

Omnisphere: a Personal Communication Environment

Franck Rousseau, Justinian Oprescu, Laurentiu-Sorin Paun, Andrzej Duda
LSR-IMAG Laboratory
BP. 72, 38402 Saint Martin d'Hères
France

{Franck.Rousseau, Justinian.Oprescu, Laurentiu-Sorin.Paun, Andrzej.Duda}@imag.fr

Abstract

Small ubiquitous devices connected by wireless networks will become future Internet appliances. To support them, communication networks must evolve to seamlessly assist appliances and provide advanced functionalities. We present a personal communication environment called Omnisphere that provides a communication and information universe surrounding wireless appliances. It is based on a high level concept called ambient services that allows to construct complex services out of primitive ones by connecting them with typed data flows. A typed data flow is an abstract view of communication between ambient services. It encapsulates three elements: channels, control, and metadata. Omnisphere provides a predefined service for discovery of component services and binding them together with data flows. Our strategy for service discovery is to delegate most of the operations to the network infrastructure and to automate them as much as possible. Based on the User ID and Appliance ID, Omnisphere retrieves the information that restricts the set of possible services: User Preferences, Device Capabilities, and Context. It then makes use of existing discovery protocols such as SLP, Jini, or UPnP to discover relevant services and matches them with the required characteristics. Such a discovery process relieves appliances, which may have limited resources, from the operation that may consume scarce resources and may require the availability of different discovery protocols on the appliance.

Keywords: *ambient networking, ambient services, wireless appliances, service discovery*

1 Introduction

Small ubiquitous devices connected by wireless networks will become future Internet appliances. To support them, existing communication networks must change their role to seamlessly assist appliances and provide advanced functionalities. This transition will be enabled by the concept of *ambient networking*.

The idea of ambient networks expresses two basic needs. First, wireless appliances require everywhere connectivity of various form, coverage, and capacity such as provided by wireless LANs or mobile telecommunication networks (GSM, GPRS, UMTS). Second, wireless appliances should be able to dynamically discover available communication sources and required services so that when the user moves from one place to another, she may benefit from the same environment (or similar if possible) augmented with some services specific to the current location. This last objective can be fulfilled by an infrastructure of services that can be acquired on demand and matched with the user preferences, device capabilities, and context information. In this way, appliances evolve in a spontaneous environment created and customized depending on a given appliance, its location, current usage, user preferences and intentions, as well as on network connectivity. We present a personal communication environment called *Omnisphere* that aims at designing and prototyping such an ambient network for wireless appliances.

Omnisphere is a communication and information universe surrounding wireless appliances. When an appliance enters Omnisphere, it authenticates itself so that the user preferences and device capabilities can be retrieved. Based on this information and the current context, Omnisphere discovers interesting communication sources and services, and dynamically binds them to form more complex services. Such complex services are specified in a declarative language able to describe how primitive services are interconnected, what data types are accepted on input and generated on output, and what are the attributes of services that should

be discovered and dynamically bound. The whole process is designed for minimal user intervention — we would like to automatize most of the operations needed when appliances move and change network connections.

The paper is organized as follows. Section 2 introduces issues in ambient networking, Section 3 presents the related work, and Section 4 describes the architecture of Omnisphere. In Section 5, we present the platform chosen for the implementation, give its status, and provide some performance comparisons. Finally, we present conclusion and future work (Section 6).

2 Ambient Networking for Wireless Appliances

The idea of ambient networks expresses the need for an infrastructure that seamlessly assist appliances and devices with basic functionalities required by applications such as *storing* information (caching, replication), *processing* (transcoding, adaptation), and *communicating* (moving and locating, call set-up, service publishing and discovery, security). Depending on applications, we can add other specialized functions such as presence detection, camera control, alarms, content selection, and many others. Ambient networks should also take into account an increasing number and a varying nature of computer devices: they become increasingly embedded into our environment (sensors and actuators) and connected via different types of networks, mostly wireless in case of portable devices. In a recent work, David Tennenhouse has introduced the concept of *proactive computing* [21]: our traditional office-centered computing based on close human-computer interaction needs to evolve to a new mode of operation in which networked systems composed of many processors, sensors and actuators work in an independent way. In such proactive systems the role of humans is reduced to supervision. We think that these ideas can be successfully applied to ambient networking.

Ambient networks rely on communication links of different types and characteristics (bandwidth, delay, coverage). Currently, many different networks coexist without real integration: the Internet, phone networks (POTS, cellular, GPRS, and future UMTS), digital radio and TV broadcast (DAB/DVB), home networks (X10, LonWorks, HomeRF). An appliance in an ambient network should be able to connect to a network it is closest to or to the network that fits the application requirements the best with respect for example to the cost, quality of service (QoS), current load, or neighborhood environment.

