ubiSOAP: A Service Oriented Middleware for Ubiquitous Networking

Mauro Caporuscio, Pierre-Guillaume Raverdy, Valérie Issarny

*ubi*SOAP: A Service Oriented Middleware for Ubiquitous Networking

Mauro Caporuscio, Pierre-Guillaume Raverdy and Valerie Issarny

Abstract—The computing and networking capacities of today's wireless portable devices allow for ubiquitous services, which are seamlessly networked. Indeed, wireless handheld devices now embed the necessary resources to act as both service clients and providers. However, the ubiquitous networking of services remains challenged by the inherent mobility and resource constraints of the devices, which make services a priori highly volatile. This paper discusses the design, implementation and experimentation of the *ubi*SOAP service-oriented middleware, which leverages wireless networking capacities to effectively enable the ubiquitous networking of services. *ubi*SOAP specifically defines a layered communication middleware that underlies standard SOAP-based middleware, hence supporting legacy Web Services while exploiting nowadays ubiquitous connectivity.

Index Terms—Service Architecture, Advanced Services Innovation Framework, Service Delivery Platform.

1 Introduction

With network connectivity being embedded in most computing devices, any networked device may seamlessly consume but also provide software applications over the network. Service-Oriented Computing (SOC) then introduces natural design abstractions to deal with ubiquitous networking environments [1]. Indeed, networked software applications may conveniently be abstracted as autonomous loosely coupled services, which may be combined to accomplish complex tasks. In addition, the concrete instantiation of SOC paradigms provided by Web Services (WS) technologies by means of Web-based/XML-based open standards (e.g. WSDL, UDDI, HTTP, SOAP) may be exploited for concrete implementation of ubiquitous services.

However, while Web services standards and implementations targeting wide-area domains are effective technologies, supporting Web service access in ubiquitous networking environments is still challenging. In fact, in such kind of networking environments both service consumers and providers often run on resource-scarce platforms (e.g., personal digital assistants and mobile phones), which have limited CPU power, memory, and battery life. Moreover, these devices are usually interconnected through one or more heterogeneous wireless links, which compared to wired networks are characterized by lower bandwidth, higher error rates, and frequent disconnections. The former issue has led to the introduction of lightweight middleware enabling base WS-oriented communication patterns among wireless portable devices (i.e., SOAP-based messaging and dynamic service discovery) [2], [3]. The latter issue has further led to examine specific SOAP transports [4]. However, a key feature of ubiquitous networking environments is the diversity of radio links available on portable devices, which may be exploited towards ubiquitous connectivity. Specifically, as nodes get directly connected via multiple radio links, thorough scheduling and handover across those links allow enhancing overall connectivity and actually making it ubiquitous [5], [6]. This calls for making services network-agnostic [7], so that the underlying middleware takes care of scheduling exchanged messages over the embedded links in a way that best matches Quality of Service (QoS) requirements [8], and further ensures service continuity through vertical handover [9]. In this setting, a primary requirement for supporting service-oriented middleware is to provide a comprehensive networking abstraction that allows applications to be unaware of the actual underlying networks while exploiting their diversities in terms of both functional and extra-functional properties.

This paper introduces the *ubi*SOAP middleware, which strives to provide ubiquitous networking to services. Specifically, *ubi*SOAP defines a two-layer architecture which respectively provide *network-agnostic connectivity* and *WS-oriented communication* in ubiquitous networking environments.

The design rationale for *ubi*SOAP is discussed in the next section. Sections 3 and 4 then detail the core functionalities of *ubi*SOAP, namely *network-agnostic connectivity* and *SOAP communication*, while Section 5 presents a service-discovery service for ubiquitous environments. The assessment of *ubi*SOAP is carried out in both Sections 6, which evaluates *ubi*SOAP performance, and in Section 7, which shows a set of service-oriented applications leveraging *ubi*SOAP. Finally, Section 8 summarizes our contribution with respect to related work, and Section 9 sketches our perspectives for future work.

M. Caporuscio is with Dipartimento di Elettronica ed Informazione, Politecnico di Milano, Italy – E-mail: caporuscio@elet.polimi.it.
P-G. Raverdy and V. Issarny are with INRIA, France – E-mail: FirstName.LastName@inria.fr

2 *ubi*SOAP Design

With the drastic evolution of wireless technologies, software services can become truly ubiquitous, being not solely accessed but also hosted by wirelessly networked portable devices. Still, enabling ubiquitous service provisioning on mobile hosts requires special care, as resources are far more constrained than resource-rich Internet servers originally targeted by service-oriented computing and its Web Service instantiation. Further, the mobility of wireless hosts requires special attention.

Portable devices now embed multiple radio interfaces that may be combined to bring ubiquitous networking to mobile applications [7]. Specifically, the scheduling of communications over embedded interfaces according to application requirements can significantly increase the overall QoS. In particular, saving energy is critical for enhanced autonomy of hosts and thus requires selecting as far as possible the network interface that consumes the least energy among those eligible [8].

The *ubi*SOAP communication middleware aims at effectively exploiting the diverse network technologies at once in order to create an integrated multi-radio networking environment, hence offering *network-agnostic connectivity* to services. This requires addressing a number of critical issues such as *network availability*, *user and application QoS requirements* and *vertical handover*. *Vertical handover* [10] is particularly important with respect to the *service continuity* requirement. Indeed, when a host changes its point of attachment (vertical handover between two networks), the IP address is modified accordingly in order to route packets to the new network. Hence, since the IP address is the base of any application-layer connection, all the ongoing connections break. Moreover, as devices can bind various networks at the same time, two interacting parties might communicate through multiple paths. Hence, choosing the best connection to serve a given interaction is a key issue to deal with in ubiquitous networks, as this significantly affects the QoS at large (e.g., availability, performance with respect to both resource consumption and response time, security) [11].

Another of our goals for *ubi*SOAP is to support legacy Web services and thus transparently bring the added value of today's ubiquitous networking environments to existing services. This has in particular led us to layer *ubi*SOAP as a specific *Point-to-Point transport* for SOAP engines (e.g., Axis2[1], CSOAP[2]) and to leverage WS-addressing to integrate multi-radio, multi-network connectivity in SOAP headers. In this context, it is crucial to examine carefully the performance of SOAP transports. In particular, it has been shown that the performance of default SOAP over HTTP is poor in wireless environments, further leading to study alternative transports such as TCP [12] and UDP [4]. While SOAP over UDP clearly offers the best response time, SOAP over TCP has the advantage of built-in reliability and is suitable for applications with short requests. *ubi*SOAP thus realizes SOAP-over-TCP unicast messaging as a tradeoffs solution, while integrating SOAP over UDP is an area for future work. Still, another SOAP transport that is of much interest for ubiquitous networking environments is *group transport*. Indeed, group-based interactions are central in a number of ubiquitous computing scenarios, due to the user-centric nature of ubiquitous computing and the innate group interaction skills of people [13], [14]. *ubi*SOAP thus features a base SOAP transport for group communication.

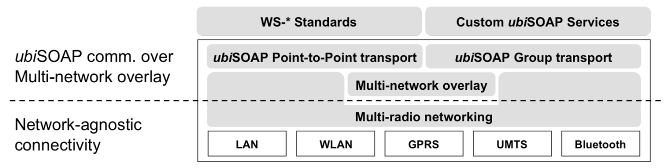


Fig. 1. *ubi*SOAP software architecture

Following the above discussion, the *ubi*SOAP architecture (see Figure 1) exploits a two-layer design where each layer deals with a specific issue. Specifically, (i) *network-agnostic connectivity* deals with network heterogeneity and provides multi-radio networking (see Section 3) and (ii) *ubi*SOAP *communication* implements a multi-network overlay achieving both *ubi*SOAP point-to-point and group transports among nodes in ubiquitous networks (see Section 4).

3 Network-agnostic Connectivity

The *ubi*SOAP *network-agnostic connectivity* layer provides Multi-Radio Networking (MRN) functionality by means of two entities (see Figure 2): (i) Multi-Radio Networking Daemon (*MRN-Daemon*) is the main entity implementing all the provided features, and (ii) Multi-Radio Networking API (*MRN-Api*) allows for an easy and transparent access to the functionalities offered by *MRN-Daemon*. On top of this layer, a *ubi*LET is any entity (e.g., application) that exploits the *network-agnostic connectivity* layer by accessing the functionalities provided by *MRN-Daemon* through *MRN-Api*.

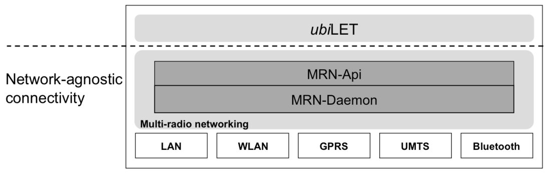


Fig. 2. Network-agnostic connectivity layer

Specifically, the *network-agnostic connectivity* layer offers the core functionalities to effectively manage the underlying multi-radio environment through: (i) a network-agnostic addressing scheme together with (ii) QoS-aware network link selection and (iii) base unicast and multicast communication schemes.

