JXTA-SOAP: Implementing Service-Oriented Ubiquitous Computing Platforms for Ambient Assisted Living

Michele Amoretti, Maria Chiara Laghi, Francesco Zanichelli, and Gianni Conte

Dipartimento di Ingegneria dell'Informazione, University of Parma,
Via Usberti 181/a, 43039 Parma, Italy

Abstract. The challenging context of Ambient Assisted Living (AAL) demands for a service-oriented technological shift in the field of ubiquitous computing. Recently, novel paradigms have been proposed, most of them envisioning arbitrary pairs of peer application entities communicating and providing services directly with each other and to users. In order to enforce these paradigms even to systems which include devices with limited processing and storage resources, lightweight middleware components are required. JXTA-SOAP, a portable software component supporting peer-to-peer sharing of Web Services, is a suitable solution. We illustrate its features and a possible deployment to enable AAL services.

Key words: ubiquitous computing, ambient assisted living, services, peer-to-peer

1 Introduction

The concept of ambient intelligence (AmI), which refers to a digital environment that proactively supports people in their daily lives, was introduced by the information Society Technologies Advisory Group (ISTAG) of the European Commission [11]. AmI overlaps with other concepts, such as ubiquitous computing, pervasive computing, context awareness, embedded systems and artificial intelligence [21].

In the AmI context, the European Commission recently started the Ambient Assisted Living (AAL) technology and innovation funding programme, aiming at extending the time older people can live in their home environment by increasing their autonomy and assisting them in carrying out activities of daily living, feeling included, secure, protected and supported. AAL spaces are physical places featured with AmI enabling technologies, including the intelligence which supports the services. Examples of AAL spaces are the home where the user lives, the neighborhood, the town, but also the body of the user itself. The technical challenge is to develop an integrated technological platform that allows the practical implementation of the AAL concept for the seamless and natural access to those services indicated above, to empower the citizen to adopt ambient intelligence as a natural environment in which to live.

In this paper we mainly focus on AAL exploitation based on the concept of *ubiquitous computing*, whose main objective is to provide globally available services and resources in a network by giving users the ability to access them anytime and anywhere. In particular, we consider novel paradigms for regulating the interactions among the software entities of AAL-oriented AmI systems.

Recently, Gaber [7] has proposed two alternatives to the traditional client/server paradigm (CSP) to design and implement ubiquitous and pervasive applications: the Adaptive Services/Client Paradigm (SCP) and the Spontaneous Service Emergence Paradigm (SEP). In other words, the peer-to-peer paradigm is completed respectively by the self-organization and the self-adaptation principles. In SCP a decentralized and self-organizing middleware that implements an intelligent network should be able to provide services to users according to their availability and the network status. In SEP, spontaneous services can be created on the fly and be provided by mobile devices that interact through ad hoc connections without any prior planning.

In order to enforce these paradigms to systems which include devices with limited processing and storage resources, lightweight middleware components are strongly required. In [5], Bodhuin *et al.* compare some traditional solutions for net-centric computing middleware, such as Jini, OSGi and CORBA, listing their pros and cons. Not surprisingly, the survey does not include Sun MicroSystem's JXTA [25], probably due to the fact that in year 2005 an implementation for mobile devices was not completed. JXTA is mainly the specification of a set of open protocols for building overlay networks, independent from platforms and languages. Currently there are three official implementation of JXTA protocols: J2SE-based, J2ME-based and C/C++/C♯-based. In particular, an almost complete version of the JXTA Java Micro Edition (JXTA-J2ME, a.k.a. JXME) has been recently released. It provides a JXTA compatible platform on resource constrained devices using the Connected Limited Device Configuration (CLDC) with Mobile Information Device Profile 2.0 (MIDP), or Connected Device Configuration (CDC). Supported devices range from smartphones to PDAs.

How does JXTA cope with the service concepts characterizing the previously summarized paradigms for ubiquitous computing? The Web Service community considers services as *the only mean* for accessing resources (this concept has been explicitly formalized in the WSRF specification [31]), yet centralized registries, themselves exposed as services (like UDDI), are still deemed the primary tool to support the publication and the discovery phases.

Unfortunately, a peer-to-peer network of Web Service providers with a publication/discovery infrastructure implemented as a set of interacting Web Services would be absolutely unefficient due to the heaviness of the SOAP messaging protocol. On the other side, in JXTA each peer's service is just an example of resource which can be exploited by the user which owns the peer, or shared in the network, *i.e.* advertised by the user and exploited by other users. Resource descriptions have the shape of XML documents, namely advertisements. A JXTA advertisement can be filled with any document, *e.g.* a WSDL interface if the shared resource is a Web Service. In summary, JXTA provides a lot of

flexibility by separating basic infrastructural services, mandatory for all peers, from specialized services, with different levels of description and efficiency.

Within the context of JXTA and Web Service integration, we are responsible for the development and maintenance of the JXTA-SOAP component [12], enabling Web Service deployment in JXTA peers, as well as distributed WSDL publication and discovery, and SOAP message transport over JXTA pipes (*i.e.* virtual communication channels which may connect peers that do not have a direct physical link, resulting in a logical connection bound to peer endpoints corresponding to available peer network interfaces with an example being a TCP port and associated IP address). JXTA-SOAP is currently implemented in two versions: J2SE-based (fully featured, extending JXTA-J2SE) and J2ME-based (partially featured, extending JXME).

