Communicating Process Architectures 2009
Peter Welch, Herman Roebbers, and Tobe Announced (Eds.)
IOS Press, 2009

JCSP Agents-Based Service Discovery for Pervasive Computing

Anna KOSEK [a], Jon KERRIDGE [a], Aly SYED [b] and Alistair ARMITAGE [a]
[a] *School of Computing, Napier University, Edinburgh, EH10 5DT, UK*
[b] *NXP Semiconductors Research, Eindhoven, The Netherlands*

Abstract. Device and service discovery is a very important topic when considering pervasive environments. The discovery mechanism is required to work in networks with dynamic topology and on limited software, and be able to accept different device descriptions. This paper presents a service and device discovery mechanism using JCSP agents and the JCSP network package *jcsp.net2*.

Keywords. Device discovery, service discovery, CSP, JCSP, agents, jcsp.net2.

Introduction

Ubiquitous computing, also known as pervasive computing, was first described by Weiser in 1991. In Weiser's vision the physically available devices will soon become invisible to the user [1]. The term *ubiquitous* implies that devices in the environment will finally become so pervasive, that they are hardly noticed [2]. Devices from pervasive systems come in various sizes, have different functions and capabilities offering different services to the environment. A *smart space* is a pervasive computing environment in which available computing devices collaborate with each other to assist humans. A smart space will have to provide functions to discover all the devices, learn about different capabilities and services provided by them, couple appropriate devices to perform some tasks and enable communication between them. As in a pervasive system functionalities are distributed in the environment it is desirable to have the device and service discovery also distributed so as not to have a central device that does service discovery and is a single point of failure. Achieving distributed device and service discovery is a challenging task. When considering device and service discovery, all the devices in the smart space are treated as equal, so all of them will have to have mechanisms to:

- Discover other devices,
- Gather information about available capabilities and services,
- Connect to appropriate devices and perform tasks.

The primary goal of a distributed pervasive system is to perform tasks assigned by its user by exploiting resources and services available in the environment [3]. In many cases, performing a task requires more than one device or involves choosing the service or device to use according to some criteria. Service discovery is a very important topic in the pervasive computing domain. Service discovery should be automatic and flexible to users' needs, to make the system as pervasive as possible.

Service discovery solutions like Jini [4, 5] and Salutation [4] operate on centralized or semi-centralized architectures. Jini, Java-based service discovery, uses a Central Jini Lookup Service, where all available services are registered [5]. The Salutation approach to service discovery includes service brokers, called Salutation Managers (SLMs) [4]. All available services are registered in SLMs and clients query SLMs when they require a service. These approaches are suitable for fixed or partially fixed networks, where the existence of at least one central device, to keep a repository, is guaranteed. In the case of failure of the central device, service discovery is impossible. These approaches assume that the network is stable and communication is reliable [3]. In pervasive systems, the network is dynamic; devices are mobile and can appear or disappear from the space at any time. Therefore it is undesirable that devices rely on the existence of other devices to register and advertise services.

The service discovery in pervasive computing requires a distributed approach [3]. One option is Konark, a delivery and service discovery protocol developed by the University of Florida [6]. Service discovery is distributed, all devices run a small version of a Web Service using SOAP messages to request service and exchange information [6]. This approach is more suitable for pervasive systems, but this approach is undesirable for resource constrained devices, because all devices are required to run a Web Service and process XML-based SOAP messages. To get information from a Web Service, the structure of data stored in its database has to be known. Therefore before getting information from a device in the Konark system, the exact format of the available data has to be known. Fixed format of service description presented by a Web Service can be a disadvantage when considering a pervasive system with extendible services.

Universal Plug and Play (UPnP) is another well known standard used for service discovery [4]. UPnP uses Simple Service Discovery Protocol (SSDP) that is based on multicast for service discovery. An UPnP network consists of Control Points and Devices. Devices can be controlled by one or many Control Points. Control Points are responsible for discovering new devices on the network and sending a multicast message (SSDP) requesting a service. This approach is similar to broadcast-based and it generates overhead on the network, as many devices are involved in the processing of the search request. The approach for service discovery presented is this paper is not based on broadcast or multicast and, unlike UPnP model, allows every device to initiate the service discovery mechanism.