Ambient networks should offer to wireless appliances an infrastructure of flexible services that can be acquired on demand. In this way, appliances evolve in a spontaneous environment created and customized depending on a given

appliance, its location, current usage, user preferences and intentions, as well as network connectivity. An important technical issue related to the infrastructure is *service discovery*. Devices present in an ambient network have to propose their services whereas appliances need to discover what functions are available, choose the right ones, and bind them so that they can start to use them. Another problem is related to *service description*: how to express the properties of a service in a compact, yet sufficiently informative form so that appliances can easily decide which service to use.

How to use ambient networks and their service infrastructure is the next technical issue. As everything becomes dynamic and pluggable, we can imagine that applications can be created out of available services discovered in the infrastructure and combined together. We believe in a simple paradigm for specifying how services, content sources, and communication flows are connected, and what control operations should be performed to supervise their execution. In fact, we need something similar to the WWW linking system (based on URL addressing and the HTTP protocol) and the WWW declarative composition language (HTML) that has allowed the WWW to become a universal information space. In this view, services are elements to compose complex structures by means of typed communication connections able to transport raw data or temporal events, metadata describing their contents, and control commands.

3 Related Work

Recently, many projects have started to explore ambient networks and pervasive computing.

MIT project *Oxygen* aims at bringing an abundance of computation and communication to an infrastructure of mobile and stationary devices connected by a self-configuring network [5, 16]. The *Portolano* project at University of Washington seeks to create a testbed for investigating three fundamental areas of the emerging field of invisible computing: user interfaces, network infrastructure, and distributed services [13]. The HP Labs *Cooltown* project explores situations in which e-services meet the physical world, where humans are mobile, devices, and services are federated and context-aware, and everything has a web presence [8].

Several projects focus more on building an information space for pervasive and ubiquitous devices:

- At UC Berkeley, the *Ninja* architecture [2] seeks to enable a broad innovation of robust, scalable, distributed Internet services. The architecture defines the concept of *Ninja Paths* used for composing services, client devices, and active proxies.
- The goal of *DEAPspace* project at IBM Research is to provide a framework for interconnecting pervasive de-

vices by means of wireless networks [7]. Key features of the framework include hidden computing, spontaneous interaction, and proximity-based applications.

- The *Infosphere* [19] project at Georgia Institute of Technology Oregon Graduate Institute aims to build system software support for information-driven applications. Its main contribution is the *Infopipe* abstraction to support information flow with quality of service [11].
- The *Appliance Data Services* (ADS) project at Stanford University [9, 10] has similar goals to Omnisphere. It explores a vision of an appliance computing world in which users move data seamlessly among various devices. It emphasizes three principles that guide the design of such a computing infrastructure: bring devices to the forefront, minimize the number of device features, and place functionality in the network infrastructure.
- Part of the *Stanford Interactive Workspaces* project [20], the *Paths* mediation framework tries to bridge protocol and data format mismatches between components wishing to communicate. It provides a framework for dynamic connection of such endpoints on demand and for supporting ad-hoc interactions that are integral parts of ubiquitous computing [12].

Service discovery is a key aspect of the Omnisphere architecture: client hosts want to discover services proposed by servers. Many service discovery protocols do not require any network infrastructure, clients and servers being responsible to maintain a coherent distributed state. There are two basic modes of operation: *pull* and *push* mode. In the pull mode, clients query the environment about some services and servers concerned with a particular request reply. In the push mode servers periodically advertise their available services while the clients listen for these advertisements. The network may offer a central repository with which servers register their services so that clients may query the repository (*centralized pull*). An important aspect for the centralized mode is bootstrapping: how servers and clients find the repository. Besides the problems directly related to the protocol itself, like scalability, efficiency and security, adjacent aspects, like service description and query language, affect the overall performance of service discovery protocols. In the context of small wireless appliances, the additional constraints should be taken into account: limited resources and scarce bandwidth.

Several service discovery protocols have been proposed so far. Service Location Protocol [22, 6] operates in the pull mode (with a central repository or not). Java-based Jini relies on a central repository (centralized pull) [15]. The

Bluetooth protocol stack includes a simple discovery protocol [1]. Universal Plug & Play [14] is a complex protocol operating in two modes: distributed pull and push. It was designed for interoperability with specialized devices and uses XML service descriptions, the SOAP method invocation on services, and GENA for event notification. DEAPSpace project proposed a service discovery protocol based on the principle of distributed push optimized for wireless devices in an ad-hoc environment [7, 18]. Federation of different service discovery protocols has been proposed in an unifying architecture [3].

Omnisphere shares the overall vision with these proposals. However, it contributes in a different way to the design of the service architecture: we propose an innovating view of service composition that allows us to easily create complex services out of components. Unlike other projects, encapsulating data flows with metadata and control protocols favors horizontal integration involving devices that currently do not cooperate. We also think that our approach to service discovery based on delegation of discovery into the fixed infrastructure and on meta-queries provides better flexibility and performance than other existing schemes.

4 Omnisphere

Omnisphere is a communication and information universe surrounding wireless appliances. It proposes different services to satisfy rich application requirements that may depend on the current user, the current location, or context. Omnisphere provides a set of services that varies in time because some appliances may quit the current proximity or their neighborhood may change after a movement.