The remainder of the paper is organized as follows. Section 2 illustrates User Activity Monitoring as emblematic example of complex AAL service, which requires the cooperation of several components providing diversified information and/or specific services. Section 3 describes related work on AmI (and in particular AAL) systems, middleware for peer-to-peer service-oriented ubiquitous computing, and Web Services on resource-constrained devices. Section 4 illustrates the internal design of the JXTA-SOAP component. Some interesting details of the implementation, referring to both J2SE and J2ME versions of JXTA-SOAP, are given in section 5. Section 6 describes how the component has been tested, with several different settings. Finally, section 7 provides a conclusive discussion and describes future work.

2 AAL service example: User Activity Monitoring

One of the most challenging AAL services is *User Activity Monitoring*, which is transversal to every AAL scenario. The SCP and SEP paradigms illustrated in section 1 are able to provide the flexibility required to deal with highly dynamic environments where devices continuously change their availability and (or) physical location (*e.g.* those which are carried or worn by the user). This complex problem of composing and decomposing connections among nodes is abstracted in an overlay network where the Activity Monitor (AM) component subscribes for raw context events coming from other distributed components (sensors, specialized data filters, etc.), searches for remote services which may provide useful information for its reasoning function, and publishes context events which describe indoor and outdoor activity of the user, taking into account different contour information such as medical prescriptions, planned agenda, etc (figure 1).

A distinction between *static* and *dynamic* activities is necessary. Static activities like "standing" or "sitting" can be inferred directly from the low-level data at a particular time instant (such as the pose of the person at a certain time using some kind of thresholding mechanism on the pose estimate). By contrast dynamic activities, such as "moving around", are usually composite activities requiring a monitoring of a full sequence of low-level data (*e.g.* context events

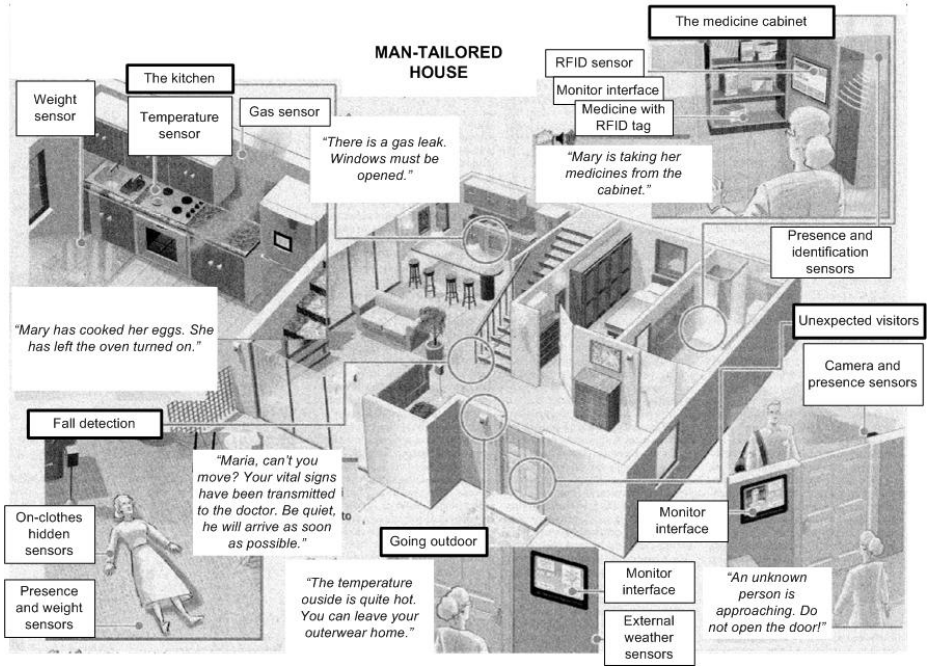

Fig. 1. User activity monitoring (indoor) as envisioned by project PERSONA.

describing ongoing sub-activities). Low-level data needs to be stored for several time frames (in a context buffer), as the whole sequence is needed to infer that activity from an evolution of the low level data. For example: "cooking" may be composed of several low-level data at different time instances: "opening the fridge", "closing the fridge", "standing in front of the oven", etc.

Outdoor user activities are even more challenging to detect. The user may wear a personal mobile device (PMD) and sensors that monitor the level of its activity. The PMD should have a mechanism to be called from an external entity to deliver the activity level. Thus, the mobile device would be both service provider and service consumer. Collected information, which is analyzed in deferred time, may be useful for several other AAL services, *e.g.* planning the weekly menu (the less activity, the less amount of calories to ingest).

3 Related work

In the first part of this section we discuss four recent AmI projects that in our opinion are the most advanced (one of them is clearly AAL-oriented). Then we revise the state of the art of technologies for ubiquitous peer-to-peer sharing of services. Finally, we discuss current middleware solutions for deploying and consuming Web Services on resource-constrained devices.

3.1 State-of-art AmI projects

The Agent-based Intelligent Reactive Environments (AIRE) project [1] is dedicated to examining how to design pervasive computing systems and applications for people. To this purpose, AIRE designs and constructs Intelligent Environments (IEs), which are spaces augmented with basic perceptual sensing, speech recognition, and distributed agent logic. AIRE's IEs have encompassed a large range of form factors and sizes, from a pocket-sized computer up to networks of conference rooms. Each of these serves as individual platform, or AIRE-space on which pervasive computing applications can be layered. Examples of AIRE applications currently under development include a meeting manager and capture application, contextual and natural language information retrieval, and a sketch interpretation system [19].