In pervasive environments devices should act autonomously when discovering services. Moreover the discovery mechanism should be adaptive to meet the requirements of highly dynamic environments [3]. Scalability is a very important requirement for a pervasive infrastructure [7]. This paper presents two discovery mechanisms: device discovery and service discovery. The device discovery is very simple and based on broadcasting (explained in details in Section 3). The main focus of this paper is an agent-based approach to discover services in highly dynamic networks of devices. Described service discovery is distributed and scalable. Presented mechanism is based on passing a search message between devices, so unlike a broadcast- based approach, it works without utilizing significant network bandwidth.

The usability of JCSP, Communicating Sequential processes [12] for Java, for pervasive environments was described in [8]. The experiment described in that paper focuses on a dynamic connection capability between devices in a smart space. The system described was equipped with very simple service and device discovery. We build on this earlier work, using the same device discovery mechanism, while improving service discovery. The service discovery presented in [8] operated only on a fixed number of device types and was broadcast-based. The service discovery mechanism presented in this paper is not broadcast-based, is more flexible, allows search to be customized and can accept and store any type of description of service offered by a device. The key idea of the presented service discovery technique is to send a “messenger” to gather all information and bring it back to the requesting device. The system uses the same mechanism to gather information about devices and propagate service descriptions to other devices.

This paper presents a CSP based system for device and service discovery in pervasive adaptive systems. Section 1 presents the *jcsp.net2* package developed by Chalmers [9] and its usability for device discovery, Section 2 of this paper describes a software structure used for service discovery; and Section 3 shows the proposed architecture and data structures used in the system. Section 4 provides summary of this work.

1. Device Discovery Using jcsp.net2

Devices described in this paper communicate using TCP/IP protocols and are provided with unique IP addresses. Every device is also called a node on a network, because it is represented by a single IP address. To discover a device and connect to it, the application needs to find its IP address. In this system device discovery is then narrowed to IP address discovery. When the IP address is determined, a JCSP network connection can be established by use of a protocol from *jcsp.net2* [9]. To make a JCSP network connection between two nodes only the IP address and the port number are needed. As the port number can be fixed, the system needs only IP address to connect to a device, and this information can be provided by proposed device discovery. A description of the device discovery mechanism for pervasive environment as in [8] is provided here for completeness.

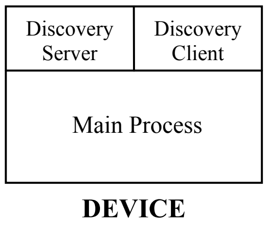

Figure 1. The device architecture.

The device discovery mechanism on every device is connected to the *Main Process* (Figure 1). The *Main Process* represents a set of processes running on a device that are responsible for device functionality (for example a messaging system, as presented in Section 4). The device discovery mechanism consists of two parts: *Client* and *Server*. The *DiscoveryServer* provides a list of available devices. Every available device has a process *DiscoveryClient* that sends an UDP (User Datagram Protocol) packet to other devices in the network. The *DiscoveryServer* receives these packets, this way *DiscoveryServer* is able to determine which devices are running.

Because the *net2* package provides mechanisms to establish network connections just using IP address and port number, only a simple discovery mechanism is needed when constructing a pervasive system. The connection established between devices will be used for a service discovery and performing tasks that needs devices collaboration.

2. JCSP Mobile Processes and Agents

The CSP model is based on processes communicating over channels [10]. The idea of mobility of processes and channels in JCSP was adopted from the π-calculus, further description can be found in [11, 12]. A mobile process is a process that is created on one node and sent to another node. When a mobile process arrives on a new node it can be connected to it and run in parallel with other processes on this node. Once sent, a JCSP mobile process resides on the destination node and it is now part of the network of processes running on that node. The mechanism of creating, delegating and running a mobile process is illustrated on Figure 2.