Omnisphere is based on the following principles:

- *Proactive behavior.* All operations of Omnisphere proceed with the minimal user intervention. Omnisphere automatically proposes the most suitable services to the user.
- *Best fit.* The most suitable services are those that fit the best the user preferences, the contents available in the network, the current context, and the characteristics of the devices.
- *Delegation.* Most of processing is delegated to the network infrastructure. Rather than performing a traditional service discovery by wireless appliances, Omnisphere takes care of finding appropriate services and proposing them to the appliances (*smart push*).
- *Implementation hiding.* Omnisphere defines high-level concepts to hide low-level implementation details or configuration settings.

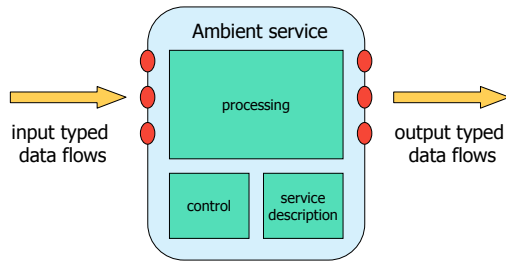


Figure 1. Ambient service.

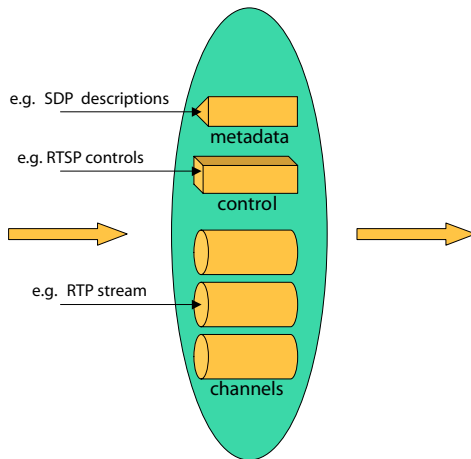


Figure 2. Structure of data flows.

- *Dynamic re-binding.* All association of services can be dynamically changed, e.g. a new better service can replace an old one. Omnisphere can also take care of service continuation after a movement.

Omnisphere comprises the following elements:

- *Ambient services.* An ambient service is an abstract view of a communication component. It represents a high-level concept that allows us to construct complex services out of primitive ones by connecting them with *typed data flows*. An ambient service encapsulates some processing, provides a signaling interface for control and publishes a service description. An ambient service is illustrated in Figure 1.
- *Typed data flows.* A typed data flow is an abstract view of communication between ambient services. It encapsulates three elements: *channels*, *control*, and *meta-data*. The structure of data flows is presented in Figure 2.
 - *Channels.* A channel provides a means for transporting typed data using a given protocol. Examples of protocols are HTTP, RTP, FTP, MPEG-2

Transport, DAB/DVB, and other media application protocols. The user can obtain the same data contents using different channels depending for example on the current context, device, or state.

- *Control.* A control protocol provides a means for controlling channels e.g. start playing, stop delivery. The control element relies on signaling protocols such as RTSP, SIP, H.323, MeGaCo, MGCP. In some cases, the control information is unified within one protocol, for example it is the case for HTTP. We also define a control protocol proper to Omnisphere needed for invoking the service discovery and binding process as well as for activating services.
- *Metadata.* Channels transport data contents having a well defined type and encoding, e.g. a video MPEG-2 or an audio MP3 stream, an e-mail. Metadata may also provide more information on the contents and the structure of transported data. Different metadata formats such as MIME, SDP, RDF, MPEG7, CPI may also be used.
- *Omnisphere service.* This predefined service runs in the fixed network infrastructure and is used for discovery of component services, binding them together with data flows, and re-binding when the state changes, for instance after a movement. It integrates traditional service discovery protocols such as SLP, Jini, or UPnP. The *Omnisphere service* matches discovered component services with user preferences, device capabilities, and the current context. In this way, the services that fit the best are used at a given instant.
- *Controller service.* It is also a predefined service, but it runs on the appliance. It takes care of requesting service discovery and activating discovered services. It communicates with the Omnisphere service by means of the *Omnisphere Control Protocol* which is a part of the data flow between these two services. The protocol allows an appliance to register within a given Omnisphere, request service discovery process, download the code of a service, and activate discovered services.

The architecture of Omnisphere is presented in Figure 3. Its goal is to automatically provide ambient services to the user who enters Omnisphere. This process starts with the user authentication and the identification of all mobile appliances brought in by the user. Then, Omnisphere discovers services that can be proposed to the user.