The Reconfigurable Ubiquitous Networked Embedded Systems (RUNES) project [22], funded by the EU Commission, envisions to enable the creation of large-scale, widely distributed, heterogeneous networked embedded systems that interoperate and adapt to their environments. The inherent complexity of such systems requires a standardised architecture allowing self-organisation to suit a changeable environment. To this purpose, RUNES aims to realize an adaptive middleware platform providing a common language that simplifies the application creation process. This should allow for a dramatic cut in the cost of new application development and a much faster time to market, transforming applications which are already technically possible into forms that are easy and straightforward for designers to use, and enabling applications which were previously unattainable. The project also examines the potential uses and implications of the technology, develop demonstrator systems and design training courses to aid in dissemination of RUNES technology. At this time, the theoretical framework proposed by RUNES is really convincing, but the middleware is almost uncomplete and has the strong limitation of being tailored for MANETs.

Another EU-funded AmI project is AMIGO [26], developing middleware that dynamically integrates heterogeneous systems to achieve interoperability between services and devices. For example, home appliances (heating systems, lighting systems, washing machines, refrigerators), multimedia players and renderers (that communicate by means of UPnP) and personal devices (mobile phones, PDAs) are connected in the home network to work in an interoperable way. This interoperability across different application domains can also be extended across different homes and locations. The project develops applications in different domains to show the potential for end-users and the benefits of the service oriented-middleware architecture for application developers. These applications are: "Home Care and Safety", "Home Information and Entertainment", and the "Extended Home Environment" in which multiple homes are connected. The enhanced service discovery and the semantic service composition methods addressed in Amigo are interesting and fall inline with requirements for composability and interoperability of services in an AmI environment. One drawback is the integration of input and output processing components into the middleware (UI service), while support for pluggability whould have been a better strategy.

Clearly AAL-oriented, the ASK-IT project [6] is driven by the vision of developing services that allow mobility impaired people to live more independently. By means of a mobile phone or PDA, users should have access to relevant and real-time information primarily for travelling but also whilst home, for work and leisure services. ASK-IT has not the objective to develop a standard ambient intelligence architecture. The emphasis is on a seamless service provision and a device that is intelligent enough to address the personal needs and preferences of the user. For example, information for a visually impaired person should be given orally, while for an illiterate person mostly in graphics.

Finally, project PERSONA [3], funded by the European Commission, aims at advancing the paradigm of Ambient Intelligence through the harmonisation of Ambient Assisted Living technologies and concepts for the development of sustainable and affordable solutions for the social inclusion and independent living of Senior Citizen, integrated in a common semantic framework. Project PERSONA seeks to develop a scalable open standard technological platform to build a broad range of AAL services, which are the services that the end user perceives as the final services and what he pays for. Each AAL service is composed by a hardware infrastructure and software components together with a user interface to communicate with the user. All accessible operations of software components provided with a usable interface are atomic services. Atomic services provided by the same component or by different components may be composed according to one or more patterns, resulting in composite services.

In conclusion, the lesson we learned is that AmI systems require a light and flexible middleware layer, providing facilities for building both client/server and peer-to-peer applications, supporting standard data and service descriptions, and allowing run-time component pluggability.

3.2 Ubiquitous peer-to-peer sharing of services

OSGi [18] is a Java-based technology which provides a service-oriented plug-in-based platform for application development. The core component of the OSGi Specifications is the OSGi Framework, which provides a standardized environment to applications (called bundles). On top of the Framework, services are specified by a Java interface. Bundles can implement this interface and register the service with the Service Registry. Clients of the service can find it in the registry, or react to it when it appears or disappears. Advanced networking features, such as *e.g.* peer-to-peer connectivity, are not provided by OSGi and must be implemented on top of it. For example, in the PERSONA platform [3] a middleware layer has been implemented (as a set of OSGi bundles), which hides the distribution and enables collaboration and communication, whether using a central registry or not.

To the best of our knowledge JXTA-SOAP is the sole open source project for P2P sharing of Web services being actively maintained and updated. WSPeer [10] is a J2SE toolkit for deploying and invoking Web Services in peer-to-peer Grid environments, which wraps Globus Toolkit core libraries to support the WS Resource Framework (WSRF) [31]. More interesting for ubiquitous computing

environments is the Mobile Web Services Mediation Framework (MWSMF) [23, 24], an adaptation of Apache ServiceMix, which is an open source ESB (Enterprise Service Bus). It provides an hybrid solution, since it must be configured as JXTA-J2SE peer and established as an intermediary between Web Service clients and mobile hosts, the latter being configured as JXME peers. Web Service clients can invoke the services deployed on mobile hosts via the MWSMF, which compresses SOAP messages (to BinXML format) and sends them through JXTA pipes. The MWSMF also manages message persistence, guaranteed delivery, failure handling and transaction support. Unfortunately, the source code is not publicly availble and few details are given about the realization of lightweight Web Service providers running on mobile hosts.

3.3 Web Services on resource-constrained devices

Besides hardware constraints, mobile devices introduce many other specific challenges which make difficult the deployment of Web Services on top of them [4]. Unlike dedicated servers, mobile devices will typically have intermittent connectivity to the network. As a result, the services offered on a mobile device may not be accessible all the time. An application that uses or composes such Web Services needs to operate in an opportunistic manner, leveraging such services when they become available. On the server side, Web Services on mobile devices should also attempt to keep messages as short as possible. Another issue to be addressed is the change of IP address which may arise when a mobile device moves between different locations, and from one administrative domain to another. However, with the P2P in place, the need for the Public IP can be eliminated and the mobiles can be addressed with unique peer ID. Each device in the P2P network is associated with the same peer ID, even though the peers can communicate with each other using the best of the many network interfaces supported by the devices like Ethernet, WiFi, etc. [23].