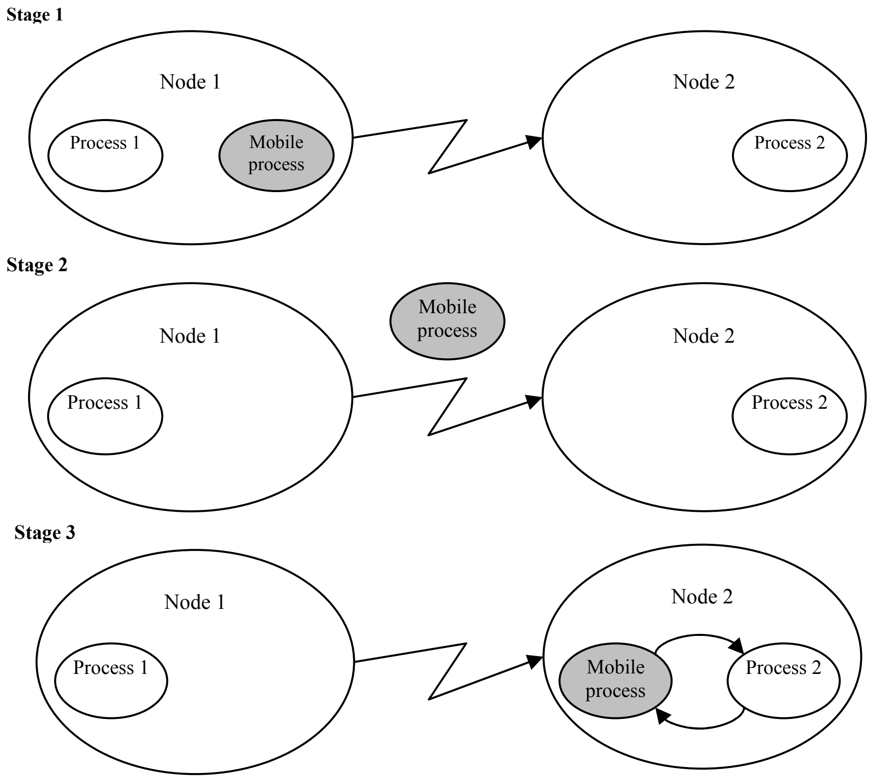

Figure 2. JCSP mobile process.

As shown in Figure 2, *Process 1* creates the *Mobile process* on *Node 1* (Stage 1). The *Mobile process* is sent by a channel to *Node 2* and received by *Process 2*. *Process 2* activates the *Mobile process* and runs it in parallel with all the processes running on the node. Channels between *Mobile process* and *Process 2* are established to enable communication.

A JCSP mobile agent is a mobile process that can be made to visit several processing nodes and undertake some operations [13], e.g. gathering data. After arriving at a node an agent will be connected to the existing network of processes to enable access to host resources. When all operations are completed the agent is disconnected from the node and moved to the next one, and so on. The mobile agent's path can be defined within the agent or it can be decided by a node. An agent can remember a traveled path and gather data from visited nodes.

A mobile agent exists in one of two states: active or passive (Figure 3). When an agent is created it is in passive state. In the active state an agent can carry out instructions, write and read from the node it is connected to. In the passive state an agent can be moved, activated or terminated. After suspension a mobile agent saves its status and when it is reactivated it starts again with the same status. When a mobile agent is terminated it cannot be reactivated. When a mobile agent is activated by another process it starts working in parallel with it, channels between those two processes can be created for communication. Suspension of the mobile agent doesn't have to be requested by the environment, it can be a decision made by mobile agent itself and the reasons for suspension can be complex.

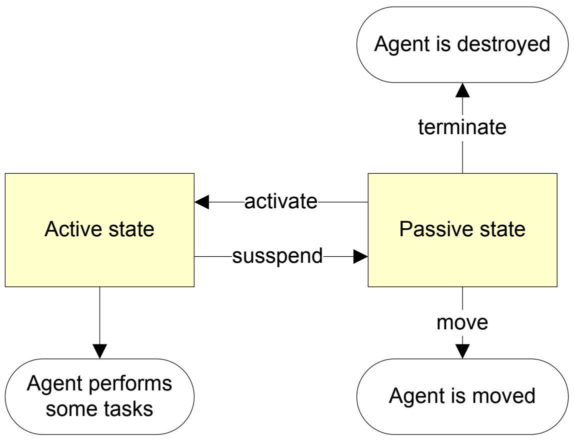

Figure 3. Agent's states.