1. http://ws.apache.org/axis2/
2. http://www-rocq.inria.fr/arles/download/ozone/

3.1 Network-agnostic addressing

Devices embedding multiple network interfaces may have multiple IP addresses, at least one for each active interface. Thus, in order to identify uniquely a given *ubi*LET in the network we associate it with a *Multi-Radio Networking Address* (MRN@). The MRN@ of a *ubi*LET instance is specifically the application's Unique ID, which maps into the actual set of IP addresses (precisely, $network\_ID \oplus IP$ addresses) bound to the device (at a given time) that runs the given instance. Referring to Figure 3, the MRN@ associated to the *ubi*LET$_j$ running on Alice's device is:

$$MRN@_{ubiLET_j} \mapsto [net_a \oplus IP_{a_1}, net_i \oplus IP_{i_1}, net_n \oplus IP_{n_1}]$$

where $\forall j \in [1,2]$, MRN@$_{ubiLET_j}$ is the ID of *ubi*LET$_j$ and, $[net_a \oplus IP_{a_1}, net_i \oplus IP_{i_1}, net_n \oplus IP_{n_1}]$ is the set of $network\_ID \oplus IP$ addresses denoting the actual location of the device[3].

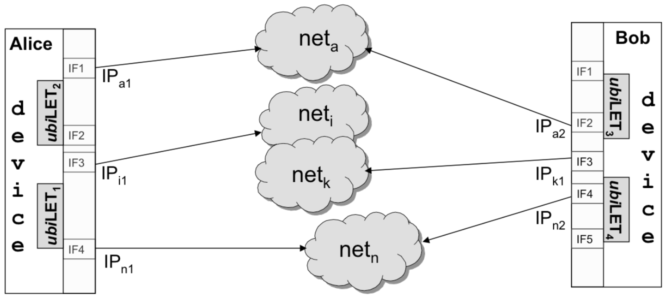


Fig. 3. Network-agnostic addressing

In order to assure that each MRN@ is unique, we generate MRN@ following the Universally Unique Identifier (UUID) standard [15]. In particular, the Name-Based UUID algorithm let users choose their own mnemonic "name". Then, given an application name we concatenate it to one of the network radio adapter IDs embedded in the device (e.g., MAC address, Bluetooth ID, UMTS IMEI), which are uniquely assigned by the manufacturer. Once the MRN@ has been assigned to the *ubi*LET, upper layers shall use it as part of their addressing scheme (e.g., through WS-addressing in the case of Web services), which replaces the traditional IP-based addressing scheme.

*ubi*SOAP *network-agnostic connectivity* layer provides the following operations to generate and manage MRN@s:

Registration – allows a *ubi*LET to register itself within the *network-agnostic connectivity* layer and generates the MRN@ that uniquely identifies it. As explained above, the subscribing *ubi*LET supplies a locally unique identifier, which in turn is concatenated to one of the network radio adapter IDs to generate the MRN@.

Management – is in charge of managing at runtime the mapping between the MRN@ and the actual set of IP-Addresses. When a device moves from a point of attachment to a new one (e.g., it detaches from a network and reconnects to a new one), such a mapping changes as result of the vertical-handover procedure. Hence all the ongoing communications (both outgoing and incoming) might break. If a moving device hosts only service consumers, then all the communications are outgoing and can be managed locally by simply switching the associated connections to the new accessed network. On the other hand, when the device is hosting service providers, and then the communications are incoming, it must inform all the remote parties involved in the communication about the mapping change. The new mapping is multicasted to all the networks currently in use in order to reach as many as possible remote parties. For each remote party, if there still exists a direct connection (e.g., a shared network) between the service provider and the consumer, the latter receives the new mapping and is able to continue the communication without any interruption. On the contrary, if the two parties do not share any network, the *network-agnostic connectivity* layer is no longer able to manage the communication and then the upper layer, which is in charge of routing packets through multi-network overlay (see Section 4), is notified about the problem occurrence.

Lookup – allows *ubi*LETs to retrieve the actual set of IP addresses related to a given MRN@. Exploiting the Ethernet Address Resolution Protocol (ARP) idea [16], if the resolution of MRN@ is not cached or needs to be updated, a request is multicasted to all the networks currently accessible and, if the device bound to the given MRN@ is reached, it will directly reply to the requester by supplying the MRN@ mapping.

3.2 QoS-aware network link selection

Next to MRN@ addressing, it is crucial to activate and select the best possible networks (among those available) with respect to required QoS. Specifically, QoS is defined by means of a set of pairs $< QoS_{attr}, QoS_{value} >$ where attributes are grouped in two subsets: (i) *quantitative* attributes that describe the performance provided by the networks – i.e., bitrate, packet loss transfer delay and signal strength – and allows for networks ranking, and (ii) *qualitative* attributes that describe those characteristics of the network that do not affect the network performance but should be considered – i.e., power consumption, price, coverage area.

Each network sensed by the *network-agnostic connectivity* layer has associated a *QoSInfo* profile that represents its quantitative and qualitative QoS.Due to the heterogeneity of the QoS attributes considered, they are normalized and abstracted as classes of values: `NULL`, `LOW`, `MID`, `HIGH`, `VERYHIGH`. Attributes not explicitly defined are set to `NULL` and are not taken into account during the link selection. Furthermore, it is possible to define priorities upon the quantitative attributes (if not `NULL`) in order to specify their relevance with respect to

3. For the sake of simplicity we refer to IP address, but it is actually implemented as IP address and port number, e.g., `128.131.10.1:90`.

the others. This is achieved by means of a special set of weights $W = \{w_i \in [0,1] | \sum_i w_i = 1\}$.

Given a set of networks, in order to rank them with respect to QoS requirements, we evaluate their *QoSInfo* profiles by obtaining a set of normalized values and then apply the `evaluate` function that, given a *QoSInfo* describing the network QoS and a set W of weights w, returns a value in $[0, 1]$, which represents the normalized value of the QoS.

To rank networks with respect to the end-user's QoS requirements, we define a `matchRequirements` function that, given two *QoSInfo* profiles specifying QoS requirements and network QoS respectively, returns a value in $[0, 1]$ representing the *matching accuracy*. Indeed, the *matching accuracy* measures how much the network QoS meets the user QoS needs. Such a value is then used to rank networks with respect to the user needs.

The two above base operations are used to build the following functionalities:

Interface activation – allows *ubi*LETs to activate the best possible interfaces (among those available) with respect to the required QoS. Specifically, the *interface activation algorithm* (see Figure 4) compares the QoS requirement (specified by the application as a *QoSInfo* profile) with the *QoSInfo* of each available interface in *ints*. If an interface satisfies the requirement, within a given approximation expressed in percentage (i.e., *matching accuracy*), then it is activated. Note that, since the interfaces are switched off, QoS refers to the theoretic values of a network interface as declared by the manufacturer (e.g., GPRS maximum bitrate = 171.2Kb/s).

```
activateInterface(List ints,QoSInfo req,real accuracy){
  for(i = ints.getNext(); ints!= empty){
        if (matchRequirements(req, i.qos) >= accuracy)
            i.activate() ;
  }
}
```

Fig. 4. Interface activation algorithm

Network selection – is performed during the establishment of the communication and takes into account both the QoS required by the *ubi*LET that is instantiating the communication and the set of networks active on the remote *ubi*LET (as given by the destination's MRN@). Specifically, the *network selection algorithm* (see Figure 5) first selects all the networks (among the available ones) that both satisfy the QoS requirements (*req*) and allow for reaching the destination (*dest*), and then sorts in descending-order the resulting set of networks with respect to the *matching accuracy*. Hence, since networks are arranged in a descending-order list, the first of the list is the one that best fits the requirement posed by the application.

```
selectNetwork(List nets,MRN dest,QoSInfo req,real accuracy){
  Set selected;
  for(n = nets.getNext(); nets != empty){
       if (isReachable(n, dest) &&
           matchRequirements(req, n.qos) >= accuracy){
              selected.put(n);
       }
  }
  sortNetworksWrtAccuracy(selected);
  return selected.first();
}
```

Fig. 5. Network selection algorithm

3.3 Multi-radio unicast and multicast

Once defined the MRN@ addressing scheme and the operations enabling the network link selection, the *network-agnostic connectivity* layer provides two base communication facilities: *synchronous unicast* and *asynchronous multicast*.