Our approach to the discovery of ambient services is to delegate most of the operations to the network infrastructure and to automate them as much as possible. Based on the User ID and Appliance ID, Omnisphere retrieves the information that restricts the set of possible services: *User*

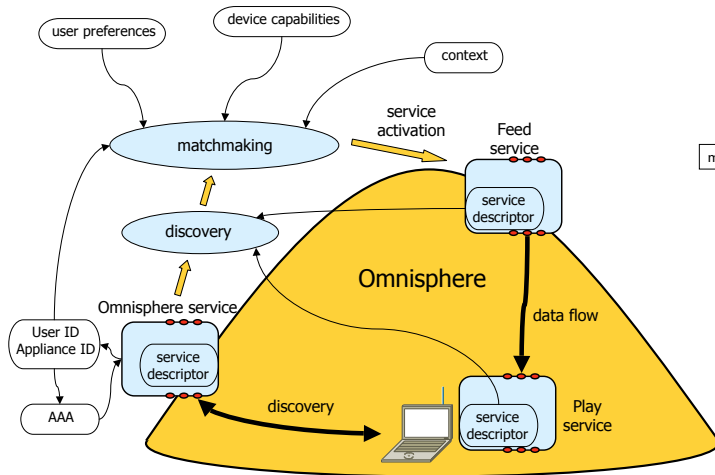


Figure 3. Omnisphere architecture.

Preferences, Device Capabilities, and Context. User Preferences define what are the most common needs of the user in terms of ambient services and different types of data. Device Capabilities provide information about the characteristics of appliances (screen resolution, processor speed, network interface). Context adds some location specific information such as the geographic location, available devices, type of environment — public or private, and all other information related to the close neighborhood of an appliance. Then, Omnisphere discovers relevant ambient services using available service discovery protocols such as SLP, Jini, or UPnP. The discovered services are matched with the required characteristics provided by User Preferences, Device Capabilities, and Context. The best fitting services are encapsulated with some control functions and a service descriptor to form ambient services that are then initialized by establishing data flows between them and activated. To take into account state changes, for example when an appliance becomes inactive, or a new appliance enters Omnisphere, these service bindings will be re-evaluated.

Such a discovery process relieves appliances, which may have limited resources, from the operation that may consume scarce resources and may require the availability of different discovery protocols on the appliance. It may also save critical radio channel resources, because only selected information is provided to the appliance.

To illustrate the discovery process in Omnisphere, we consider an example of a complex service *VideoDelivery*. It is composed of the following component services. *Feed* provides video streams of different formats. *Transcode* transcodes between different formats. *Excerpt* selects some parts of a video stream (we assume that we are able to select some interesting parts based on

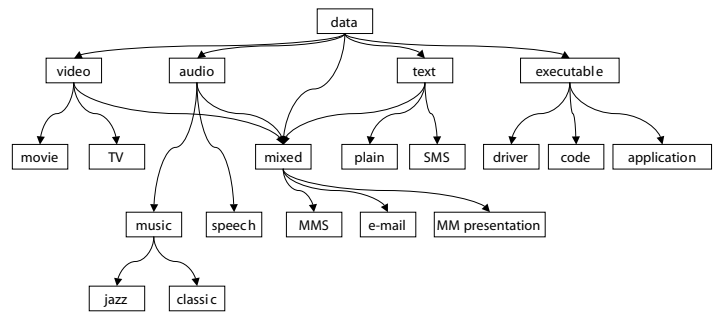


Figure 4. Hierarchy of types.

some predicate or a parameter) and provides them as a video clip, an audio summary, or just a simple text. *Play* integrates different media players that can run on different appliances (TV screen at home, notebook, PDA, GPRS phone). *Notify* sends video excerpts to different devices. There are several possible uses of these services. For example, when the user enters Omnisphere with his/her laptop at the office (abundant network bandwidth), Omnisphere will discover that suitable services are *Feed* that generates a MPEG stream and *Play* that runs on the laptop. Their interconnection will be set up automatically. When the user has to leave, *VideoDelivery* service will automatically rebind component services, for example, *Feed* will be now connected to *Excerpt*, which will be in turn connected to *Notify*. This interconnection will allow to send an excerpt of the video stream as a MMS message to the cellular phone.

As ambient services are composed out of available components discovered by Omnisphere, we propose a declarative language able to describe what data types are accepted on input and generated on output, how component services are interconnected, and what are the attributes of services that should be discovered and dynamically bound.

4.1 Service description

To compose services, we need a means for specifying their properties to discover them dynamically. Discovery of matching services requires precise definition of data types and content attributes. We define a simple yet powerful hierarchy of types that categorize the information conveyed by data flows. The hierarchy tries to capture the most common data types in a similar way to the DNS name hierarchy. Figure 4 presents a typical hierarchy that may be made more complete by adding nodes at different levels.

Note that the hierarchy does not mix encoding formats with data types as it is the case for MIME. This kind of information will rather be associated with nodes of the hierarchy as attribute information. An at-

tribute is specified as a pair name=value, for example, /audio:encoding=mp3/music/jazz describes a MP3 jazz piece. We can also associate some more metadata information with data contents, e.g. the title of a movie, the subject of an email.