A mobile agent can visit several nodes, connect to them and exchange information. This way a mobile agent can perform a smart search on nodes that it is visiting and gather particular information. This capability was used for a smart service discovery and is presented in Section 3.

3. Proposed Architecture

Let’s consider a smart space with devices available in it (Figure 4). In a pervasive system functionalities are distributed, so to perform some task, devices have to collaborate. The network is dynamic and devices can appear and disappear. In this system it is essential to discover new device, notice when a device leaves the space and discover new services to enable device cooperation.

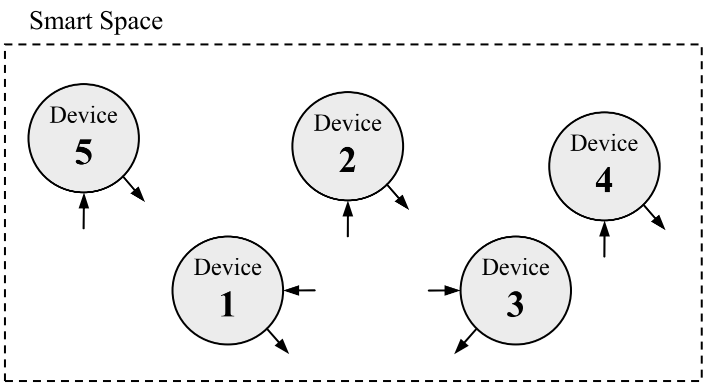

Figure 4. Smart space with devices available in it.

Devices can be distinguished only by their IP address and service information. All devices are able to recognize other participants of the smart space, discover their capabilities and send messages.

The JCSP software running on a device consists of two parts, the *Main Process* and the device discovery mechanism (Figure 5). The *Main Process* is responsible for device functionality: messaging and service discovery. The device discovery mechanism uses *DiscoveryServer* and *DiscoveryClient* to detect existence of other devices.

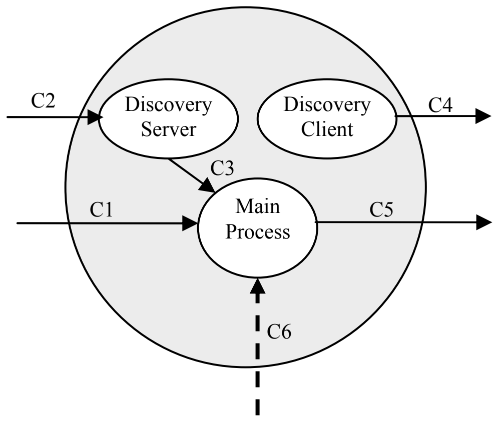

Figure 5. Device architecture.

There are six static channels used in the architecture (Figure 5):

C1 – for receiving messages in the system,
C2 – used to receive broadcast signals from other devices for device discovery,
C3 – for receiving updates about devices available in the smart space,
C4 – used to send broadcast signals to other devices for device discovery,
C5 – for sending messages and agents in the system,
C6 – to receive agents responsible for smart search for service discovery.

Channels C2 and C4 are Java socket based channels, all the remainder are JCSP channels. Channels C2 and C4 are sending simple UDP packets as there is no need to use *jcsp.net2* protocol on top of TCP/IP.

When the agent arrives at a node it is connected using two dynamic channels (Figure 6). Those channels enable communication between an Agent and a Main Process to perform smart search for service discovery.

When a device joins the network the *DiscoveryClient,* part of the device discovery mechanism, starts broadcasting UDP packets to other devices to inform them about its presence. The *DiscoveryServer*, also part of the device discovery mechanism, starts receiving signals from other devices. When a device appears in or disappears from the space, a local list of available devices held by the *DiscoveryServer* is changed. The list is then sent to the *MainProcess* where it is interpreted. The device compares the new list with its own local list, updating or deleting records as necessary.