***ubi*SOAP synchronous unicast –** allows for point-to-point communication between two *ubi*LETs sharing at least one network. Specifically, it is provided by means of a logical stream channel that is used by the *ubi*LETs to read/write the packets belonging to the ongoing communication. Figure 6 depicts the sequence diagram

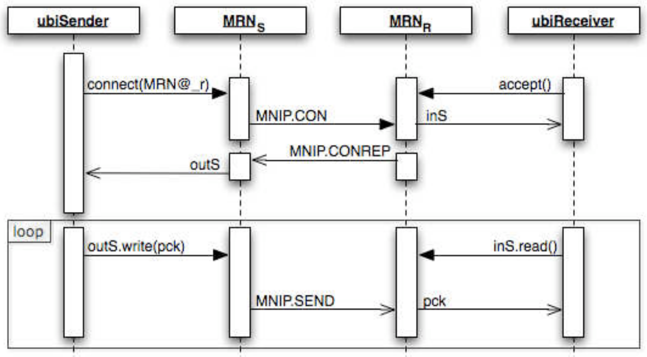


Fig. 6. Synchronous communication

showing the steps performed by the *network-agnostic connectivity* layer to provide synchronous unicast communication between two *ubi*LETs (namely *ubiSender* and *ubiReceiver*):

1) *ubiReceiver* listens for incoming connections (*accept*) and *ubiSender connect*s to it by means of its MRN@ (*MRN@_r*).
2) The connection request (*MNIP.CON*) is forwarded through the network to the *ubiReceiver*'s *MRN-Daemon* (MRN_R).
3) MRN_R accepts the connection, and returns to *ubiReceiver* an input stream (*inS*) for reading the incoming packets flow related to the ongoing communication.
4) MRN_R acknowledges *ubiSender* about the connection establishment and MRN_S returns to *ubiSender* an output stream (*outS*).
5) *ubiSender* uses *outS* to send messages to *ubiReceiver*.

***ubi*SOAP asynchronous multicast –** allows for one-to-many communication within a group of *ubi*LETs sharing at least one network. Specifically, it is provided by means of multicast packets that are sent to all members of a given group. Figure 7 depicts the sequence diagram

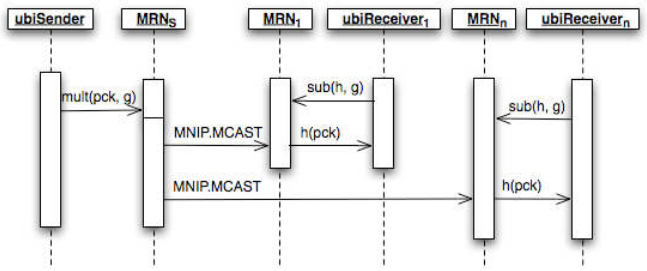


Fig. 7. Asynchronous multicast

showing the steps performed by the *network-agnostic connectivity* layer to provide asynchronous multicast communication among *ubi*LETs of a given group (namely a *ubiSender* and a set of *ubiReceivers*$_i$ ($i \in [1..n]$):

1) All *ubiReceiver*$_i$ implement the handler interface (h) and register it to a specific group name g.
2) *ubiSender* invokes the multicast method (*mult*) by providing the packet to be sent (*pck*) and the group name to which the packet belongs (g).
3) The *pck* packet is injected by MRN_S into all the networks currently available and then delivered to all MRN_i within such networks.
4) All MRN_i receive the packet and passes it to *ubiReceiver*$_i$ by means of call-back.

3.4 Customization

As already introduced, *ubi*SOAP aims at running on resources-scarce platforms (e.g., PDA and mobile phones), which have limited CPU power, memory, and battery life. To best fit the resources available on the hosting device, *ubi*SOAP provides two different (but equivalent and fully compatible) implementations of the *network-agnostic connectivity* layer, namely *shared* and *embedded*.

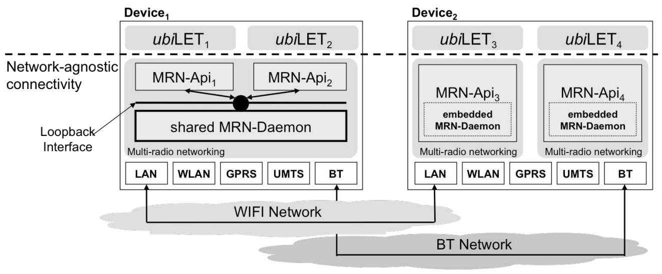


Fig. 8. Network-agnostic connectivity layer customization

Referring to Figure 8, Device$_1$ runs a *shared network-agnostic connectivity* layer where multiple *ubi*LETs (e.g., *ubi*LET$_1$ and *ubi*LET$_2$) access (through the API provided by *MRN-Api*) a "shared" instance of *MRN-Daemon*. Since all the *ubi*LETs access the same instance, possible conflicts can be solved in an automated way (i.e., two *ubi*LETs expressing conflictual QoS requirements over the interface activation).

On the other hand, Device$_2$ runs an *embedded network-agnostic connectivity* layer where the *MRN-Api* "embeds" the *MRN-Daemon*. In this case, *ubi*LET$_3$ and *ubi*LET$_4$ access two different, and standalone, instances of *MRN-Daemon*. This solution is lighter than the *shared* one, and is obviously appropriate when there exists only one *ubi*LET per device. In fact, the *shared MRN-Daemon* interacts with *MRN-Api* by means of a TCP socket bound to the loopback interface. This requires for having a synchronized thread-pool managing the incoming concurrent requests. It thus implies both larger memory footprint and computational needs. However, *embedded MRN-Daemon*s cannot communicate with each other and then, they cannot synchronize to solve possible conflicts.

4 SOAP over Multi-network Overlay

Providing SOAP communication within ubiquitous networks relates to comprehensively exploiting the rich, heterogeneous networking environment for the handling of SOAP messaging. In particular, *ubi*SOAP goal is to support:

- Mobility so that active sessions are maintained transparently to the application layer despite the mobility of nodes, as long as a network path exists.
- Efficient SOAP messages routing in multi-paths configurations (i.e., when multiple network paths exist between the client and the service). Specifically, SOAP messages must be exchanged over the most effective network link among those currently eligible, considering in particular energy efficiency and enforcement of QoS.
- Both point-to-point (i.e., client-server) and group-based SOAP communications using the same abstractions (i.e., MRN@ for service endpoint definition).
- Multi-network routing so that access to services in distant networks is enabled as long as there exists a path bridging the heterogeneous networks between the client and target service.

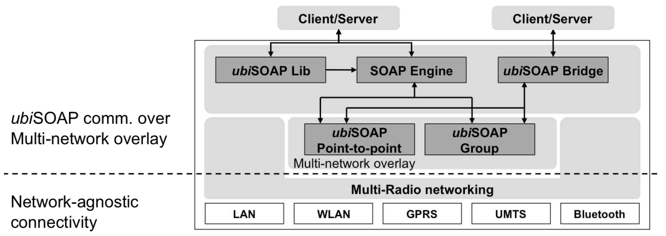


Fig. 9. *ubi*SOAP communication layer

To meet the above, *ubi*SOAP introduces a custom SOAP transport layer that leverages *network-agnostic connectivity* and is implemented as an *ubi*LET, then exploiting the API provided by the underlying *network-agnostic connectivity* layer.

Specifically, *ubi*SOAP *communication* is composed of two main entities (see Figure 9): (*i*) the *Bridge*, which is the building-block of the multi-network overlay in charge of forwarding SOAP messages across independent networks, and (*ii*) two new SOAP transports for point-to-point and group communication in ubiquitous networking, which have been implemented for different *SOAP engines* (current implementations include Axis2 and CSOAP).

Furthermore, key feature of *ubi*SOAP *communication* is that the applications containing the services or clients are no longer identified by the IP address of the device (e.g., http://www.ebay.com/services/cart), as traditionally in Web services. Rather, *ubi*SOAP exploits the MRN@ addressing scheme (e.g., b3gp://dd3ef7e3-5f50-3800-982d-62095c6e8075/services/cart) allowing services to be reached from different network paths.

Note that, while MRN@ is quite specific to *ubi*SOAP, both *unicast* and *multicast communication* support are standard features of common networking libraries. This allows for implementing the *ubi*SOAP communication layer on top of any IP-based network. Note also that, the two transports introduced by *ubi*SOAP can be used by the applications and the SOAP engines alongside other protocols such as HTTP or JMS. Indeed, which transport to use is defined by the application through the protocol defined in the URL of the targeted service.

4.1 Multi-network overlay

Thanks to the *ubi*SOAP *network-agnostic connectivity* layer, communication among nodes exploits the various network links that the nodes have in common by selecting the links that provide the required QoS. However, in some cases, it might also be desirable for nodes to be able to access services that are hosted in *distant* networks to which the requesting node is not directly connected to (e.g., to provide continuity of service despite node mobility). For example, in Figure 10, the device of Alice is connected to networks `a`, `i`, and `n`, through its various network interfaces. Clearly, the device can trivially access services hosted in these networks. However, it cannot access services hosted by Bob's device that is located in the distant networks `x`, `y`, and `z`. In fact, the *network-agnostic connectivity* layer does not provide neither an overlay IP network nor multi-network routing.

However, relying on the MRN@, together with both *unicast* and *multicast communication* schemes, *ubi*SOAP introduces an overlay network that is able to bridge heterogeneous networks, thus enhancing overall service connectivity. In particular: (*i*) MRN@ addressing provides a two-layer identification scheme (i.e., $network\_ID \oplus IP$) allowing for uniquely identifying a device irrespectively of the network it belongs to, and (*ii*) *unicast* and *multicast communication* support allows for MRN@ management across the networks. Specifically, nodes that are connected to two (or more) different networks through their network interfaces can assume the role of *bridge* nodes. *Bridge* nodes quite literally "bridge" between two separate networks, relaying *ubi*SOAP *point-to-point* and *group* messages across those networks. Still, we assume that nodes will not request services that would require the consecutive traversal of more than five wireless networks (see [17], [18] for a detailed analysis on wireless communication) in order to access them. Hence, still referring to Figure 10, Alice has to route its request through an appropriate bridge node (i.e., bridges *A*, *B* and *C*, noting that each bridge node is displayed in each network it is part of).