Since the WS message protocol, namely SOAP, introduces some significant overhead, few toolkits support the deployment of Web Services on limited devices, such as PDAs, smart phones, etc. One is gSoap [27], which provides a WS engine with run-time call de-serialization. Unfortunately, gSoap is written in C/C++, thus requiring a priori stub/skeleton generation by means of a specific compiler, which also means lack of portability.

.NET Compact Framework [17] is a subset of the .NET platform, targeting mobile devices. Its class library enables the development of Web Service clients, but does not allow to host Web Services.

Looking at the Java Micro Edition (J2ME) platform, most libraries are only for client side functionality. The Java Wireless Toolkit (WTK) provides J2ME Web Services API (WSA) [29], based on JSR 172 [13], which specifies runtime ServiceProvider interface to allow the generation of portable stubs from WSDL files. The specification contains some notable limitations, most of them due to the requirement for WS-I Basic Profile compliance. Conforming to the profile ensures interoperability, but also prevents using alternative methods. Another widely used solution is the kSoap2 [15] open source component, which is a parser

for SOAP messages (with RPC/literal or document/literal style encoding), not supporting the generation of client side stubs. kSoap2 is compliant with devices lacking JSR 172 support, and allows to access non WS-I conformant services.

To the best of our knowledge, the unique solution enabling J2ME applications (CLDC, CDC) as service endpoints is the Micro Application Server (mAS) [16]. It can be considered a lightweight version of Axis, by which it is inspired. For this reason we have chosen it to implement the J2ME version of JXTA-SOAP.

4 Internal architecture of the JXTA-SOAP component

The JXTA-SOAP component extends JXTA which is a set of open, generalized peer-to-peer protocols that allow a vast class of networked devices (smartphones, PDAs, PCs and servers) to communicate and collaborate seamlessly in a highly decentralized fashion. JXTA-SOAP has been designed having in mind ubiquitous computing needs, such as those defined by the SCP and SEP paradigms [3]. JXTA-SOAP completes the JXTA framework (which defines a naming scheme, advertisements, peergroups, pipes, and a number of core policies) with service supporting mechanisms based on state-of-art software design patterns, with the purpose to reduce the complexity otherwise required to build and deploy peer-to-peer service-oriented applications.

In this section we shortly describe the internal architecture of JXTA-SOAP, with the purpose of conceptualizing its main features at a high abstraction level. Technical details are out of the scope of this paper, but the interested reader may refer to [2].

4.1 Service deployment

In order to deploy its services, a JXTA-SOAP based peer has to instantiate and configure the related service objects (one for each hosted service), and to advertise the service interfaces in the network. The diagram in figure 2 illustrates the relationships among objects which are involved in this tasks. The Peer class represents the generic peer application implemented by the developer, and relies on JXTA-SOAP's API which provides all the other classes illustrated in the diagram.

At start-up, each peer bootstraps the JXTA platform, configuring basic connectivity settings, such as TCP/IP and HTTP ports. Bridges to existing common membership and access technologies allow the peer to establish its identity within peergroups (the main one, *i.e.* JXTA public peergroup, and its subgroups). These tasks are illustrated by the sequence diagram in figure 3, and detailed in the following of this section.

For each service to be deployed, a service descriptor must be istantiated and filled with service-specific information, such as the service name, a brief description of the functionalities the service provides, the implementation class name, the peergroup ID, the security tag. Moreover, the service descriptor accepts a

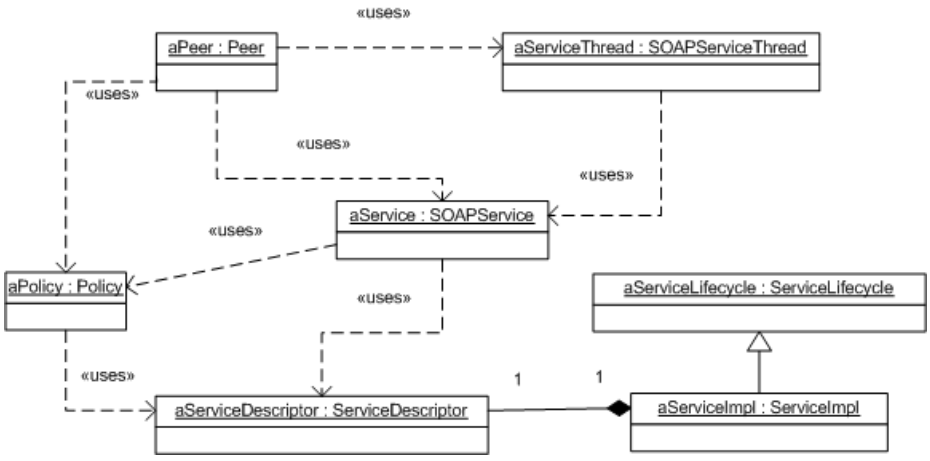

Fig. 2. Objects involved in service deployment.

Context Object [14] (constructed by the user application) to the Web Service class.