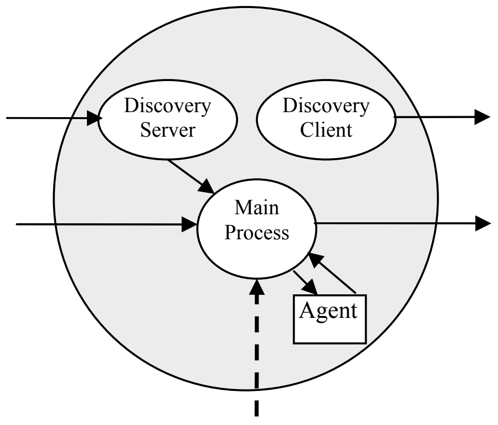

Figure 6. Device architecture with the Agent.

When a device is new in a network, it must inform others about its services and understand what services other devices in the network can offer. To do this the new device creates a list of devices to explore, called *UpdateList*. A new device sends a JCSP agent for a trip using the devices list (*UpdateList*) to inform other devices about its services and to gather information about the other devices in the network. The agent is created in the *Main Process*, and instantiated with the *UpdateList*, so it knows the route for trip it is going on. Next the agent is sent to the first device on the *UpdateList*, where it is connected to the *Main Process* and runs in parallel with other processes on the device. When connected, the agent exchanges data and informs the device of its next destination. The final destination of the agent is the device that created it. Every time the agent visits a node and obtains information about a device, it removes this destination from its *UpdateList*. When the list is empty it returns to the device from which its journey started. When the agent returns it has all the information to update its primary sender and has also informed all the devices from its *UpdateList* about the new device.

4. Example Scenario

Consider a scenario with three mobile devices in a space called a Message Room (Figure 7). Devices are personal computers offering a messaging application. The application discovers all devices that offer messaging and enables communication between people that have the same interests. The application can help to find people of interest to the user and suggest topics that they can talk about. The conversation using messenger can initiate further real-life discussions. Every device holds information about its user. Device 1 belongs to Alice, who prefers to talk about sport. Device 2 is used by Paul, who would like to talk about art. A new device appears in the smart space. Device 3 provides a messaging service for a user called Adam, who would like to talk about art or music. Device 3 is new in the space and has recognised that there are two other devices available.

The device discovery mechanism in Device 3 sends a list of IP addresses to main process in Device 3:

$$UpdateList = \{"134.27.221.10","134.27.221.11"\}$$

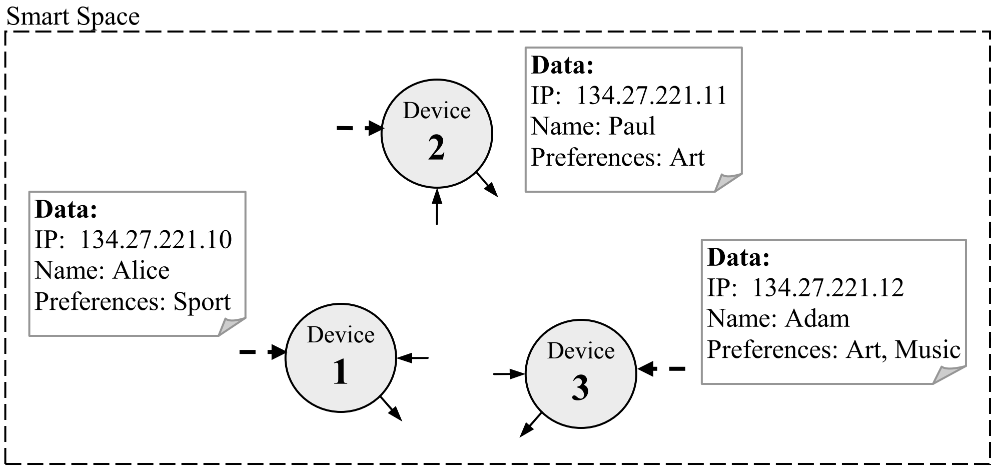

Figure 7. Message Room example.

The order of IP addresses in *UpdateList* is the same as the order of devices discovered by Device 3 in the space. It is possible to add an algorithm to modify the order of devices to reduce routing distance and overall time of service discovery.