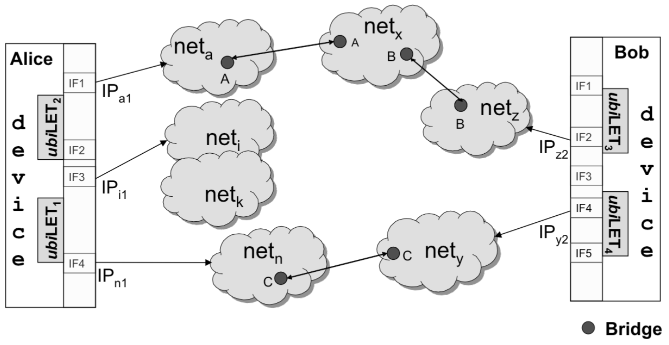


Fig. 10. Multi-network overlay through bridging

Specifically, *bridges* are in charge of routing messages within the multi-network overlay by determining the best route to reach a distant network. To achieve these tasks, bridges exchange and process presence information with each other. These routing messages are exchanged using the specific asynchronous multicast transport provided by the *network-agnostic connectivity* layer. Such messages are not forwarded across networks; thus, *bridges* only interact with the other *bridges* in their networks. To achieve effective routing across bridges, we exploit a straightforward approach based on the principle of Mobile Ad hoc NETwork (MANET) routing. In particular, bridge nodes advertise their presence to the nodes in their corresponding networks and exchange routing information. For this purpose, bridge nodes run an instance of OLSR [19] among each other. Instead of concrete node addresses, however, bridges store as destinations the identifiers of the various present networks (i.e., $network\_ID$) and as next hop the bridge that needs to be contacted next to eventually reach the target network. Being a proactive routing protocol, this inter-bridge OLSR instance gives each bridge the required routing information to reach all connected networks. Specifically, each bridge sends periodically (or upon request) presence beacon containing its connectivity information to nearby bridges according to the following protocol:

- The beacon not only contains the connectivity of the bridge, but also the connectivity information of other bridges it already received.
- A beacon is not forwarded by a bridge; each bridge sends its own beacon periodically.

- The connectivity information collected by a bridge is locally stored and used to build the multi-network topology map and find the different routes to a distant network.
- If a nearby bridge does not receive the beacon anymore, it considers that the missing bridge disappeared/is not valid anymore, updates its connectivity information, and sends again its beacon to propagate the notification that a bridge disappeared.

Whenever a non-bridge node wants to access a service outside one of the networks it is itself connected to, it may simply route the request to any bridge of choice that will then forward the request accordingly. As mentioned above, bridge nodes periodically advertise their presence to the nodes in their respective networks. As a further optimization, bridge nodes may include in their advertisements their OLSR routing tables so that non-bridge nodes may choose bridge nodes according to metrics such as network hops, etc.

4.2 *ubi*SOAP point-to-point transport

The *ubi*SOAP *point-to-point transport* is a connection-oriented transport for supporting communication between a service consumer and a service provider. As depicted in Figure 9, the *ubi*SOAP *point-to-point transport*: (i) leverages the *network-agnostic connectivity* layer to send and receive messages relying on the MRN@ addressing scheme, and (ii) interacts with a custom *SOAP engine* to deliver the message to the appropriate service.

The *SOAP engine* is in charge of parsing SOAP messages, and properly setting up the SOAP message parameters (i.e., URL of the service, action to perform). In particular, *ubi*SOAP exploits the WS-Addressing standard in order to extend the SOAP header with proprietary information that allows for storing (on the source side) and retrieving (on the destination side) relevant information about the SOAP message. For example, the set of IP addresses associated to an MRN@ is embedded in the header of request (for the client) and response (for the service) messages. This enables communicating devices in different networks to keep track of mobile nodes and maintain sessions (as long as a communication path exists). Moreover, the use of WS-Addressing allows *ubi*SOAP to be still compatible with legacy SOAP engines. In fact, depending on the service's URL, *ubi*SOAP is able to select the specific protocol (i.e., a legacy one such as HTTP or the specific ones provided by *ubi*SOAP) to use in order to properly carry on the communication.

When the URL of the destination service is specified by means of the "b3gp" protocol (e.g., b3gp://dd3ef7e3-5f50-3800-982d-62095c6e8075/services/cart), the *SOAP engine* selects *ubi*SOAP *point-to-point* as transport layer and extracts the MRN@ from the SOAP header. Figure 11 depicts the sequence diagram showing the steps performed when sending a SOAP message between two applications, namely a *SOAPSender* and *SOAPReceiver*:

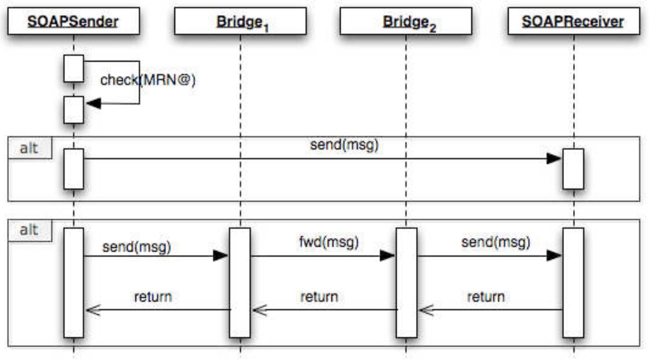


Fig. 11. *ubi*SOAP point-to-point transport

1) The *SOAPSender* first evaluates if the destination is directly reachable (i.e., the MRN@ of the sender and of the destination share a common network).
2) If true (first alternative), *SOAPSender* sends the message directly to *SOAPReceiver* by means of the *synchronous unicast* communication facility provided by the *network-agnostic connectivity* layer.
3) If not (second alternative), the transport retrieves the MRN@ of *Bridge*$_1$ (directly reachable), encapsulates as plain data the application's SOAP message into a specific forwarding message, and sends this forwarding message to the bridge.
4) This message is then forwarded across bridges (*Bridge*$_2$) until it reaches the destination (i.e., *SOAPReceiver*) where the SOAP message is extracted and dispatched. While the *SOAPSender* blocks until the response is received, the forwarding message is routed across bridges using connectionless communication.
5) The response message is then returned to *SOAPSender* by possibly following a different route (i.e., across a different set of bridges)

Note that, when both the client and the service simultaneously change the complete set of IP addresses associated to their MRN@ (and no direct link exists) the session will break and the client needs to perform a service discovery (See Section 5) to find the same service again and reestablish the communication.

4.3 *ubi*SOAP group transport

In ubiquitous networking environments, it is crucial to support group interactions since it is central to advanced middleware services like dynamic discovery [20]. We thus introduce the *ubi*SOAP *group transport* over *multi-network overlay*, building upon the *asynchronous multicast* facilities provided by *network-agnostic connectivity* layer.

Specifically, the *ubi*SOAP *group transport* is a connectionless transport for one-way communication between multiple peers in multi-network configurations. The *ubi*SOAP *group transport* interacts with the *network-agnostic connectivity* layer to send group messages based on an MRN@ identifying the group, and with the *SOAP* *engine* to deliver the group's messages to the registered services.

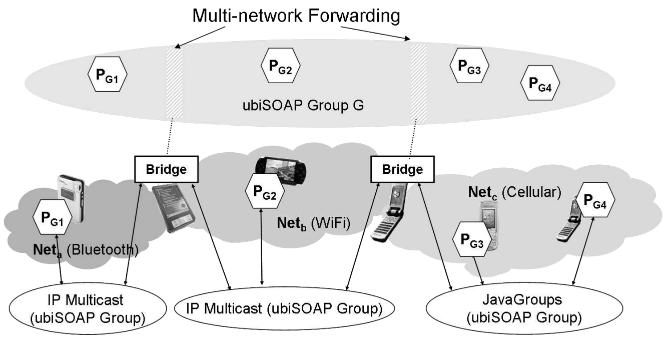


Fig. 12. Group transport over multi-network overlay

The base principles of the *ubi*SOAP *group transport* are depicted in Figure 12. Group transport is such that within an IP network, the network's multicast facility (i.e., IP multicast or higher level group communication like Java Groups) is used for communication among group members. Then, multicast messages are forwarded by bridges up to a fixed number of hops (i.e., 5 as discussed previously), while avoiding cycles and duplication.

