Service Discovery in Ubiquitous Computing: A first step towards sensor-based Context Identification

MINAS PERTSELAKIS AND NICOLAS TSAPATSOULIS

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

NATIONAL TECHNICAL UNIVERSITY OF ATHENS

9, IROON POLYTECHNIOU STR., 15773 ZOGRAFOU, GREECE

Abstract:

In this paper we propose an architecture for modeling the service discovery process of computing devices operating in ubiquitous computing environments. Forming wireless ad-hoc networks requires service discovery protocols that support both proximity based detection and abstract level service matching. In the proposed architecture, primitive services, provided by sensors and actuators, are indirectly located through polling while composite service matching is based on the intelligent agent philosophy. In this way, the proposed service discovery approach, exploits the spread of proximity based device detection and the dynamics and openness of the agent platforms.

Keywords: Ubiquitous computing, service discovery, context awareness

Introduction

Software applications that conform to the ubiquitous computing paradigm are continuously running and always (anytime, anyplace) available [1][2][3]. Personal Digital Assistants (PDAs) and cell phones can be considered computing devices that are able to host such kind of applications, while next generation devices are being designed. Several of them will be portable and even wearable, such as glass embedded displays, watch PDAs, and ring mice. From an architectural point of view, ubiquitous computing applications are made of collaborating parts spread over network components like computing devices, sensors and actuators. The network itself is, in general, an ad hoc
one while network components are characterized by a high degree of heterogeneity since they are products of different vendors. As devices and users move from one location to another, applications must adapt themselves to new environments being able to discover services offered by distributed components and dynamically reconfigure themselves to use these new service providers. Applications, themselves, are characterized by interaction transparency and context-awareness. Interaction transparency means that the human users are not aware that there is a computing module embedded in the tool or device that they are using; it is definitely a non-intrusiveness process. Context awareness refers to adaptation of the behavior of an application as a function of its current environment. A context-aware application can sense the environment and interpret the events that occur within it.

In this paper we describe an architecture for modeling the functionality of ubiquitous computing devices, called intelligent artefacts in our terminology. Special attention is given to the way intelligent artefacts form wireless ad hoc networks with sensors, in order to become context-aware, with actuators, to allow proactivity, and to other intelligent artefacts, so as to build complicated and really smart applications. Forming ad hoc networks requires some kind of service discovery; the peculiarities of service discovery in ubiquitous computing have strong influence to the proposed intelligent artefact architecture. For example, contemporary sensors and actuators lack the ability of implementing general-purpose communication protocols. Instead they support specific protocols through which communicate/receive their data to/from conforming devices. However, in the near future more and more sensors and actuators will support wireless communication protocols and will be Bluetooth enabled; this means that other Bluetooth-enabled devices can detect them based on radio proximity. In the proposed artefact architecture, primitive services, provided by sensors and actuators are indirectly located through polling. The service discovery approach utilizes intelligent agents to implement high-level service discovery and Bluetooth SDP [4] for device detection. In this way exploits the spread of proximity based device detection and the dynamics and openness of the agent platforms.
Service Discovery Protocols

In this Section we briefly examine some of the most popular Service Discovery Protocols and state their deficiencies wrt wireless ad hoc networks.

Salutation [5,6] is a service discovery and session management protocol independent of operating systems, communication protocols, and hardware platforms. Salutation was created to solve the problems of service discovery and utilization among a broad set of appliances and equipment in an environment of wide-spread connectivity and mobility. The architecture provides applications with different services that are scattered throughout the network. It also contains functions to find out capabilities of remote services and provides features for an application to establish interoperable sessions with any remote service. The Salutation architecture defines an entity called the Salutation Manager (SLM) that functions as a service broker for services in the network. Different functions of a service are represented by functional units. Functional Units represent essential features of a service (e.g. fax, print, scan etc). Furthermore, the attributes of each Functional Unit are captured in the Functional Unit Description Record. Salutation defines the syntax and semantics of the Functional Unit Description Record (e.g. name, value). SLM can be discovered by services in a number of ways such as: (a) Using a static table that stores the transport address of the remote SLM, (b) Sending a broadcast discovery query using the protocol defined by the Salutation architecture, (c) Inquiring the transport address of a remote SLM through a central directory server. This protocol is undefined by the Salutation architecture, however, the current specification suggests the use of SLP, (d) The service specifies the transport address of a remote SLM directly. The service discovery process can be performed across multiple SLMs. A SLM can discover other remote SLMs and determine the services that are registered there.