Device 3 creates an agent to find information about other devices. The agent is initialized with *UpdateList* and sent to the first destination on the list, the device with IP address 134.27.221.10, Alice's device (Figure 8, Step 1). The agent is connected to Main Process from Device 1. The Agent then removes IP 134.27.221.10 from its *UpdateList* and requests data from Device 1 (Figure 8, Step 2).

Next the Agent decides where it should be sent and consults Main Process if the next destination from the *UpdateList* still exists. The Device 1 uses its resources and capabilities to send the Agent to the Device 2. In Step 3 the Agent is sent to Device 2 and connects to it to gather information about its user and introduces itself to the device (Figure 8, Step 4). At this point the Agent's *UpdateList* is empty, so the Agent requests that Device 2 send it to its home node (Step 5). When reaching the final destination the Agent attaches to the Main Process of the device, namely Device 3 and transfers all gathered information.

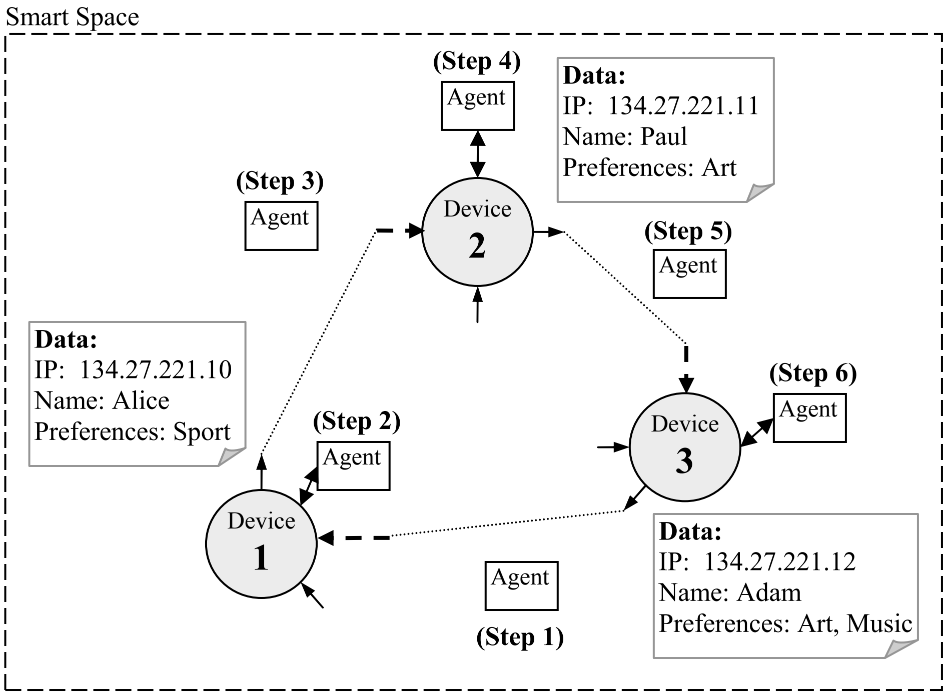

Figure 8. Agent's trip.

To simplify the implementation the service discovery data gathered by the system has a fixed format as follows:

Data={*Service, IP, Name, Preference1, Preference2, Preference3*}

For example, the data received by the Agent from Device 2 is:

Data={"Messaging", "134.27.221.11", "Paul", "Art", "", ""}

In this example scenario the format of the data gathered by an agent is fixed, but an agent can gather data of different types, depending on what is revealed by the device that it visits.

After the Agent returns to its home node, all devices that appeared on *UpdateList* are now informed about user Adam and that he prefers to talk about art or music. Adam is informed about available users, their names and preferred topics. Now depending on Adams's choice, a connection between him and particular users can be established to start conversation. Every time a new Agent arrives at a node, the device's User Interface (UI) is updated with the new user's name and conversation preferences.

All discovery agents are built using the same agent class. There can be many agents in the system, because many devices can send an agent to discover others. In this implementation the only difference between agents is data that they hold; this data and network state determines their behavior. This way all agents can adapt to network topology and offers flexible service discovery.