As noted above, groups are identified with an MRN@. Multicast-based applications usually assume that all group members agree beforehand on a specific IP address for the group. We therefore also assume that all group members use the same MRN@ for the group. Hence, the packet is injected to the accessed networks by exploiting the asynchronous multicast communication provided by the *network-agnostic connectivity* layer. Figure 13 depicts the steps performed when an application (*SOAPSender*) sends a SOAP message to a group of $SOAPReceivers_i$ ($i \in [1..n]$) interconnected by means of a single *Bridge*:

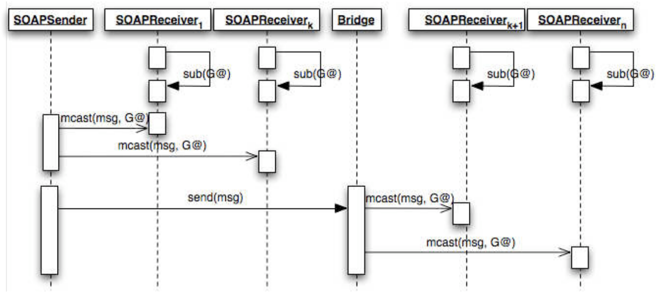


Fig. 13. *ubi*SOAP group transport

1) Each $SOAPReceiver_i$ subscribes to a specific group name specified as MRN@ (*G@*).
2) *SOAPSender* invokes the *mcast* method by providing the packet to be sent (*msg*) and the group name to which the packet belongs (*G@*).
3) The message is then disseminated, by means of the *asynchronous multicast* communication, through all the available networks directly accessed and delivered to all $SOAPReceiver_i$ ($i \in [1..k]$).
4) The same message is also sent to *Bridge* by means of the *ubi*SOAP *point-to-point transport* since *Bridge* is not subscribed to *G@*. *Bridge* is not aware about the group names.
5) *Bridge* forwards message to the other networks it is connected to.
6) The message is then delivered to all $SOAPReceiver_i$ ($i \in [k+1..n]$).

Note that when a bridge node receives a group message to disseminate, it first needs to decide whether the dissemination process should be terminated or not (i.e., maximum hop count reached). If not, it should then retrieve the list of networks to which this message has already been forwarded to from the SOAP header, and decides whether it should forward the message to some of the networks this bridge is connected to (i.e., new networks). If some of the bridge's networks have not been traversed, the bridge updates the list of networks traversed in the SOAP header with the new networks, and sends the message to the peers within the new networks. Such a message, can obviously reach another bridge, which in turn will possibly forward the message to other networks. While this algorithm prevents duplication due to cycles, message duplication (i.e., group members in some networks receive the same message multiple times) may happen if multiple network paths exist between the source and these networks. While not implemented currently, unique message Ids may be attached to group messages so that the *ubi*SOAP *group transport* may record the Ids of the messages received and discard duplicates.

Moreover, as group communication in the underlying *network-agnostic connectivity* layer is multicast-based, it does not guarantee the ordering or the delivery of messages. While ordering may be easily achieved on the receiving side, the overhead to provide group reliability is deemed too costly due to the dynamics of ubiquitous networks.

Also, while many mobile devices may run the same collaborative application, a user may only be interested in interacting with the ones at its location. Such scoping may be achieved by limiting the forwarding of group's messages or by adding forwarding constraints [21]. Moreover, while services are not able to directly return a result to a client (one-way multicast), a service may send a message (one-way unicast) on the group directed at a specific peer (i.e., similar to the socket call).

5 Ubiquitous Service Discovery

The service-oriented computing paradigm is structured around three key architectural entities: (i) service provider, (ii) service consumer, and (iii) service repository. The three entities interact with each other by following the so called service-oriented interaction pattern where service providers *publish* their service descriptions into a service repository, whereas service consumers query the service registry to *discover* service providers. If one or more target providers are present in the service registry, then the service consumer can select and *bind* to any of them at runtime.

Coordination among services providers, consumers and repository, relies on a service description formalism (or language) to describe the functional and non-functional properties (such as QoS, security or transactional aspects of networked services) complemented with a Service Discovery Protocol (SDP). Many academic and industry supported SDPs have already been proposed and leading SDPs in dynamic environments use a pull-based approach (SLP, WS-Discovery, Jini, SSDP), often supporting both the centralized and distributed modes of interaction: clients send requests to service providers (distributed pull-based mode) or to a third-party repository (centralized pull-based mode) in order to get a list of services compatible with the request attributes.

Building on the tremendous number of proposed service discovery protocols and accounting for the specifics of ubiquitous computing [20], we introduce a Ubiquitous Service-Discovery Service (*ubi*SD-S) that provides dynamic, interoperable, context-aware service discovery. *ubi*SD-S is mainly a reengineering of the open source MUSDAC multi-protocol service discovery platform [21] on top of *ubi*SOAP in order to support service discovery in multi-radio, multi-network environments. *ubi*SD-S uses a hierarchical approach for service discovery in multi-network environments (see Figure 14). Indeed, a (logically) centralized repository coordinates service discovery within an independent network, while *ubi*SD-Ss in different networks communicate together in a fully distributed way to disseminate service information. While in MUSDAC service discovery and access were tightly integrated, *ubi*SD-Ss are only concerned with service discovery, and rely on *ubi*SOAP group communication to disseminate service information across networks. Changes in the multi-network overlay (e.g., broken propagation paths) are then taken care of transparently.

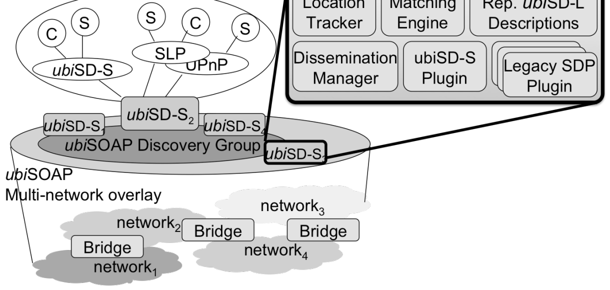


Fig. 14. Ubiquitous service discovery

Any *ubi*SD-S stores *ubi*SD-Language (*ubi*SD-L) [22] service descriptions that are either generated by legacy SDP plugins (e.g., UPnP2*ubi*SD-S plugin) or directly registered by service providers using the *ubi*SD-S plugin. We use a hierarchical service description format that actually combines a number of distinct documents specifying different facets of the service. The *ubi*SD-L description acts primarily as a top-level container for additional files describing facets of the service. For example, a WSDL document may be used to describe the service interface while non-functional properties can be described using existing QoS and context models. The *ubi*SOAP grounding of host, that identifies the networks and IP address at which the service's host is network-reachable (i.e., MRN@ and mapping) is described in such separate document thus facilitating dynamic updates.

*ubi*SD-S provides an explicit API supported by the *ubi*SD-S *plugin* that enables clients (resp. providers) in a network to discover (resp. advertise) a service in the multi-network environment. Indeed, it enables clients and providers to benefit from advanced discovery features (e.g., context-awareness) by directly issuing requests or advertisements in the *ubi*SD-L format. Specific *legacy SDP plugins* register with the active SDPs in the network, and translate requests and advertisements in legacy formats to *ubi*SD-L (e.g., SLP and UPnP in Figure 14), which are stored in the *ubi*SD-S *repository*.

The *matching engine* then combines various matching algorithms to support the various elements of the service description (for both requests and advertisements), and thus provides comprehensive interoperability between SDPs [22]. Finally, the *dissemination manager* controls the dissemination of local requests and the compilation of the results returned by distant *ubi*SD-Ss, while the *location tracker* collaborates with lower-level services in the *ubi*SOAP middleware to maintain the physical address of mobile services discovered in the environment.

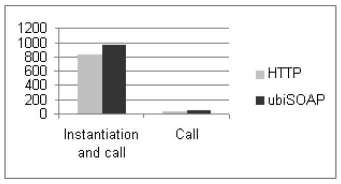
6 *ubi*SOAP Performance Evaluation

The *ubi*SOAP middleware has been implemented using Java for both desktop (J2SE) and mobile (J2ME CDC) environments and released under open source license[4]. To assess the efficiency of the *ubi*SOAP middleware, we evaluate the processing time to call a simple Echo Web service under various network configurations. We evaluate both the time required to call the Web service the first time, which includes the dynamic service creation/instantiation, and the time required to call the Web service after it is deployed. Tests are performed on a Windows XP PC with a 2.6GHz processor and 1 GB of memory for the desktop platform, and on a HP iPaq hw6910 (Intel PXA 270 at 416 MHz) and a HP iPaq 110 (PXA310 at 624 MHz) for the mobile platforms. We use IBMs J9 JVM (J2ME CDC 1.1), and the open source CSOAP lightweight SOAP engine[5]. Results presented are the average of 5 runs with 100 calls each.

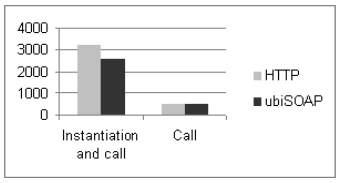
Figures 15.a) to d) assess the performance in ms, of *ubi*SOAP unicast transport versus HTTP in the following configurations: a) both client and service provider

4. *ubi*SOAP is composed of Multi-radio Networking and B3GSOAP packages available at http://www.ist-plastic.org

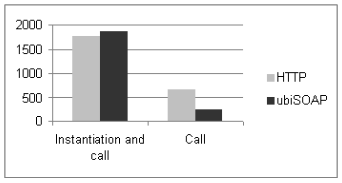
5. available at http://www-rocq.inria.fr/arles/download/ozone/.