The Service Location Protocol (SLP) [7] is being developed by the IETF Svr-Loc working group and is currently available in Version 2 [8]. SLP aims to be a vendor-independent standard. It is designed for TCP/IP networks and is scalable up to large enterprise networks. The SLP architecture consists of three main components:

- User Agents (UA) perform service discovery, on behalf of the client (user or application);
Service Agents (SA) advertise the location and characteristics of services, on behalf of services; and

Directory Agents (DA) collect service addresses and information received from SAs in their database and respond to service requests from UAs.

When a new service connects to a network, the SA contacts the DA to advertise its existence (Service Registration). When a user needs a certain service, the UA queries the available services in the network from the DA (Service Request). After receiving the address and characteristics of the desired service, the user may finally utilize the service.

Jini technology is an extension of the programming language Java and has been developed by Sun Microsystems [9]. It addresses the issue of how devices connect with each other in order to form a simple ad hoc network, called a Jini “community”, and how these devices provide services to other devices in this network. Jini consists of an architecture and a programming model. The protocol is mostly defined as exchanges of serialized Java objects, mostly via Java Remote Method Invocation (RMI) [10]. The Jini architecture principle is similar to that of SLP. Devices and applications register with a Jini network using a process called Discovery and Join. To join a Jini network, a device or application places itself into the Lookup Table on a lookup server, which is a database for all services on the network (similar to the DA in SLP). Besides pointers to services, the Lookup Table in Jini can also store Java-based program code for these services. This means that services may upload device drivers, an interface, and other programs that help the user to access the service. When a client wants to utilize the service, the object code is downloaded from the Lookup Table to the JVM of the client. Whereas a service request in SLP returns a Service URL, the Jini object code offers direct access to the service using an interface known to the client. This code mobility replaces the necessity of pre-installing drivers on the client.

Universal Plug and Play (UPnP) [11,12], pushed primarily by Microsoft, is an evolving architecture designed to extend the original Microsoft Plug and Play peripheral model to a highly dynamic world of many network devices supplied by many vendors. UPnP works primarily at lower layer network protocols suites (i.e. TCP/IP), implementing standards at this level. By providing a set of defined network protocols UPnP allows
devices to build their own Application Programming Interfaces that implement these protocols - in whatever language or platform they choose. UPnP uses the Simple Service Discovery Protocol (SSDP) to discover services on Internet Protocol based networks. SSDP can be operated with or without a lookup or directory service in the network. SSDP operates on the top of the existing open standard protocols, using the Hypertext Transfer Protocol over both unicast User Datagram Protocol and multicast User Datagram Protocol. The registration process sends and receives data in hypertext format, but has some special semantics. When a service wants to join the network, it first sends out an advertise (or announcement) message, notifying the world about its presence. In the case of multicast advertising, the service sends out the advertisement on a reserved multicast address. If a lookup or directory service is present, it can record such advertisements. Meanwhile, other services in the network may directly see these advertisements as well. The “advertise” message contains a Universal Resource Locator (URL) that identifies the advertising service and a URL to a file that provides a description of the advertising service. When a service client wants to discover a service, it can either contact the service directly through the URL that is provided in the service advertisement, or it can send out a multicast query request. In the case of discovering a service through the multicast query request, the client request may be responded by the service directly or by a lookup or directory service. The service description does not play a role in the service discovery process.

Bluetooth Service Discovery Protocol (SDP) addresses service discovery specifically for the Bluetooth environment. It is optimized for the highly dynamic nature of Bluetooth communications. SDP focuses primarily on discovering services available from or through Bluetooth devices. SDP does not define methods for accessing services; once services are discovered with SDP, they can be accessed in various ways, depending upon the service. This might include the use of other service discovery and access mechanisms such as those mentioned above; SDP provides a means for other protocols to be used along with SDP in those environments where this can be beneficial. While SDP can coexist with other service discovery protocols, it does not require them. In Bluetooth environments, services can be discovered using SDP and can be accessed using other protocols defined by Bluetooth.
**Deficiencies in the existing service discovery architectures**

All the above-mentioned procedures and architectures have been developed to explore the service discovery issues in the context of distributed systems. While many of the architectures provide good base foundations for developing systems with distributed components in the networks, they do not adequately solve all the problems that may arise in a dynamic domain. Moreover, with the exception of Bluetooth, they have not been specifically designed for mobile environments. Some of their deficiencies are summarized below:

*No provision for indirect service discovery.* Most service discovery protocols demand active presence of the available services; the service providers either advertise their services or are able to establish a negotiation channel with the applicant client. Primitive services provided by simple sensors such as thermometers, although useful for context-awareness cannot be accessed by existing discovery schemes even if they are able to communicate their data.