Then, the service advertisement (*i.e.* a XML document describing a resource, in JXTA jargon) must be filled with the service WSDL. Next step is to create the context object, whose parameters are stored in a hashmap, and to pass it to the service instance, along with the associated security policy.

Moreover, the initialization of the service triggers the creation and publication of the advertisement of a public pipe for the invocation of the service itself. For each service implementation, the peer spawns a thread which waits for consumers' connections to that service, on the service public pipe, and creates a pool of invocation threads to serve requests concurrently, according to the *Threadpool* pattern [20]).

4.2 Service publication and lookup

The main enhancement of JXTA-SOAP with respect to traditional Web Service frameworks is the adopted distributed approach for service advertising and lookup. JXTA-SOAP allows to encapsulate WSDL interfaces in specific JXTA advertisements, which can be spread into the network using one of the routing policies which can be inserted in JXTA protocol stack.

Service publication is a distributed process, which uses network nodes as a distributed repository (on the contrary, traditional UDDI registries are centralized interface description repositories). As for publication, service lookup is a distributed process, which can be conceptualized as message exchange between low-level JXTA modules.

JXTA's default message routing protocol for advertisement sharing and discovery is called SRDI. Its description is out of the scope of this paper, thus we suggest the interested reader to read the paper of Traversat *et al.* [25] for details.

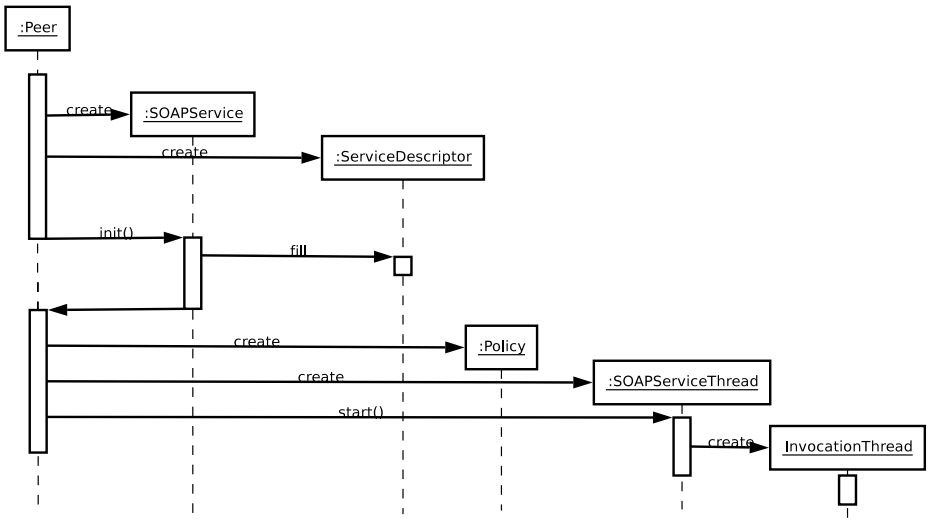

Fig. 3. The service deployment process represented with a sequence diagram.

4.3 Service invocation

Figure 4 illustrates associations among objects which are involved in the service invocation task, performed by a generic peer. The latter, once it has discovered the service advertisement and the WSDL interface of the Web Service, creates a SOAP transport deployer, which manages the transmission of SOAP messages to and from the service using its pipe. Moreover, the peer creates a service descriptor which is used by the call factory to instantiate a call (the adopted strategy is the *Factory Method* [9]). Each call implements the *Requestor* pattern [28], which constructs a message from the absolute reference of the remote service, the operation name, its arguments and return type.

5 Implementation of the JXTA-SOAP component

We implemented two (interoperable) versions of JXTA-SOAP: J2SE-based, extending JXTA-J2SE, and J2ME-based, extending JXTA-J2ME. In the following we describe their features and the different technological solutions they rely on.

5.1 JXTA-SOAP for Java Standard Edition (J2SE)

The J2SE version of the JXTA-SOAP component supports service deployment, discovery, and invocation, with optional use of standard mechanisms to secure communications among peers. The core of the component is the Apache Axis engine (v1.4), which is a producer/consumer of SOAP messages. Usually Axis is deployed in a Web application server, such as Apache Tomcat, together with

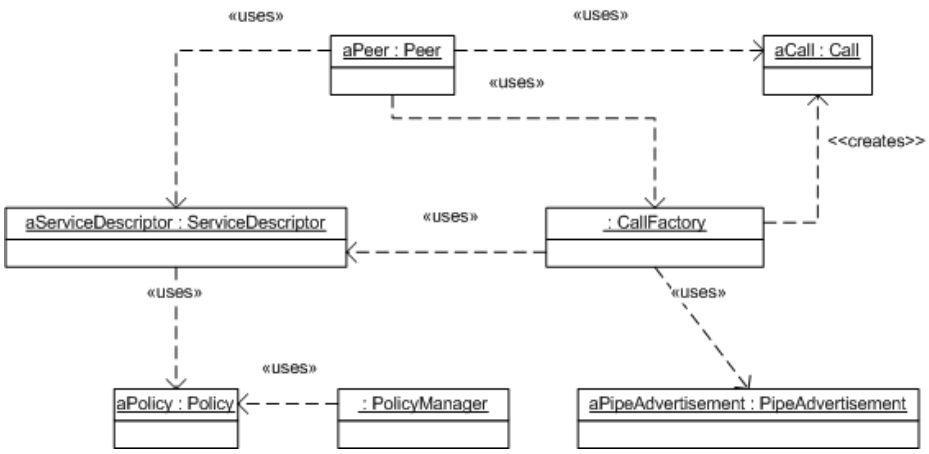

Fig. 4. Objects involved in service invocation.

the implementions of the Web Services to be deployed, while client applications use the Axis Java API to create request instances. The Axis engine provides the processing logic, either client or server. When running, it invokes a series of Handlers according to a specific order which is determined by two factors - deployment configuration, and whether the engine is a client or a server. The object which is passed to each Handler invocation is a MessageContext, *i.e.* is a structure which contains several important parts: 1) a "request" message, 2) a "response" message, and 3) a bag of properties.