Initial experiments were undertaken using the Java programming language and JCSP libraries [15]. Devices are composed in a network and run Java code, communicating over a TCP/IP network provided by a wireless router. Every device is a PDA (Personal Digital Assistant) Dell Axim X5 with Microsoft® Pocket PC operating system, an Intel® PXA255/400MHz processor, RAM capacity of 64MB and IBM J9 Java Virtual Machine.

Table 1. Size of the system

Application classes:	23.22 KB
Additional JCSP libraries:	628.00 KB
Total:	**651.22 KB**

The choice of PDAs for the experiment was made based on equipment availability. A PDA may seem a big device, but the system was kept to a small size (Table 1). The system requires a JVM to run Java code. In the JVM a process is represented by a thread. Limitations of the equipment can influence the number of threads that can be run on one JVM, and this number can be further decreased by size of a thread. A JCSP system controlling a LEGO NXT robot described in [14], due to VM and device memory limitations, could only run 90 simple threads. Experiments for the project described in this paper were first simulated on PC and then performed on PDAs, so the minimal requirements for the equipment that can run the system were not tested.

The initial experiment used the *jcsp.net* package [15] for network connections. The device discovery was performed with the use of central repository, as a capability provided by a new protocol for *jcsp.net2* was not available. The second version of the implementation uses distributed device discovery presented in [8]. The service discovery mechanism using JCSP agents remained the same for both implementations.

In the device discovery mechanism the frequency of sending packets was determined by experiments. The aim was to minimize the reaction time for device failure. In the actual implementation a UDP is sent over a Java socket and has a size of 9-bytes. The frequency of sending UDP packets is every three seconds. A *DiscoveryServer* holds a constantly updated list of available devices. To discover a new device the *DiscoveryServer* needs just one packet with IP address that does not exist in its local list. Therefore it takes about three seconds plus a network latency to discover a new device. To detect disappearing devices *DiscoveryServer* has to wait for longer time, to allow all active devices to report. In existing implementation the time to wait for devices to report depends on number of devices that already have been discovered. A margin of few seconds is added to this calculation, because reversing an incorrect decision takes considerable time. Once *DiscoveryServer* determines that the device left the space, the *Main Process* is informed and all information about leaving device and connections are discarded. If the *DiscoveryServer* was wrong, all connections will have to be reestablished and service discovery undertaken again.

5. Issues and Further Work

The system was tested on up to ten PDA devices and deals with devices entering and leaving the space. The service discovery can adapt to changing topology, using mechanisms implemented in an agent. In this implementation an agent is sent only once to discover devices capabilities in the network. The agent can be dropped if the home destination does not exist anymore or if the device fails while dealing with the agent. If the agent is lost or dropped before visiting all nodes that it was sent to, some of the nodes may not have information about the new device. This problem can be solved by setting a timer for the agent to return to its home node. If this timer expires, the device can send another agent for service discovery. This agent will have to visit all nodes again and if it finds the same problem as the predecessor it will be dropped, which can lead to infinite loop of sending and dropping agents. If the agent returns with partial information about the network, a new agent can be sent directly to unexplored region to discover available services.

The service discovery is designed to send agents only from new devices to avoid repetition of gathered data and flooding network with agents that perform the same task. The agent sent from a new device not only gathers data but informs visited device about services offered by the sender. If an agent is lost or dropped existing devices may not receive information about the new device since service discovery deployed in those devices has no mechanisms for finding this information. A solution for this problem can be exploring the data that is held in the arriving agent, this way the device can get information about nodes that the agent has visited. However, if the agent is implemented so it gathers only specific information, the information extracted from the agent might be useless for this particular device.

It is possible that devices may loose or gain new services. In this case, the device acts like a new device in the network and sends an agent offering new services. Services are revoked or expanded by overwriting their descriptions and informing all devices in the network.

The service available in the presented implementation is to provide information about a person that uses a particular device. In the presented implementation the information that the agent is obtaining from every node has a fixed description. To extend the solution, the information about the service can be described using XML format or very popular in Web Service Discovery domain OWL-S (Ontology Web Language, Semantic Markup for Web Services) description [16]. The description provided in OWL-S can be quite flexible, so different types of services can be expressed, even complicated services like electronic payment.