*Lack of rich representation.* The existing service discovery infrastructures lack expressive languages, representations and tools that are good at expressing a broad range of service descriptions and are good for reasoning about the functionalities and the capabilities of the services. Reasoning requires metrics that can be used to evaluate the offered services. The previously described service discovery schemes are satisfied with finding a service only and do not deal with selecting among similar services. Moreover, they do not consider whether the discovered service would be able to serve the requester.

*Lack of inexact matching.* Most existing work supports an attribute-based discovery as well as a simple name lookup to locate a service. Usually there are only a set of primitive attribute types, such as string and integer, to characterize a service. Thus, the service discovery process is primarily based on type matching, string comparison, or integer comparison. For representing real world objects such as network services, it is necessary to have more complex data structures to capture richer semantics. Moreover, a user may not be able to specify the exact values of interested attributes. Thus, approximate matching is desirable.
The Architecture of an Intelligent Artefact

In our consideration ubiquitous computing elements fall into three main categories: **Intelligent Artefacts, Sensors and Actuators**. These elements must be able to cooperate efficiently by exchanging data and services in order to accomplish their individual or collective tasks and define a place for ubiquitous computing applications.

The proposed agent-based architecture, shown in Figure 1, consists of two basic components; the Agent Platform and the Communication Manager. The Agent Platform includes the Service Manager, which is the controller of the system and coordinates the message passing protocol between Clients and Services, and the Intelligent Unit, which executes intelligent tasks -by implementing Composite Services- using available Services, either Primitive or Composite. Intelligent unit’s functions act also as providers of Composite Services to other Clients (e.g. other Intelligent Units).

**The Communication Manager.** The Communication Manager (CM) is responsible for the communication between the sensors or actuators and the Service Manager. It implements drivers for a number of different protocols that correspond to various communication needs.
communication modules, such as one that handles infrared, another that works with Bluetooth, one that works with wireless LAN etc. The Communication Manager talks via a certain socket (ACC) to the Service Manager. This allows CMs and SMs to be located on different systems. When the Communication Manager receives information from a Primitive Service, it sends this information directly to the Service Manager through ACC. When it receives data from a Service Manager, the Communication Manager configures its status, validates the data and sends them to the appropriate network element.

**The Agent Platform.** The Agent Platform (AP) provides an infrastructure in which agents can be deployed. An agent must be registered on a platform in order to interact with other agents located in that platform or in other platforms. AP is divided in two basic parts of operation: the Intelligent Unit and the Service Manager.

*Intelligent Unit (IU).* The Intelligent Unit includes all the operations and software needed to implement the intelligence of the artefact. Its role is to act mainly as a Client requesting Services (Primitive or Composite) from other network elements through the Service Manager. When these Services become available, the IU processes them and produces other Composite Services, which can be just conclusions or actions (actually commands destined to actuators). Each Intelligent Unit may be implemented using intelligent agents and/or algorithms derived from computational intelligence (e.g. neural networks, fuzzy logic, etc. or combinations of them) but this issue is not addressed here.

*Service Manager (SM).* The Service Manager acts as a mediator between Services (primitive or composite) and Clients. When a Service starts up, it has to register with the Service Manager, sending its Registration File. This file contains its name, id and the interfaces it implements. When a new Client comes along, the Service Manager sends to it a Service List Object. This Service List Object changes dynamically, according to the services registered or deregistered within the Service Manager, allowing the Client to have always an updated list of services. Selecting a particular service causes the SM to send the Registration File for that service. The SM then updates its database to reflect that the specific Client is interested in the requested Service. Whenever the SM gets a status update of the Service, it will send it to all interested Clients. The Client will continue to receive status reports from the Service, until it deregisters itself. The Client sends the new Registration File to the SM, after invoking the interfaces of the Service.
On receiving this Registration File, the SM validates the Client and the Registration File. If the Service is still available, the SM sends the Registration File to it; otherwise it is queued for sometime. Once this timeout expires, an error is returned to the Client.