Knowing the type of data, we can decide whether it is possible to connect different services. Type T_i is said *compatible* with T_j , if T_i is in the subtree of T_j in the type hierarchy. This notion expresses the fact that if one service can accept data of type T_j , it can also accept data of type T_i . Converse is not true.

For example, a data flow of type /audio/speech is compatible with /audio. In terms of application scenarios, this means that we can send a speech clip to a service that accepts audio, however we cannot send a data flow of type /audio to a service such as a speech recognition module that only accepts /audio/speech.

Using our data types we are able to describe services of our example:

```
Feed: OUT /video;
Transcode: IN /video; OUT /video, /text, /audio;
Excerpt: IN /video;
          OUT /mixed/MMS, /text/SMS, /mixed/e-mail;
Play: IN /video, /text, /audio;
Notify: IN /mixed/MMS, /mixed/e-mail, /text/SMS;
```

Note that the specification at this level is fairly general, we do not need to specify a particular encoding, unless explicitly required.

To instantiate a service, we need to find services with compatible data types of flows and with desired properties. To accomplish the discovery process, services have to publish their description so that we can match them with the description of the composite service. For example, a data flow provided by an instance of Feed service can be partially described as follows:

```
Flow1
  Data
    Type=/video:encoding=MPEG2/movie:title="Amelie
          from Montmartre"
  Channel1
    Type=RTP
    Address=video-server.org
    Port=11111
  Channel2
    Type=HTTP
    Address=http://video-server.org:80/amelie1.mpg
  Metadata
    Type=RDF
    Address=http://video-server.org:80/amelie.rdf
  Control
    Channel=Channel1
    Type=RTSP
    Address=rtsp://video-server.org/amelie1
```

The service may provide other flows having different formats.

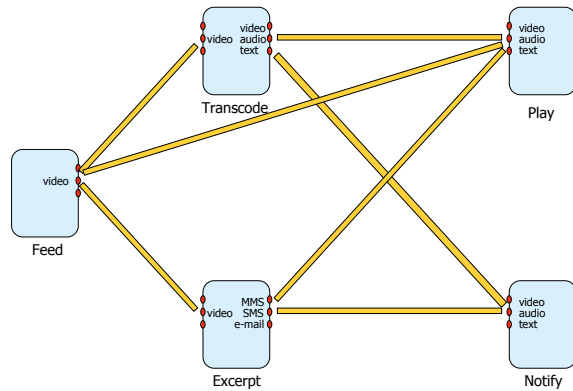


Figure 5. The structure of VideoDelivery service.

During the discovery and binding of VideoDelivery service, Omnisphere sends a query for a service that generates a flow of type video and Feed will reply with its description.

4.2 Service composition

The composition of services specifies which component should be placed between data flows:

```
Service$_i$->Service$_1$, ..., Service$_n$;
```

This construct possibly connects the output flows of $Service_i$ with the input flows of $Service_1, \dots, Service_n$. Note that only compatible data flows can be effectively connected and actual bindings depend on some additional information such as User Preferences, Device Capabilities, and Context.

The composition of services of our example is illustrated in Figure 5. It can be declared as

```
Feed->Transcode, Excerpt, Play;
Transcode->Play, Notify;
Excerpt->Play, Notify;
```

In this way we can express different uses of the service. For example, when the user is well connected, the flow can be directly played on an appliance with sufficient capacities:

```
Feed->Play;
```

When the user is connected via a limited bandwidth wireless link and uses a PDA, the flow should be transcoded:

```
Feed->Transcode;
Transcode->Play;
```

When only using a cellular phone, the user may want to receive video clips:

```
Feed->Excerpt;
Excerpt->Notify;
```

4.3 User Preferences, Device Capabilities, and Contexts

To complete the information needed in the discovery process, we need to specify what are the user preferences, the device capabilities, how to take into account the current context. In general, this information forms a multidimensional space with the following axes: data types, device capabilities, and current context. For instance, we should be able to express the fact that the user prefers to watch any video stream on a large TV screen when at home. It is difficult to linearize this information and encode it in configuration tables in a compact way. After some reflection we have arrived at the definition presented below.

User Preferences provide the information about a particular user in terms of data types, devices and contexts. The profile expresses the priority with which the user prefers to receive a given data type on a given device in a given context. For example:

```
/video TV-screen, notebook, PDA, phone office
/video none commuting
/audio notebook, PDA, phone office
/audio headphones, phone home
/text notebook, PDA, phone everywhere

/mixed/e-mail notebook office, home
/mixed/e-mail PDA travel
/mixed/e-mail speakers commuting
/mixed/e-mail phone otherwise

/audio/music/jazz any airport

/audio speakers private
/audio headphones public
```

If some types are incompatible, such as /mixed/e-mail on speakers, Omnisphere will be able to use a transcoding service to follow user indications.

Omnisphere can associate history with User Preferences to dynamically determine which services have been used in which context. Frequent usage of some services can be detected and added to the profile.