At runtime, for a service provider its service objects are deployed in the Axis engine, which implements the JAX-RPC API, one of the standard ways to program Java services, also supporting the lifecycle of service endpoint instances. After being loaded and instantiated, the JAX-RPC runtime system is required to initialize the service instance before any requests can be serviced. A context parameter is pssed to the initialization function, enabling the service instance to access the context provided by the underlying JXTA-SOAP based runtime system. The context parameter is typecasted to an appropriate Java type. For services deployed in a JXTA-SOAP based runtime system, the Java type of the context parameter is defined by the developer that is using the JXTA-SOAP API, and passed to the service object. The latter instantiates the Service Descriptor, creates and publishes the public pipe and the service advertisement, and notifies itself to the Axis engine. Services can be deployed anytime, without the need to restart the peer.

Once a service instance has been initialized, the Axis engine may dispatch multiple remote invocations to it. After that, when the Axis engine determines that the service instance needs to be removed from service of handling remote invocations, it destroys it. In the implementation of the destruction functionality, the service object releases its resources.

For remote service invocation, a consumer peer needs to intantiate a Call object. JXTA-SOAP's Call class extends Axis' default one, overloading the use of service URLs with the use of the Service Descriptor and the public pipe advertisement of the service. To create Call instances, the peer uses the implementation of the Call Factory class provided by Axis.

We previously described the tasks which are performed when a Web Service is deployed by a peer, and we mentioned that some parameters are put in the Service Descriptor for further use by the Axis engine. In particular, one of these parameters is the Web Service Deployment Descriptor (WSDD). When the WSDD is sent to the Axis engine running in the peer, an *Invoker* [28] is informed that it supports the new Web Service. Thus, when an invocation reaches the peer, the Invoker looks up the class which implements the service, and lets the instance handle the request message. In details, the Invoker reads incoming messages and demarshals the parameters inserted by the consumer peer's Requestor (absolute reference of the service, operation name, arguments, return value) and dispatches the message to the targeted service.

5.2 JXTA-SOAP for Java Micro Edition (J2ME)

The J2ME version of the architecture illustrated in section 4 supports Connected Device Configuration (CDC) and Personal Profile. We implemented the API which enables the development of peers that are able to deploy, provide, discover and consume Web Services in a JXTA-SOAP network. Since Axis is not available for the CDC platform, we adopted kSoap2 [15] as SOAP parser (for consumer functionalities) and, for service provision, we integrated the mAS [16] lightweight engine.

Service invocation is allowed by a kSoap2 based implementation of the Call Factory class. The latter instantiates a kSoap2's Soap Object, and sets all the properties for message exchanging through JXTA pipes. Soap Object is a highly generic class which allows to build SOAP calls, by setting up a SOAP envelope. We have maintained the same structure of J2SE-based version for Call Factory, to allow portability of service consumer applications from desktop PCs or laptops to PDAs. Internally, the Call Factory class creates a Soap Object passing references to the Service Descriptor, the public pipe advertisement of the service and the peergroup as parameters for the creation of the Call object.

The Call Factory class also allows to create an instance of kSoap Pipe Transport, the class we implemented to manage the transmission of SOAP messages using service pipes. The kSoap2 API provides a Transport class that encapsulates the serialization and deserialization of SOAP messages, but does not manage communication with the service; the HTTP Transport subclass, both in CDC and CLDC version, allows service invocation over HTTP, setting up the required properties, but it uses URLs as absolute references of remote services, and it is not suitable for usage in JXTA-SOAP, where services (as every resource) are identified by JXTA-IDs and must be invoked through JXTA pipes. Thus, we extended the Transport class with the implementation of a call functionality that configures a JXTA pipe and creates the messages to be sent over it.

After instantiating the transport using the Call Factory class, the consumer peer creates the request object, indicating the name of the remote method to invoke and setting the input parameters as additional properties. This object is assigned to a Soap Serialization Envelope, as the outbound message for the soap call; Soap Serialization Envelope is a kSoap2 class that extends the basic Soap Envelope, providing support for the SOAP Serialization format specification and simple object serialization. The same class provides a getResponse method that extracts the parsed response from the wrapper object and returns it.

Referring to service provision, we integrated the Server class of the Micro Application Server (mAS) into the basic service class of the JXTA-SOAP API.

mAS implements the Chain of Responsibility pattern [9], the same used in Axis. It avoids coupling the sender of a request to its receiver by giving more than one object a chance to handle the request; receiving objects are chained and the request passed along the chain until an object handles it. Moreover, mAS allows service invocation by users and service deployment by administrator; it also supplies browser management of requests, distinguishing if the HTTP message contains a Web page request or a SOAP envelope.