Despite all the problems associated with loosing and dropping agents or getting only partial information, the use of agents provides a flexible solution for service discovery in networks with dynamic topology. The agent is initialized with list of devices and decides where to be sent; therefore it will not be sent to a device that appears to be switched off. The journey plans stored in the agent can be changed and easily fitted to the dynamic environment. The agent can also perform a smart search; it might only gather specific types of information, so the issuing device will receive only the requested information. The agent can reveal different information about its home device depending on the type of the device visited or a trust level. For example, an agent might carry detailed information about the issuing device, but reveal some parts of it visited devices, depending on rules that it has adopted.

6. Summary

This paper shows a new approach for distributed service discovery. The usability of JCSP as a software architecture for pervasive computing was described in [8]. The device mobility and service distribution is desirable in pervasive system. We show that service discovery can be achieved with JCSP mobile agents. JCSP agents, which are an extension of mobile processes, can move around the system, connect to nodes and exchange data. In this work these agents were used for to implement a smart search capability for service discovery. The JCSP agent, as a software structure, can be used to perform different functions, e.g. to represent person or object physically moving in space or to model mobile social networks [13].

Implementation of the system presented in this paper with use of JCSP libraries on PDA devices has been done as proof of concept.

References

[1] Weiser, M., The Computer for the 21st Century. Scientific American, 1991: pp. 66-75.

[2] Coulouris, G., J. Dollimore, and T. Kindberg, Distributed Systems: Concepts and Design. 2005: Pearson Education.

[3] Chakraborty, D. and T. Finin, Toward Distributed Service Discovery in Pervasive Computing Environments. IEEE Transactions on Mobile Computing, 2006.

[4] Bettstetter, C. and C. Renner. A Comparison Of Service Discovery Protocols And Implementation Of The Service Location Protocol. in 6th EUNICE Open European Summer School: Innovative Internet Applications. 2000.

[5] Gupta, R., S. Talwar, and D.P. Agrawal, Jini Home Networking: A Step toward Pervasive Computing. IEEE Computer Society, 2002. pp. 34-40.

[6] Helal, S., Desai N., Verma V. and Lee C., Konark - A Service Discovery and Delivery Protocol for Ad-Hoc Networks. Wireless Communications and Networking, 2003.

[7] Henricksen, K., J. Indulska, and A. Rakotonirainy. Infrastructure for Pervasive Computing: Challenges. in Workshop on Pervasive Computing INFORMATIK 01. 2001. Vienna.

[8] Kosek, A., Kerridge, J., Syed A. and Armitage A., A Dynamic Connection Capability for Pervasive Adaptive Environments Using JCSP. Artificial Intelligence and Simulation of Behaviour (AISB) 2009.

[9] Chalmers, K., Investigating Communicating Sequential Processes for Java to Support Ubiquitous Computing, PhD thesis in School of Computing. 2008, Napier University: Edinburgh.

[10] Hoare, C.A.R., Communicating Sequential Processes. 1985: Prentice Hall International Series in Computer Science.

[11] Chalmers, K. and J. Kerridge. jcsp.mobile: A Package Enabling Mobile Processes and Channels. in Communicating Process Architectures, 2005.

[12] Chalmers, K., J. Kerridge, and I. Romdhani, Mobility in JCSP: New Mobile Channel and Mobile Process Models. in Communicating Process Architectures, 2005.

[13] Kerridge, J., J.O. Haschke, and K. Chalmers, Mobile Agents and Processes using Communicating Process Architectures. Communicating Process Architectures, 2008.

[14] Kerridge, J., A. Panayotopoulos, and P. Lismore. JCSPre: the Robot Edition to Control LEGO NXT Robots. in Communicating Processes Architectures. 2008. York, UK.

[15] Welch, P.H. and P.D. Austin, The JCSP Home Page. http://www.cs.ukc.ac.uk/projects/ofa/jcsp/.

[16] Martin, D., et al. OWL-S: Semantic Markup for Web Services. 2004; Available from: http://www.w3.org/Submission/2004/SUBM-OWL-S-20041122/.