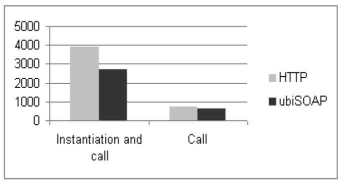
a) Local



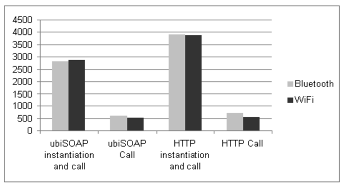
b) Desktop to PDA over Bluetooth



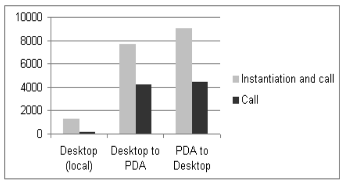
c) PDA to Desktop over Bluetooth



d) PDA to PDA over Bluetooth



e) Bluetooth versus WiFi



f) Shared *MRN-Daemon* usage

Fig. 15. *ubi*SOAP communication performance evaluation

running on the desktop platform, b) client running on desktop and service provider on PDA, c) client running on PDA and service provider on desktop, and d) client and server running each on a PDA. Provided results subdivide into the response time of the initial call including the service instantiation (i.e., *Instantiation and call*), and the average response time of subsequent calls (i.e., *Call*). Configuration a) shows that the time taken for service instantiation far exceeds the one of calls. The response time of *ubi*SOAP is slightly higher than the one of HTTP, which is due to the management of MRN@ whose processing gets noticeable due to the fact that the communication is local. Configuration b) then demonstrates that *ubi*SOAP outperforms HTTP by about 25% at instantiation time. This is explained by the fact that the processing overhead (header and session management) is much lighter in *ubi*SOAP, and the difference in processing is emphasized by the limited capacities of the devices (slow task switching, memory access, ...). As shown by Configuration c), in the case where the service provider runs on the desktop, HTTP performance for instantiation is slightly better, because the difference in the management of session is negligible on the desktop, while *ubi*SOAP on the desktop adds the cost of MRN@ resolution through multicast. On the other hand, the *ubi*SOAP performance for calls is better, again due to lightweight session management on the client side and the fact that the MRN@ needs only to be resolved at the first call. Finally, Configuration d) aggregates the above results b) and c), showing the enhanced performance of *ubi*SOAP over HTTP when both client and service provider run on a PDA.

Figure 15.e) complements the above measures by showing the performance of wireless communications between PDAs using both WiFi and Bluetooth. The higher bandwidth of WiFi makes it obviously better positioned for handling communication compared to Bluetooth, even for small size messages. However, this takes into account performance only, while other criteria are of relevance like power consumption [8], as addressed by the QoS-aware network selection realized by the *ubi*SOAP *network-agnostic connectivity* layer. Although the theoretical bandwidth for Bluetooth and WiFi is significantly different, the actual bandwidth available to applications depends on the hardware, drivers, and, in our case, the Java VM ability to cope with the load. As demonstrated in the experiments, the actual bandwidth for Java applications on PDA is almost identical for Bluetooth and WiFi.

In general, the scheduling of communication over network interfaces shall account for the QoS requirements of networked applications. Such requirement and specifically conflicts between different applications should be taken into account when activating/deactivating network interfaces or performing handover. To this extent, the *shared* version of *ubi*SOAP (see Section 3.4) deals with coordinated usage of the network interfaces, so that the actual network used for communication is selected according to the applications that are currently run. However, such a solution suffers from the resource availability of PDAs that is still currently limited, and accommodates poorly the concurrent execution of applications. Indeed, as shown by results of Figure 15.f) that provides the response time of *ubi*SOAP with the *MRN-Daemon* running, while the overhead on the desktop is reasonable, it increases dramatically when performed on the PDA due to the constant process switching.

	SLP	UPnP	WSDL
Discovery request	22.8	32.4	243
Discovery request + transl. to *ubi*SD-L	23.4	85.1	287
Overhead of the translation	0.6	52.7	44

TABLE 1
Legacy to *ubi*SD-L translation (micro-seconds)

To complete the discussion on the performance of *ubi*SOAP, we report here the evaluation of *ubi*SD-S [22]. In particular, Table 1 provides the processing time for the translation of service descriptions (requests and advertisements) from selected legacy SDPs to *ubi*SD-L descriptions. The first line represents the time to process a discovery request using SLP, UPnP and WSDL excluding the time to parse XML descriptions. The second line represents the time to process a discovery request in addition to the time to translate the request to *ubi*SD-L. Finally, the third line represents the overhead of the translation. In this experiment times are given in microseconds. As it can be observed, this time increases with the complexity of the original description, and in particular the complexity and size of the original XML data to process. Overall, the translation time is not significant (tens to hundreds of micro-seconds) compared to the overall discovery time.

	Requested service					
	*ubi*SD-L SLP		*ubi*SD-L UPnP		*ubi*SD-L WSDL	
	Parse	Match	Parse	Match	Parse	Match
SLP	1798	9.2	1844	9.0	1832	9.9
UPnP	1907	10.7	1868	18.0	1859	16.9
WSDL	1882	10.3	1829	18.1	1922	17.4

TABLE 2
Parsing and matching processing time (micro-seconds)

Table 2 provides the processing time of the matching algorithms for the different combination of service requests and advertisements. From the results of this experiment we can notice that the time needed to parse service and request descriptions is almost the same, because they are all *ubi*SD-L descriptions (1865 microseconds on average with less than 2% of standard deviation). Overall, parsing and matching *ubi*SD-L descriptions is also negligible with respect to the processing time for SOAP communication (see Figure 15).

Next section further assesses our work by demonstrating how *ubi*SOAP served to implement innovative real-life service-oriented applications.

7 *ubi*SOAP IN ACTION

*ubi*SOAP has been developed as part of a larger initiative on assisting the development of dependable services for ubiquitous networking environments, which was undertaken by the European IST PLASTIC project [23]. The PLASTIC project specifically investigated the development of the PLASTIC platform, decomposing into a development environment, service-oriented middleware and validation framework for the target ubiquitous services. *ubi*SOAP then defines the PLASTIC core middleware while advanced middleware services have been developed on top of it to address the requirements of ubiquitous networking (i.e., ubiquitous service discovery – provided by *ubi*SD-S – and composition, security and context management). Further, the PLASTIC platform has been assessed against actual case studies in the area of eHealth, eLearning, eBusiness and eVoting. This in particular allowed us to extensively experiment with the *ubi*SOAP middleware. More specifically, three applications have been built as mobile ubiquitous services[6]: (i) *Pocket doctor* that allows for carrying on medical consultations to qualified health professionals, (ii) *Field service management* that allows for optimizing the management of -on the field- (or out of the office) operations, and (iii) *Crisis management system* that allows for providing e-voting services to mobile voters that connect through mobile devices in an ad hoc manner.

These applications have been implemented as a set of Web services exploiting the capabilities provided by *ubi*SOAP. In particular, *ubi*SOAP has been used to solve communication issues due to the participants mobility. In fact, by means of the *multi-radio networking* layer and *multi-network overlay*, *ubi*SOAP increases the perimeter of reachable participants assuring their availability anytime and everywhere. Furthermore, all scenarios exploit both *ubi*SOAP *point-to-point* and *ubi*SOAP *group transports* in order to achieve service access and events dissemination, respectively.

The assessment carried out by the industrial partners pointed out the effectiveness of *ubi*SOAP for achieving service-oriented computing in the context of ubiquitous networking. In particular, industrial partners have appreciated the *multi-network overlay* and *ubi*SOAP *group transport*, which have been stated as innovative and useful middleware functionalities. On the other hand, the *network-agnostic connectivity* functionality has been evaluated difficult to deploy due to its dependencies on external commercial libraries. Indeed, this layer is in charge of managing network radio adapters at low-level, and its implementation heavily depends on the target device in terms of hardware, operating system and drivers. Since current Java implementations do not provide any class enabling the low-level management of network radio adapters, we developed a set of C#-based drivers for Windows Mobile by relaying on third-party libraries (e.g., Bluetooth stack), which can be accessed from Java. Unfortunately, these libraries might suffer of compatibility issues depending on the operating system running on the device (e.g., Windows CE, Windows Mobile 5 and Windows Mobile 6).

8 RELATED WORK

Work related to *ubi*SOAP is manifolds and range different research areas from ubiquitous computing to wireless web-service technologies and multi-radio networks integration. However, to the best of our knowledge *ubi*SOAP is the first attempt to consider all these aspects together to offer an integrated set of middleware facilities for achieving service provision in ubiquitous networking environments.