6 Experimental evaluation

Using a simple, ping-pong like Web Service, we tested several point-to-point configurations, combining different settings for each participant. JXTA-J2SE peers have been deployed on laptops and desktop computers running either Windows XP, Linux or Mac OS X, equipped with 1GB RAM and 1.6GHz processors. A JXTA-J2ME peer has been hosted on an I-Mate JASJAR Pocket PC PDA, equipped with 64MB RAM and 520MHz processor.

Table 1. JXTA-SOAP test configurations

overlay peers	data link	multicast	t_r (s)	t_s (s)	t_i (s)
edge J2SE c, rdv J2SE p	Ethernet	off	2.5	0.5	0.1
edge J2SE c, rdv J2SE p	Ethernet	on	n.n.	2.0	0.1
edge J2SE c, edge J2SE p	Ethernet	on	n.n.	0.5	0.1
edge J2SE c, edge J2SE p, rdv J2SE b	Ethernet	off	2.5	0.5	0.1
edge J2SE c, rdv J2SE p	WiFi	off	2.0	0.5	0.7
edge J2SE c, rdv J2SE p	WiFi	on	n.n.	2.0	0.7
edge J2SE c, edge J2SE p	WiFi	on	n.n.	1.0	0.4
edge J2ME c, rdv J2SE p	WiFi	on	n.n.	3.1	2.0
edge J2ME c, edge J2ME p	WiFi	on	n.n.	2.5	2.0
adhoc J2SE c, adhoc J2SE p	adhoc	on	n.n.	1.0	0.4
edge J2ME c, rdv J2SE p	adhoc	on	n.n.	1.0	4.4
adhoc J2ME c, adhoc J2SE p	adhoc	on	n.n.	1.0	0.4
adhoc J2ME c, adhoc J2ME p	adhoc	on	n.n.	1.0	0.5

The memory footprint of a Java program is predominantly due to objects, classes, and threads that the users create directly, and native data structures (like the constant-pool, the string-table, etc.), native code, and the virtual machine (JVM) itself that are loaded indirectly by the user. For JXTA-SOAP peers running on laptops and desktop conputers with J2SE v1.5, we measured a 22.5MB footprint (at least 10MB are needed by the sole JVM). On the other side, the peer installed on the Pocket PC with J2ME (personal profile v1.1) had a 7MB RAM footprint (with 3MB for the JVM).

All tested configurations are listed in table 1. At the overlay network level, *i.e.* JXTA level, a peer may be configured in one of the following modes:

- rendezvous supernode (shortly, rdv) - routes messages using the JXTA-SRDI strategy [25]
- relay supernode (relay) - provides message relaying services, enabling cross firewall traversal
- ad-hoc node (adhoc) - such peer will not use any infrastructure peers (rendezvous or relay), but rely on ad-hoc multicast to discover other peers to connect with
- edge node (edge) - in addition to supporting the Ad-Hoc behavior, an Edge node can attach to an infrastructure peer (a Rendezvous, Relay, or both)

In our testbed, we did not use relays and we configured peers as service providers (p), consumers (c), or bridge nodes (b) which store advertisments and route messages but do not provide or consume Web Services. At the data link layer we considered Ethernet, WiFi and ad-hoc mode.

Experimental results refer to the following sequential actions performed by the consumer peer:

- elapsed time for rendezvous peer discovery (t_r)
- elapsed time for service discovery (t_s)
- elapsed time for service invocation (t_i)

Rendezvous peer discovery is not necessary (n.n.) when multicast is active (on), but service discovery requires much time with respect to the multicast off case. Without multicast, a list of rendezvous hosts must be used to allow peers join the network at bootstrap. Once an edge peer is connected to its rendezvous, if the service has been advertised (and replicated among rendezvous peers) the discovery process is very fast.

Test results are encouraging, being performance significant in almost all examined cases. It appears that, when hosts are connected in ad-hoc mode, best performance is achieved if also at the application level peers are configured in ad-hoc mode.

7 Conclusions and future work

To design ubiquitous and pervasive applications, the traditional client/server approach is being superseded by new emerging paradigms, such as the Adaptive Services/Client Paradigm (SCP) and the Spontaneous Service Emergence

Paradigm (SEP), based on a peer-to-peer approach. JXTA middleware is a viable solution to implement such architectures. In this paper we presented the JXTA-SOAP component, which enables Web Service deployment in JXTA peers, as well as distributed WSDL publication and discovery, and SOAP message transport over JXTA. Particularly, the JXTA-J2ME implementation enables Web Service invocation from mobile platforms in a JXTA P2P Network.

We proposed JXTA-SOAP as a powerful solution for building service-oriented, peer-to-peer ubiquitous platforms, for which Ambient Intelligence is a natural application field.

Future work on JXTA-SOAP will mainly focus on supporting the Web Service Resource Framework [31], in order to provide peers the ability to access and manipulate state, *i.e.* data values that persist across, and evolve as a result of, Web Service interactions. This is particularly important for AAL services like User Activity Monitoring, which requires to collect contextual data but also historical information from services dispersed over the network.