The previously described procedure holds for services that are able to advertise themselves; this is not the case for primitive services provided by contemporary sensors and actuators. In order to address such cases SMs are also responsible for executing indirect service discovery (polling) and leasing. SMs cooperates with the Communication Manager to discover devices offering primitive services using proximity (either via Bluetooth or IrDA). For example, sensors that support Bluetooth are able to answer a radio proximity poll by providing their identity. What they do not provide is the protocol that should be used to access their raw data. It is a responsibility of SM to find this information, either by contacting a database registry or even by letting its human user to do it. In most cases the identity of the polled device provides enough information to infer which protocol should be used so as to access the primitive services of that device. The lease lasts until the device stops to respond to radio proximity polling, executed periodically by SM.

**Forming Wireless ad-hoc Networks**

Using the inter-platform communication protocol and the agents’ properties of mobility and interoperability, proactive and context-aware applications spread over network entities such as intelligent artefacts, sensors and actuators can be formed. For this reason an application level network communications protocol was designed to operate on top of the TCP/IP protocol stack in a transparent way; low-level communication issues are solved by the cooperation of the ACC and the Communication Manager. Given the importance of efficiently utilizing network resources, the protocol supports remote messaging between agents and platforms (e.g. instead of blindly transferring agents from one site to another, a “look before you leap” protocol mechanism is employed which allows an agent at a platform to query the status of another platform before migrating to it). In this way, mobile agents will only be transferred across the network if it is known beforehand that the remote artefact is capable of providing the desired services. It should be clarified that migration of agents it is feasible only between artefacts; sensors and actuators are not supporting agent platforms and thus, are
accessed via the Service Managers of artefacts. Many of the network communications
protocol messages can be directly invoked via the agent command language specified as
part of the mobile agent based architecture.

The usual operation of an intelligent artefact is to create composite services and to
perform specific tasks by requesting and using the primitive services provided by
sensors and/or actuators. This operation includes the following steps: Initially, the
Intelligent Unit is assigned with a specific task. According to this task, IU sends to the
Service Manager a message to ask for the necessary services. The SM then, with the
help of the Communication Manager, returns a Service List Object with all available
Services, ids and states, thus allowing the IU to choose among the offered Services and
start the Client-Service process. If the IU needs more services than these available, then
the SM commences a service discovery process. When a requested not-available-before
Service becomes available the SM updates its Service List and sends it again to the IU,
which reacts accordingly. If a registered Service for some reason seize to exist or to
function properly, the Communication Manager becomes aware of it immediately and
informs the SM to update the registry.

In case an intelligent artefact requires a primitive service that it cannot be found in its
registry, it is able to search for that service in service deposits of nearby artefacts. This
is realized through the Inter-Platform Communication Protocol. In such case the artefact
that requests services acts as a Client Artefact (representing the request of its Intelligent
Unit), while the artefact that may provide the requesting services acts as a Server
Artefact. The Client Artefact sends a message to the Service Manager of the Server
Artefact in order to query the status of the platform and the Services available, before
sending a mobile agent. The Server’s Service Manager responds by sending its Service
List Object and a verification message that the Artefact is able to accept a new Client.
Then, the Client is able to launch a mobile agent into the Server Artefact through the
Agent Communication Channels, following the Inter-Platform Communication
Protocol. After the execution of the mobile agent’s script a connection is established in
order for the Client Artefact to use the desired Primitive Service via the Server’s
Communication Manager, as it is shown in Figure 2.
In another situation, an artefact may be in need to use some of the Composite Services produced by other Artefacts. The case is similar with the previously described one, with one difference: The Communication Manager of the Server Artefact may not be involved in the procedure (see Figure 3); that is the transaction between the two intelligent units can be performed transparently to the lower level communication protocols. For example both artefacts may be connected in a wireless LAN and their communication do not requires the drivers build in the Communication Manager, which mainly handle lower level communication with network elements like sensors and actuators.
Experimental Results

The common approach (typically based on a client-proxy-server model) of wireless/ubiquitous/mobile/ubiquitous computing treats mobile phones and palmtop devices as consumers (clients) of goods or services, while the services or goods come from servers on the wired side. Variants of this approach are now emerging from several commercial companies in the form of systems that allow phone based “microbrowsers” or PDAs to access domain specific internet content or buy goods in a one-way service providing.