Literature about ubiquitous and pervasive computing proposes plenty of different middleware classes each addressing a specific issue: (i) Context-aware middleware [24] deals with leveraging context information to

6. Further details including videos and assessment reports can be found at [23].

provide user-centric computation, (*ii*) Mobile computing middleware [25] aims at providing communication and coordination of distributed mobile-components, (*iii*) Adaptive middleware [26] enables software to adapt its structure and behavior dynamically in response to changes in its execution environment. However, each middleware provides an ad hoc approach, whereas standard-compliant solutions are still missing. On the contrary, *ubi*SOAP aims at providing a communication layer enabling WS-* standards within ubiquitous networking environments.

The widespread adoption of WS technologies combined with mobile networking has led to investigate the definition of architectures dedicated to mobile Web services [27]. Overall, existing efforts towards enabling mobile Web services platforms resemble those in the area of mobile services platforms at large. That is, those platforms address the development of service-oriented applications on mobile, wireless devices that act mostly as Web service clients. Indeed, although todays wireless network and device technologies enable mobile devices to act as Web services providers, technological mobile WS platforms primarily target mobile devices as service clients. To this extent, many optimizations for SOAP have been proposed. First, various SOAP engines have been implemented to improve memory and CPU usage [28] of resource-constrained devices. On the other hand, SOAP message compression [29], [30] improves the bandwidth requirement of SOAP communication by compressing the XML text in binary data at the expense of CPU usage and latency. For instance, the SOAP Message Transmission Optimization Mechanism aims at making WS interactions more efficient by relying on binary-based messages, which consume less bandwidth, and processing and memory capacities, than XML-based messages [31]. However, the processing overhead of SOAP messages, associated with the handling of header and body parts, still affects performance of resource-scarce devices. To this extent, μSOA [32] aims at enabling SOAP messaging in the context of embedded systems by reducing the message size and parsing overhead. Specifically, μSOA relies on introducing a gateway entity, called μSOA Proxy Service, which translates SOAP messages to the μSOA message format and forwards them to the destination node. However, this solution does not take into account the mobility aspect inherent to ubiquitous networking environments.

To effectively enable mobile Web services and related wireless SOAP, *ubi*SOAP comprehensively exploits the ubiquitous networking environment by dealing with multi-radio networking on the mobile device. Concerning this issue, the Third Generation Partnership Project (3GPP) specifies protocol standards for dealing with the new telecommunication networks. Further 3GPP defines a standard layered architectures (decomposing into the network, control and service layers) enabling service-oriented applications in the B3G network [33]. In that direction, IMS (IP Multimedia Subsystem) and UMA (Unlicensed Mobile Access) introduce standardized architectures for telecom operators that want to provide mobile and fixed multimedia services in the multi-radio networking environment. In particular, UMA and IMS systems integrate unlicensed wireless networks and 3G Cellular networks into a new complex network, which enables subscribed devices to seamlessly roam between them. Both systems require the network operator to deploy new entities within the network that allow the native infrastructures to work together. Hence, the network operator controls and manages the new infrastructure, requiring clients to subscribe contracts regulating the network access. This does not allow clients to self-organize in spontaneous communities exploiting the network facilities. Contrary to this closed, network-controlled approach, *ubi*SOAP provides clients with abstractions that let them to autonomously adapt to the available networks and to benefit from their characteristics. This requires neither to modify the network infrastructure nor to establish contracts with a predetermined network operator.

A well-known fact to be considered is that most user devices do not have a globally visible IP address, but a private one that is translated to and from global address(es) via firewalls or Network Address Translation (NAT) routers. This is usually not a problem as only service consumers are executed on these hosts, while providers have public IP addresses. Indeed, incoming connection requests are a problem as the intermediate hosts has to forward the request. Both network- and application-level solutions have been proposed for enterprise environments – i.e., WS-Dispatcher [34] and TARGET [35]. Such solutions basically rely on an statically configured intermediary that both consumer and providers access to communicate. On the other hand, *ubi*SOAP focuses on pervasive environments composed by loosely coupling multi-networks through devices embedding multiple network interfaces. Indeed, on such devices, we assume that network routing, firewalls or NATs will not be activated. However, if a user needs to reach services behind firewalls, we consider that the bridge component on the users mobile device automatically connects to the bridge in the home network. Alternatively, the bridge component may be deployed in the DMZ of an enterprise network.

9 Conclusion

Service-oriented computing appears as a promising paradigm for ubiquitous computing systems that shall seamlessly integrate the functionalities offered by networked resources, both mobile and stationary, both resource-rich and resource-constrained. In particular, the loose coupling of services makes the paradigm much appropriate for wireless, mobile environments that are highly dynamic. However, enabling service-oriented computing in ubiquitous networking environments raises key challenges among which overcoming resource constraints and volatility of wireless, mobile devices. This has in particular led to introduce lightweight service-oriented middleware [2], [1], [3], [36]. However, to the best of our knowledge, none of the existing solutions comprehensively integrate the full capacity of today's ubiquitous networking environments, which allow wireless devices to interact via multiple network paths. Such a feature actually enables ubiquitous networking and in particular overcoming the nodes' mobility through vertical handover across networks. This allows for tuning network usage according to application requirements, thus enhancing overall QoS.

Exploiting multi-radio connectivity has led to the definition of various algorithms for optimizing the scheduling of communications over multiple radio interfaces, e.g., [5], [37], [8]. Building on this effort, this paper has introduced a *network-agnostic connectivity* layer, which leverages multi-radio networking by means of a special addressing scheme for networked services, namely MRN@, a QoS-aware network selection mechanism and both *unicast* and *multicast* communication facilities. In particular, this layer is in charge of managing the low-level heterogeneity inherent to multi-radio networking environments, by allowing for the exploitation of different application-level communication protocols.

Building upon these functionalities, the *ubi*SOAP *communication* layer implements two different SOAP transports, namely *ubi*SOAP *point-to-point* and *ubi*SOAP *group*, which leverage *network-agnostic connectivity* to enable the ubiquitous networking of Web services deployed on various devices – e.g., PDAs and smart phones – embedding multiple radio interfaces. Furthermore, in order to make the service-oriented computing paradigm effectively ubiquitous, *ubi*SOAP provides also an Ubiquitous Service-Discovery Service (*ubi*SD-S), which allows services to be published and discovered in the ubiquitous networking environment.

*ubi*SOAP has been extensively experimented as part of the PLASTIC European project, being the basis for the development of various ubiquitous services in the area of eBusiness, eHealth, eLearning and eVoting. Experiment further shows that the performance of *ubi*SOAP are in general better than default SOAP-over-HTTP transport, thanks to lightweight session management. It is worth noting that *network-agnostic connectivity* can be used as low-level transport for providing application-level interaction protocols different from SOAP. For instance, both the "Pocket doctor" and "Field service management" applications make use of the *ubi*SOAP *communication* layer for achieving service-oriented interactions, as well as of a content-based communication protocol built on top of the *network-agnostic connectivity*.

Recently, the Devices Profile for Web Services (DPWS) [38], which specifies a minimal set of implementation constraints to enable Web Service on resource-constrained devices, has been accepted as part of the WS standards. Indeed, DPWS extends WS technology allowing for seamless integration of device-provided services. Specifically, DPWS defines an architecture where devices run two types of services: (*i*) *hosting services* that represent the devices, and (*ii*) *hosted services* that are mostly functional and represent the services hosted by devices. DPWS specifies also a set of built-in services such as: discovery services, metadata exchange service and publish/subscribe eventing services. To this extent, even though *ubi*SOAP is already compliant with SOAP (then enabling de facto the use of WS-*), our current work on further evolution of *ubi*SOAP is towards meeting the numerous requirements of ubiquitous computing, as well as the DPWS specification. Part of this effort lies in introducing UDP-based SOAP transport [39] and making (*ubi*SD-S) compliant with the new Web Services Dynamic Discovery [40] standard. Obviously, *ubi*SOAP needs also to be complemented with a number of middleware services to deal with QoS; such an issue has been addressed as part of the PLASTIC project where middleware solutions for security and context awareness were investigated. We are investigating how to provide reliability and privacy on top of the *ubi*SOAP *network-agnostic connectivity* layer [41]. We are also examining the coupling with semantic-based solutions to enable dynamic composition of services [42].

Acknowledgments

This work is part of the IST PLASTIC project and has been funded by the European Commission, FP6 contract number 026955, http://www.ist-plastic.org/.

References

[1] U. Bellur and N. C. Narendra, "Towards service orientation in pervasive computing systems," in *Proc. of the international conference on information technology: coding and computing*, 2005.

[2] V. Issarny, D. Sacchetti, F. Tartanoglu, F. Sailhan, R. Chibout, N. Levy, and A. Talamona, "Developing ambient intelligence systems: a solution based on web services," *Journal of automated software engineering*, vol. 12, no. 1, 2005.

[3] F. Aijaz, B. Hameed, and B. Walke, "Towards peer-to-peer long lived mobile web services," in *Proc. of the 4th international conference on innovations in information technology*, 2007.

[4] G. Gehlen, F. Aijaz, and B. Walke, "Mobile web service communication over UDP," in *Proc. of the 64th vehicular technology conference*, 2006.

[5] A. Qureshi and J. Guttag, "Horde: separating network striping policy from mechanism," in *Proc. of the 3rd international conference on mobile systems, applications, and services*, 2005.