References

1. MIT Computer Science and Artificial Intelligence Laboratory, *AIRE Group*, http://aire.csail.mit.edu/index.shtml
2. M. Amoretti, M. Bisi, F. Zanichelli, G. Conte, *Enabling Peer-to-Peer Web Service Architectures with JXTA-SOAP*, IADIS International Conference e-Society 2008, Algarve, Portugal, April 2008.
3. E. Avatangelou, R. F. Dommarco, M. Klein, S. Muller, C. F. Nielsen, M. P. S. Soriano, A. Schmidt, M.-R. Tazari, R. Vichert, *Conjoint PERSONA-SOPRANO Workshop*, Proc. of the first European Conference on Ambient Intelligence (AmI-07), Darmstadt, Germany, November 2007.
4. S. Berger, S. McFaddin, C. Narayaswami, M. Raghunath, *Web Services on Mobile Devices - Implementation and Experience*, Proc. of the Fifth IEEE Workshop on Mobile Computing Systems & Applications, Monterey, CA, USA, October 2003.
5. T. Bodhuin, G. Canfora, R. Preziosi and M. Tortorella, *Open Challenges in Ubiquitous and Net-Centric Computing Middleware*, 13th IEEE International Workshop on Software Technology and Engineering Practice, September 2005.
6. S. Edwards, *User Driven and Seamless Mobility Services for Disabled and Older People: the ASK-IT Project*, Proc. of the 5th Annual Moving On Conference, Glasgow, March 2006.
7. J. Gaber, *Spontaneous Emergence Model for Pervasive Environments*, IEEE Globecom Workshop 07, Washington DC, November 2007.
8. L. Z. Granville and A. Panisson, *GigaMAN P2P project*, http://gigamanp2p.inf.ufrgs.br
9. Gamma E, Helm R, Johnson R, Vlissides J. *Design Patterns.* Addison-Wesley, 1995.
10. A. Harrison, I. Taylor, WSPeer - An Interface to Web Service Hosting and Invocation, *Proc. of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS05)*, Denver, Colorado, USA, May 2005.
11. IST Advisory Group, *Scenarios for Ambient Intelligence in 2010*, European Commission, 2001.
12. Distributed Systems Group and Sun MicroSystems, *JXTA-SOAP project*, https://soap.dev.java.net

13. Sun MicroSystems, *JSR 172: J2ME Web Services Specification*, http://jcp.org/en/jsr/detail?id=172
14. Krishna A., Schmidt D. C., Stal M., Context Object: A Design Pattern for Efficient Middleware Request Processing. *Proc. of the 12th Pattern Language of Programming Conference*, Allerton Park, Illinois, September 2005.
15. S. Haustein, J. Seigel, *kSoap2 project*, http://ksoap2.sourceforge.net
16. P. Plebani, *mAS project*, https://sourceforge.net/projects/masproject
17. Microsoft, *.NET Compact Framework* http://msdn.microsoft.com/en-us/netframework/aa497273.aspx
18. OSGi Alliance, *OSGi: the Dynamic Module System for Java*, http://www.osgi.org
19. S. Peters, H. Shrobe, *Using Semantic Networks for Knowledge Representation in an Intelligent Environment*, Proc. of 1st Annual IEEE International Conference on Pervasive Computing and Communications (PerCom '03). Ft. Worth, TX, USA, March 2003.
20. I. Pyarali, M. Spivak, R. Cytron, D. C. Schmidt, *Evaluating and Optimizing Thread Pool Strategies for Real-Time CORBA*, Proc. of the ACM SIGPLAN Workshop on Optimization of Middleware and Distributed Systems (OM 2001), Snowbird, Utah, USA, June 2001.
21. C. Ramos, J. C. Augusto, D. Shapiro, *Ambient Intelligence - the Next Step for Artificial Intelligence*, IEEE Intelligent Systems, Vol. 23, No. 2, March/April 2008.
22. P. Costa, G. Coulson, C. Mascolo, L. Motolla, G. P. Picco, S. Zachariadis *A Reconfigurable Component-Based Middleware for Networked Embedded Systems*, International Journal of Wireless Information Networks, Springer, 2006.
23. S. N. Srirama, M. Jarke and W. Prinz, *A Mediation Framework for Mobile Web Service Provisioning*, Proc. of the 10th IEEE International Enterprise Distributed Object Computing Conference Workshops (EDOCW'06), Hong Kong, China, October 2006.
24. S. N. Srirama, M. Jarke and W. Prinz, *MWSMF: a Mediation Framework Realizing Scalable Mobile Web Service*, Proc. of Mobilware'08, Innsbruck, Austria, February 2008.
25. B. Traversat, A. Arora, M. Abdelaziz, M. Duigou, C. Haywood, J.-C. Hugly, E. Poyoul, B. Yeager, Project JXTA 2.0 Super-Peer Virtual Network, *Technical Report*, Sun Microsystems, 2003.
26. M. Vallee, F. Ramparany, L. Vercouter, *A multi-agent system for dynamic service composition in ambient intelligence environments*, Proc. of the Third International Conference on Pervasive Computing (Pervasive 2005), Munich, Germany, May 8-11, 2005.
27. R. A. van Engelen and K. Gallivan, *The gSOAP Toolkit for Web Services and Peer-To-Peer Computing Networks*, Proc. of the 2nd IEEE International Symposium on Cluster Computing and the Grid (CCGrid2002), pp.128-135, Berlin, Germany, May 2002.
28. M. Volter, M. Kircher, U. Zdun, *Remoting Patterns*, Wiley, 2005.
29. Sun MicroSystems, *J2ME Web Services APIs (WSA)*, http://java.sun.com/products/wsa/
30. Banaei-Kashani F, Chen C, Shahabi C. WSPDS Web Services Peer-to-peer Discovery Service. *The 2004 International Symposium on Web Services and Applications*, Las Vegas, Nevada, USA, June 2004.
31. OASIS, *Web Services Resource Framework (WSRF) v1.2*, April 2006.