Device Capabilities give more information on formats supported by each device and the service that takes care of a given type on an appliance. In general, this information can be quite detailed: screen resolution, processor characteristics, etc, and we can reuse existing formats such as CC/PP [23]. We present below the information relevant to our example:

```
TV-screen IN
/video:encoding=MPEG2, MPEG4
notebook IN OUT
/video:encoding=MPEG2, MPEG4, H.263, H.261
PDA IN
/video:encoding=H.263, H.261
phone IN OUT
/mixed/MMS, /text/SMS
```

Context provides information about the physical characteristics of a given environment: location, time, date, indoor/outdoor, public/private, devices available nearby as well as a higher level information related to the context of the user: at home, commuting, at work, traveling etc.

The context information may change user preferences or give more details about the choice of actual devices or data flows:

```
office:public
camera OUT /video

meeting-room:public
projector IN /video

home:private
TV-screen IN /video
speakers IN /audio

airport:public
TV-screen IN /video
speakers IN /audio
```

4.4 Service discovery and binding

Once we are able to describe complex services, their components and interconnection, user preferences, device capabilities, and context, we can use all this information to dynamically find the best matching services, bind them together, and activate. *Omnisphere* provides a flexible scheme for service discovery. Figure 6 presents the details of the architecture involved in the service discovery process.

The Omnisphere service runs in the fixed network infrastructure and integrates different service discovery protocols. When needed, it can find services that correspond to the descriptions of component services. The Controller service runs on the appliance to supervise service discovery, binding, and activation. There is a control flow established between the Omnisphere and Controller services. We have defined OCP (Omnisphere Control Protocol) for organizing their interaction. OCP defines the following commands:

- BEACON - Omnisphere advertises itself for nearby appliances.
- REGISTER - the appliance sends the User ID and Appliance ID for authentication.
- SERVICE - if needed, Omnisphere can provide a service by downloading the code using a data channel.

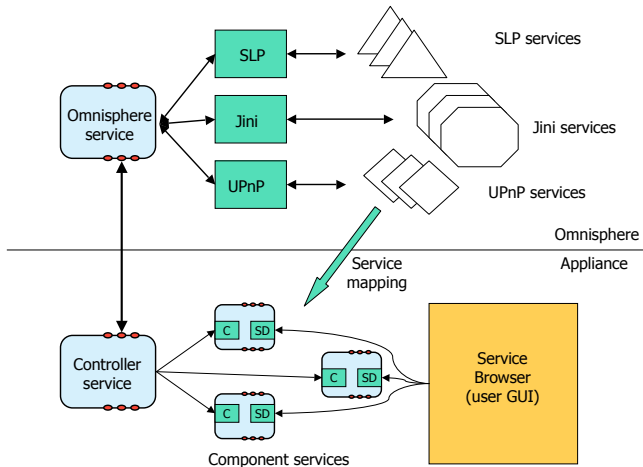


Figure 6. Architecture of service discovery and binding.

- **ACTIVATE** - used by Omnisphere to pass all the parameters needed to establish data flows and activate a service.
- **REBIND** - when the state of an appliance changes, it can ask Omnisphere to find new best matching services.
- **QUERY** - appliance can issue an explicit request for discovering other services (not those that are proposed based on the user preferences).

When an appliance enters Omnisphere, it receives **BEACON** and replies with **REGISTER**. If authentication succeeds, the Omnisphere service discovers the best matching services by using existing service discovery protocols (SLP, JINI, and UPnP in our figure). Some services will stay in the network and some others can be downloaded onto the appliance. Omnisphere provides the service code with **SERVICE** command which may use a data channel for transfer. Then Omnisphere passes the parameters to establish data flows (addresses, ports, applications) with **ACTIVATE** command and activates services on the appliance. The Controller service uses the control interface of services to activate or stop them. The same type of control is performed by Omnisphere service on all services located in the fixed network infrastructure.

The user may have an explicit control over the Omnisphere by running the Service Browser, which is a GUI to Omnisphere. The browser retrieves the descriptions of all services and cooperates with the Controller service to eventually request some new services from Omnisphere.

5 Wireless implementation platform

We have chosen to implement the first prototype of Omnisphere on a wireless platform that includes the following elements:

- **Wireless LAN/PAN.** We use the IEEE 802.11b wireless LAN and Bluetooth. So far, we have developed a QoS support based on DiffServ coupled with efficient micromobility hand-offs between nearby cells [4]. A cell is managed by an Access Router (a Linux box) that takes care of resource allocation. The Access Router uses a 802.11 driver supporting the Master mode so it behaves like a software access point. We have developed a tool allowing to approximately locate an appliance by measuring the signal strength from several Access Routers. We have also experimented with PANs by augmenting the Access Router with Bluetooth cards in such a way that both types of networks can coexist with minimal interference.
- **Wireless appliances.** Our target devices are laptops and PDAs. They will be surrounded by some I/O devices such as cameras, mikes, screens, and projectors as well as other devices such as presence or location detectors (e.g. a key ring detector at home activates Omnisphere when the user returns back home).
- **Service discovery.** The Omnisphere service should be able to discover services published using different protocols. We have chosen three main protocols as the basis for implementation: SLP, Jini, and UPnP. We have chosen XML as the pivot representation of services descriptions, because we can easily map SLP and Jini descriptors to XML. Our descriptors can be mapped into Jini as Entry classes benefiting in this way from the basic Jini matching system. As SLP is based on URLs, we use it directly to find a XML description and process it further when the description downloaded.
- **Active gateways.** We have prototyped an active node able to run services on Linux [17]. Each service processes packets forwarded by the node. A packet filter recognizes some packets according to the information in the packet header and passes them to the right proactive service. Intercepting packets can be activated and disabled dynamically, so that there is no overhead for forwarding packets that do not require active processing. Active processing of packets is efficient—deviating packets to the user space for processing incurs only a slight overhead. Our Access Routers are implemented as Linux active nodes so that we can easily develop some of the needed components such as media transcoders or flow forwarders as active services.

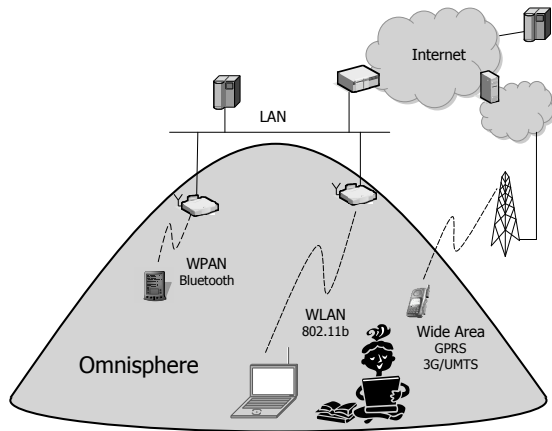


Figure 7. Sketch of the implementation platform.

All these elements give us a good basis for experimentation. Figure 7 presents a sketch of the platform. We consider several scenarios that guide our implementation: *transparent e-mail notification* – the user is notified about e-mail arrival on different appliances when moving through different places (notebook at the office, GSM phone when commuting), *follow me video* – the user can follow a video stream presented on different appliances when moving through different places.

We have already started the implementation of Omnisphere with the service discovery, binding, and activation. We are currently working on the Controller service and the Service Browser.

The first implemented element is the Omnisphere service through which an appliance delegates service discovery to the fixed infrastructure. We have implemented it using Java by using two protocols: SLP and Jini. We have compared its performance with the traditional approach in which an appliance activates discovery itself to find a needed service. We have measured the performance of both approaches using a notebook as a mobile appliance (Intel Celeron 800 MHz, 376MB). It uses a 802.11b WLAN for communication with the fixed infrastructure. In one case, the Omnisphere service runs on one machine and another machine runs the Jini Lookup Service and the SLP's Directory Agent. As SLP can provide only a URL and a set of attribute values, we have implemented a matching algorithm in the Omnisphere service. Figure 8 presents the performance of the two strategies in function of the number of services found. The performance measure is the ratio of the

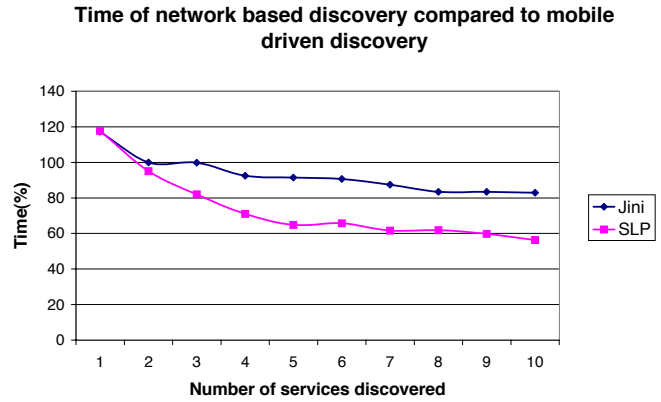


Figure 8. Comparison of different service discovery strategies.

discovery times by the Omnisphere service to the discovery performed by an appliance. The figure shows that for the number of services greater than 1, the discovery time of the Omnisphere approach outperforms the discovery by an appliance: it represents 60% for SLP and 80% for Jini.

6 Conclusion

We believe that the future of the Internet will be dominated by wireless appliances for which we need to invent new paradigms for integrating communications in a similar way to the invention of the WWW. *Omnisphere* presented in this paper proposes a communication and information universe surrounding wireless appliances. It is based on a few high level, yet simple concepts: *ambient services* and *typed data flows*. It has been designed to work in wireless environments connecting devices of different kinds and functionality. Such environments present many constraints: almost all resources (radio channel bandwidth, CPU, memory, and power) are limited. Our strategy for service discovery is to delegate most of the operations to the network infrastructure and to automate them as much as possible. In addition to that, we want our infrastructure to behave as a *proactive system* in which the role of the user is reduced to supervision of a large number of heterogeneous devices that work in an independent way. By specifying service descriptions and matching them with user preferences, devices capabilities, and context information, we obtain a system that follows the needs of the user without an explicit intervention.