Our approach of an ad-hoc network consists of entities with hardware and software components, which automatically become aware of each other, establish basic (wireless) communication, exchange information about their capabilities (e.g. services they can offer) or requirements (e.g. payments they need), and learn to cooperate effectively to achieve their individual and collective tasks. In this framework there are no explicit clients and servers, but peers (see Figure 4), which can be both consumers and providers of different services. The known Mobile Commerce (M-Commerce) thus is transformed into what researchers refer to as Me-Commerce, or more generally Me-Services.

This idea of ”ad-hoc” teams of entities that are dynamically formed to pursue individual and collective goals can be used to create the software infrastructure needed by the next
generation of mobile applications. These will use the emerging third and fourth generation broadband wireless systems, as well as short-range narrow band systems as Bluetooth.

**Simulation**

Current mobile devices have well known inherent limitations like limited power supply, smaller user interface, limited computing power, limited bandwidth and storage space. These limitations necessitate the development of systems that provide mobile users with high quality, precise and context relevant information. It is important that these systems be highly scalable since the demand for service searches will increase in the future.

![Simulation Diagram](image)

Figure 5: Simulation of the proposed network

Our main goal in this emulation was to simulate the appropriate procedures of a service discovery protocol like bluetooth in mobile devices. These procedures include the discovery of other devices and their identities, the exchange of data or services and the establishment of basic communication. For integration purposes, we used two well-known protocols to support the networking infrastructure. The HTTP protocol and the
TCP/IP protocol, by using two cases of operation either over an http server or via a LAN (see Figure 5). In addition, for creating state-of-the-art networked products we developed our model using Sun’s Java 2 Micro Edition (for the Clients) and Java 2 Enterprise Edition (for the Server).

**The Chatroom application.** In order to implement and test the functionality of our network, a chatroom application was developed where mobile devices, as phones or PDAs, can communicate and exchange data or services. The integration of the clients was based on the J2ME - Wireless Toolkit, which includes mobile phones and PDA emulators able to compile and execute J2ME applications. The service provider was integrated in a Server PC with J2EE installed, which was responsible for the interaction among the devices.

In this chatroom application, each user is able to see in real-time how many other clients use the chat service, as also their identity. In addition, each client can choose to communicate with all or one of the other clients. More specifically, the user of each mobile device must fill in his nickname, the name of the server, or the site which runs the server, and the corresponding socket, so as to connect as a client. If the details are correct, the new client enters the chatroom, accepting a welcome message, and he is able to see all other clients using the same server. Then, he can communicate with all of them inserting his message in the appropriate field, or he can choose to chat with one of them inserting in the sending message the target’s id name in brackets. It must also be noted that all clients are constantly aware for other clients entering or leaving the room.

**Implementation Issues**

In order to test the proposed architecture, we set up a wireless ad-hoc network using the following components, as it is shown in Figure 6. The laptop, supplied with a Bluetooth enabled PMCIA card (Xircom PCMCIA Bluetooth), was properly configured to play the role of the Intelligent Artefact, while one PDA and one cellular phone, both Bluetooth enabled, were participated as an actuator and a sensor respectively. The cellular phone is a typical case of devices that can be detected via proximity-based means, while their data are accessed via proprietary protocols, provided by their vendors. PDAs, on the other hand, they can actually support an agent platform, but for the purpose of our realization we consider them as plain actuators simulating actions via
their GUIs. We used the Zeus Agent Platform [14], a Java-based multi-agent environment was created to implement the system architecture. In order to integrate Bluetooth capabilities in these devices, the XJB 100 Bluetooth Host Stack Protocol [15] was implemented, which supports the serial port, generic access and service discovery application profiles for the hosts.

Figure 6: A realization of the proposed architecture

Conclusion

In this work we present an approach that combines agent-based architectures and radio proximity device detection to handle the service discovery problem in wireless ad hoc networks. Service discovery is of high importance in ubiquitous computing where entities like intelligent artefacts, sensors and actuators should cooperate in a consistent way in order to form applications that expose context-awareness and proactivity. The authors, at this stage, utilize existing technologies, modified accordingly, in order to set up the proposed architecture. However, redesign of the service discovery procedure so as to cope with the peculiarities of mobile ad hoc networks is on their near future plan.

Acknowledgment. The current work has been elaborated in the framework of ORESTEIA project (IST-2000-26091/TBD, http://www.image.ntua.gr/oresteia/). The authors wish to acknowledge the contribution of the whole ORESTEIA consortium.

References:


