgent mediation infrastructure for the remote supervision of uninterrupted power supply (UPS) devices. The overall architecture is depicted in the figure below.

An embedded mediation server is running on each UPS device and a full mediation server is running on the supervision node, in cooperation with a J2EE application server.

The mediation application is described as a set of interacting components (grey round boxes). The deployment process (dotted arrows), piloted from a central control station, allows the mediation components to be installed on their execution node.

On the UPS side, components collect usage data (e.g. QoS parameters, alarms) in real-time, and compute pertinent indicators that are transmitted to the supervision node. Indicators coming from all the UPS devices are processed on the supervision node to provide reporting data on one hand and business data on the other hand.

Modifications on the application structure (e.g. evolution of the physical configuration or adding components to compute new indicators required by the operator) are applied to the global application description. Then the automated deployment process is executed to allow parts of the distributed application to be reconfigured as necessary.

Ongoing research work is conducted on dynamic reconfiguration (using the ADL approach) and reliability for very large-scale systems. Finally, it should be noted that mediation infrastructures share a number of properties with distributed infrastructures involved in application integration and data integration. Therefore, another direction of work is to adapt the mediation infrastructure described in this paper to address the construction of a scalable and flexible EAI (Enterprise Application Integration) infrastructure.

6. References