[6] J. Sorber, N. Banerjee, M. D. Corner, and S. Rollins, "Turducken: hierarchical power management for mobile devices," in *Proc. of the 3rd international conference on mobile systems, applications, and services*, 2005.

[7] J. Su, J. Scott, P. Hui, J. Crowcroft, E. de Lara, C. Diot, A. Goel, M. Lim, and E. Upton, "Haggle: seamless networking for mobile applications," in *Proc. of the 9th international conference on ubiquitous computing*, 2007.

[8] D. Charlet, V. Issarny, and R. Chibout, "Energy-efficient middleware-layer multi-radio networking: an assessment in the area of service discovery," *Comput. Netw.*, vol. 52, no. 1, 2008.

[9] H. Huang and J. Cai, "Improving TCP performance during soft vertical handoff," in *Proc. of the 19th international conference on advanced information networking and applications*, 2005.

[10] H. J. Wang, R. H. Katz, and J. Giese, "Policy-enabled handoffs across heterogeneous wireless networks," in *Proc. of the 2nd IEEE workshop on mobile computer systems and applications*, 1999.

[11] M. Caporuscio, D. Charlet, V. Issarny, and A. Navarra, "Energetic performance of service-oriented multi-radio networks: issues and perspectives," in *Proc. of the 6th international workshop on software and performance*, 2007.
[12] K. Y. Lai, T. K. A. Phan, and Z. Tari, "Efficient SOAP binding for mobile web services," in *Proc. of the 30th IEEE conference on local computer networks*, 2005.
[13] J. Grudin, "Group dynamics and ubiquitous computing," *Com. ACM*, vol. 45, no. 12, 2002.
[14] B. Wang, J. Bodily, and S. K. S. Gupta, "Supporting persistent social groups in ubiquitous computing environments using context-aware ephemeral group service," in *Proc. of the 2nd international conference on pervasive computing and communications*, 2004.
[15] P. Leach, M. Mealling, and R. Salz, "A universally unique identifier (uuid) urn namespace," RFC 4122, 2005.
[16] Network Working Group, "An ethernet address resolution protocol," RFC 826, 1982.
[17] P. Gupta and P. Kumar, "The capacity of wireless networks," *IEEE Transactions on information theory*, vol. 46, no. 2, 2000.
[18] J. Li, C. Blake, D. S. J. De Couto, H. I. Lee, and R. Morris, "Capacity of ad hoc wireless networks," in *Proc. of the 7th ACM international conference on mobile computing and networking*, 2001.
[19] P. Jacquet, P. Muhlethaler, T. Clausen, A. Laouiti, A. Qayyum, and L. Viennot, "Optimized link state routing protocol for ad hoc networks," in *Proc. of the IEEE international multi topic conference: technology for the 21st century*, 2001.
[20] F. Zhu, M. W. Mutka, and L. M. Ni, "Service discovery in pervasive computing environments," *IEEE pervasive computing*, vol. 4, no. 4, 2005.
[21] P.-G. Raverdy, O. Riva, A. de La Chapelle, R. Chibout, and V. Issarny, "Efficient context-aware service discovery in multi-protocol pervasive environments," in *Proc. of the 7th international conference on mobile data management*, 2006.
[22] S. B. Mokhtar, P.-G. Raverdy, A. Urbieta, and R. S. Cardoso, "Interoperable semantic & syntactic service matching for ambient computing environments," in *Proc. of the 1st international workshop on ad hoc ambient computing*, 2008.
[23] Plastic Consortioum, "PLASTIC: Providing lightweight and adaptable service technology for pervasive information and communication," http://www.ist-plastic.org.
[24] K. K. Ellebaek, "A survey of context-aware middleware," in *Proc. of the 25th conference on IASTED International Multi-Conference*, 2007.
[25] C. Mascolo, L. Capra, and W. Emmerich, "Middleware for mobile computing (a survey)," in *Neworking 2002 Tutorial Papers*, 2002.
[26] P. McKinley, S. Sadjadi, E. Kasten, and B. Cheng, "Composing adaptive software," *Computer*, vol. 37, no. 7, pp. 56–64, 2004.
[27] F. Hirsch, J. kemp, and J. Ilkka, *Mobile Web Services: Architecture and Implementation*. John Wiley & Sons, 2006.
[28] R. A. van Engelen and K. Gallivan, "The gSOAP toolkit for web services and peer-to-peer computing networks," in *Proc. of the 2nd International Symposium on Cluster Computing and the Grid*, 2002.
[29] S. Sakr, "Xml compression techniques: A survey and comparison," *J. Comput. Syst. Sci.*, vol. 75, no. 5, pp. 303–322, 2009.
[30] M. Ericsson, "The effects of xml compression on soap performance," *World Wide Web*, vol. 10, no. 3, pp. 279–307, 2007.
[31] XML Protocol Working Group, "SOAP message transmission optimization mechanism," http://www.w3.org/TR/soap12-mtom/.
[32] A. Wolff, S. Michaelis, J. Schmutzler, and C. Wietfeld, "Network-centric middleware for service oriented architectures across heterogeneous embedded systems," in *Proc. of the 11th International EDOC Conference Workshop*, 2007.
[33] P. Asprino, A. Fresa, N. Gaito, and M. Longo, "A layered architecture to manage complex multimedia services," in *Proc. of 15th International Conference on Software Engineering and Knowledge Engineering*, 2003.
[34] D. Caromel, A. d. Costanzo, D. Gannon, and A. Slominski, "Asynchronous peer-to-peerweb services and firewalls," in *Proc. of the 19th IEEE International Parallel and Distributed Processing Symposium*, 2005.
[35] F. Liu, G. Wang, W. Chou, L. Fazal, and L. Li, "Target: Two-way web service router gateway," in *Proc. of International Conference on Web Services*, 2006.
[36] S. B. Mokhtar, D. Preuveneers, N. Georgantas, V. Issarny, and Y. Berbers, "EASY: efficient semantic service discovery in pervasive computing environments with QoS and context support," *J. Syst. Softw.*, vol. 81, no. 5, 2008.
[37] R. Klasing, A. Kosowski, and A. Navarra, "Cost minimisation in wireless networks with bounded and unbounded number of interfaces," *Networks*, vol. 53, no. 3, pp. 266–275, 2009.
[38] OASIS, "Devices Profile for Web Services - Version 1.1," http://docs.oasis-open.org/ws-dd/dpws/1.1/cs-01/, 2009.
[39] ——, "SOAP-over-UDP - Version 1.1," http://docs.oasis-open.org/ws-dd/soapoverudp/1.1/cs-01/, 2009.
[40] ——, "Web Services Dynamic Discovery - Version 1.1," http://docs.oasis-open.org/ws-dd/discovery/1.1/cs-01/, 2009.
[41] R. S. Cardoso and M. Caporuscio, "Exploring multi-path communication in hybrid wireless ad hoc networks," in *Proc. of 1st International Workshop on Ad-hoc Ambient Computing*, Sophia Antipolis, France, 2008.
[42] S. B. Mokhtar, N. Georgantas, and V. Issarny, "COCOA: Conversation-based service composition in pervasive computing environments with qos support," *Journal of System and Software*, vol. 80, no. 12, pp. 1941–1955, 2007.

Mauro Caporuscio is Assistant Professor at the Dipartimento di Elettronica e Informazione, Politecnico di Milano – Italy. He received his Ph.D. in Computer Science from the University of L'Aquila, Italy (2006). He was Professional Research Assistant at the University of Colorado (2002), and Postdoctoral Researcher at INRIA Paris-Rocquencourt (2006-2009). He has published various papers on the most important international journals and conferences and has served in the program committee of various international conferences. His research interests mainly focus on the application of Software Engineering methodologies and techniques to the field of distributed systems. To this extent, particular attention is devoted to Event-based systems, Service Oriented Architecture and Pervasive Computing. Mauro Caporuscio current research is funded by the European Commission, Programme IDEAS-ERC, Project 227077-SMScom (http://www.erc-smscom.org).

Pierre-Guillaume Raverdy is now a Research Engineer at INRIA, France. He received his PhD from University Paris VI (Pierre & Marie Curie) in 1996 on load balancing in heterogeneous networks. He then worked for Sony R&D in Tokyo and in California on the integration of mobile devices in home networks. He then joined INRIA in 2004 to work on several european projects dealing with B3G networks and distributed computing, and focusing in particular on service discovery. He is interested in the various aspects of mobile computing (network, middleware and usages).

Valerie Issarny is Directrice de Recherche at INRIA, France. Since 2002, she is the head of the INRIA ARLES research project-team at the Paris-Rocquencourt research center. Her research interests relate to distributed systems, software engineering, middleware, pervasive computing, and service-oriented computing. She has been involved in a number of European and French projects and initiatives in the above areas. She has further been member of organization and program committees of leading events in those areas. She is in particular steering committee member of the Middleware and ESEC/FSE conferences. Further information about Valerie's research interests and her publications can be obtained from the ARLES Web site at http://www-roc.inria.fr/arles/.