We have set up a wireless implementation platform composed of wireless LANs and PANs, various wireless appliances, different service discovery protocols, and active gateways. We have started the implementation of Omnisphere with the discovery service that allowed us to compare the

approach based on delegation with the traditional approach in which a wireless device discovers itself all needed services. Our measurements show that delegating service discovery to the fixed network infrastructure presents substantial performance gains.

Starting from the first implementation, we are working on a complete prototype that includes the Controller service, basic component services and the Service Browser. We also investigate how to support some interesting features such as mobility of services (what happens when the user moves during the operation of a service), disconnected operation (how to mask the effects of the disconnection), and integration of a large number of devices in a home network environment or in networks of sensors.

References

- [1] Bluetooth SIG. Specification of the Bluetooth System. Volume I: Core Specification, February 2001. www.bluetooth.com/dev/specifications.asp.
- [2] S. Gribble et al. The Ninja Architecture for Robust Internet-Scale Systems and Services. *IEEE Computer Networks. Special Issue on Pervasive Computing*, 35(4):473–497, March 2001.
- [3] A. Friday, N. Davies, and E. Catterall. Supporting Service Discovery, Querying and Interaction in Ubiquitous Computing Environments. In *2nd ACM International Workshop on Data Engineering for Wireless and Mobile Access (MobiDE 2001)*, pages 7–13, Santa Barbara, CA, May 2001.
- [4] J. Antonio García-Macías, Franck Rousseau, Gilles Berger-Sabbatel, Leyla Toumi, and Andrzej Duda. Quality of Service and Mobility for the Wireless Internet. In *Proc. First ACM Wireless Mobile Internet Workshop*, Rome, 2001.
- [5] J.V. Guttag. Communicating Chameleons. *Scientific American*, July 1999.
- [6] E. Guttman, C. Perkins, J. Veizades, and M. Day. RFC 2608 : Service Location Protocol, Version 2. June 1999.
- [7] R. Hermann et al. DEAPSpace Transient Ad Hoc Networking of Pervasive Devices. *Computer Networks*, 35:411–428, 2001.
- [8] HP Cooltown Project Webpage. www.cooltown.hp.com.
- [9] A. C. Huang, B. C. Ling, J. Barton, and A. Fox. Appliance Data Services: Making Steps Towards an Appliance Computing World. In *Proceedings of CHI 2001 Workshop: Building the Ubiquitous Computing User Experience*, Seattle, WA, April 2001.
- [10] A. C. Huang, B. C. Ling, J. Barton, and A. Fox. Making Computers Disappear: Appliance Data Services. In *Proceedings of MobiCom 2001*, Rome, Italy, July 2001.
- [11] J. Huang, A. P. Black, J. Walpole, and C. Pu. InfoPipes - an Abstraction for Information Flow. In *Proceedings of the ECOOP Workshop on the Next 700 Distributed Object Systems*, Budapest, Hungary, June 2001.
- [12] E. Kiciman and A. Fox. Using Dynamic Mediation to Integrate COTS Entities in a Ubiquitous Computing Environment. In *Proceedings of the Second International Symposium on Handheld and Ubiquitous Computing 2000 (HUC2k)*, Bristol, England, September 2000.
- [13] M. Esler and J. Hightower and T. Anderson and G. Borriello. Next Century Challenges: Data-Centric Networking for Invisible Computing - The Portolano Project at the University of Washington. In *Proceedings of MobiCom 1999*, pages 256–262, Seattle, WA, 1999.
- [14] Microsoft Corporation. Universal Plug and Play. www.upnp.org.
- [15] SUN Microsystems. Jini Architectural Overview. www.sun.com/jini.
- [16] MIT Oxygen Project. White Paper. www.oxygen.lcs.mit.edu/Publications/Oxygen.pdf.
- [17] H.-B. Nguyen and A. Duda. An Active Node Architecture for Proactive Services. In *Proc. Fourth Annual International Working Conference on Active Networks*, Zurich, 2002.
- [18] M. Nidd. Service Discovery in DEAPSpace. *IEEE Personal Communications*, 8(4):39–45, August 2001.
- [19] C. Pu, K. Schwan, , and J. Walpole. Infosphere Project: System Support for Information Flow Applications. *ACM SIGMOD Record*, 30(1), March 2001.
- [20] Stanford University. Interactive Workspaces Project Homepage. graphics.stanford.edu/projects/iwork.
- [21] D. Tennenhouse. Proactive Computing. *Communications of the ACM*, 43(5), 2000.
- [22] J. Veizades, E. Guttman, C. Perkins, and S. Kaplan. RFC 2165 : Service Location Protocol. June 1997.

[23] Composite capability/preference profiles (cc/pp): A user side framework for content negotiation. W3C Note.