A Dynamic Connection Capability for Pervasive Adaptive Environments Using JCSP

Anna Kosek[1], Aly Syed [2], Jon Kerridge [1] and Alistair Armitage[1]

Abstract. The house, office or warehouse environment is full of devices that make users' life and work easier. People nowadays use personal computers, laptops, Personal Digital Assistants, mobile phones and many more devices with ease. The mechanism to connect, enable co-operation and exchange data between devices will help to use devices' full capabilities. This paper investigates the usability of Communicating Sequential Processes for Java in pervasive systems and the adaptation possibilities offered by this environment. It focuses on dynamic connection capabilities. The paper also describes an experiment that organizes an adapting pervasive environment which uses dynamic connections for data flow.

1 INTRODUCTION

The number of devices surrounding us increases every day. The devices that we use have become more complicated and have more capabilities and functions. A mobile phone is now not only used to make calls, but also to store and edit files, take photos and videos, play music, browse the web and much more. Some of those functions are repeated in many devices. Most of the devices that we use are autonomous and making them co-operate with other devices is often difficult and time consuming.

The mechanism to connect devices, make them co-operate and use each other's functions in a simple and transparent way would help to manage a network of devices used every day. The method of making many computers physically available, but effectively invisible to the user is called ubiquitous computing [1] or pervasive computing. The concept was introduced by Weiser in 1991 and many researchers have been investigating it since.

Making connections is a crucial capability when considering a network of devices. Dynamic connection requires device and service discovery. A device arriving in an unknown network has to have mechanisms to adapt to the environment and discover distributed capabilities that can be useful. The same device in a different location can work differently depending on services that are available in a particular space.

This paper describes dynamic connection capabilities for pervasive adaptation using Communicating Sequential Processes for Java (JCSP) [2]. In Section 2 the background to the research is provided. In section 3 pervasive software requirements are presented and compared with JCSP properties. Section 4 describes JCSP channels in more detail. An experiment using dynamic connections for synchronization is presented and evaluated in Section 5 of this paper. Conclusions and future work are stated in Section 6 of the paper.

2 BACKGROUND

The background section of this paper presents the research areas of pervasive computing, ad-hoc networks and adaptation. Fundamental concepts of CSP and a simple example of a CSP based system are also described.

Pervasive (Ubiquitous) Systems

The ubiquitous computing concept is an approach to making many computers physically available, but effectively invisible to the user [1]. Devices in the system vary in size, capability and function, each suited to a different task. The term 'ubiquitous' suggests that devices will finally become so pervasive in everyday objects, that they are hardly noticed [3]. Ubiquitous computing is also called pervasive computing [4].

Devices in pervasive computing systems are enabled with communication capability and are designed to be useful in a single environment such as home or hospital. The idea is to integrate all the devices that are connected to the network to co-operate and use each other to achieve an expected performance and capability. In this system computers are no longer tools but assist humans in their everyday activities. All the computers in the environment will not be considered as autonomous but parts of a larger system that is targeted to users' needs. Devices in pervasive system have to be aware of their location and surroundings. If a computer knows its position, it can adapt its behavior to the environment [1].

Network Structure

The intelligent environment should work with any set of devices, and enable collaboration of available devices depending on their capabilities. In the event of a device failure, the system should reconfigure itself and continue to work. A central control workstation or remote control device is not the goal of a pervasive system. It is likely that a mixed control system will be required that is neither fully distributed nor fully centralized but the particular mix of control function needs to be determined and may vary with the application environment.

The pervasive adaptive environment is often a mix of fixed and mobile devices. Some devices can stay in the environment for a long period of time; some can visit a specific space only briefly. As a central control system is not present, devices in the network have to monitor the network topology. These kinds of networks are called ad-hoc networks and defined as a collection of mobile and fixed devices communicating over wireless links [5]. A mobile ad-hoc network should be a set of devices that recognize each other, decide about inter-network connections, organize a virtual topology, exchange and use resources and capabilities.

Devices Types

A pervasive adaptive environment can consist of different types of devices. Equipment varies from very small devices like sensors, capable of sending only a precise type of signal or FPGA's programmable for specific tasks, to PDA's and powerful stationary and mobile computers. Rapid technology advances affect device configuration, therefore, it is difficult to pinpoint what constitutes a big or a small device, but for the sake of argument we define a small device to have 100 MHz processor and less than 64kB RAM. A medium device is defined to have 200-400 MHz processor and less than 100MB RAM. A large device is considered to have more than 400 MHz processor power and more than 100MB RAM. Some parts of the pervasive adaptive system can be very simple and be based on primitive signals, other parts must be powerful enough to run algorithms associated with learning and analyzing data. Therefore communication capability must be based on uncomplicated data structures and acknowledgments, so a wide variety of devices can communicate.

Adaptation

The ability to adapt is an important requirement for a pervasive environment. New services, functionalities, interaction mechanisms or devices can be added to the pervasive system requiring them to be adapted to the specific characteristics of the environment [6]. Mobility of devices makes a system's network topology even more dynamic. Mobile devices have to be able to detect change of location and exploit knowledge about their current situation; this is called location-awareness [7]. All mobile devices have to detect and react to the environment and therefore adapt to a new space to improve quality of communication. On the other hand existing elements from the system might be adapted to a user's requirements or changing conditions in the environment [6]. This characteristic of pervasive systems is called context-awareness.

Adaptation should be supported by a flexible and dynamic system. Dynamic communication is a very important capability for pervasive adaptation. When a mobile device arrives in a new location communication has to be established. Depending on the situation and devices present in this environment connections have to be created or destroyed. The ability to manage connections between many devices in an intuitive and transparent way is very useful for pervasive adaptation.

Concurrent Systems

A system is called concurrent when there is more than one process existing at a time [8]. A concurrent style of programming enables the creation of a system with processes working in parallel. One of the key aspects of parallel system design is that simple processes can be composed into larger networks. The concurrent style of programming better reflects the nondeterministic environment surrounding us and can be used to design and construct pervasive adaptive systems.

CSP

Communicating Sequential Processes (CSP) is a formal language consisting of mathematical models and methods to construct component processes to interact with each other by communication [8] and provides a mathematical notation for concurrency theory [9].

CSP applications are process-oriented. A system consists of processes that are sequences of instructions. Processes run separately and can communicate with other processes using channels [10]. Processes can work on different processors or even communicate across different devices. A channel is a point-to-point connection between two processes [10]. A simple version of a CSP channel consists of an in-end and an out-end performing a one-way communication. One process writes to a channel and the other reads from it. To establish a connection it is necessary to have at least two processes, one process is connected to an in-end of the channel and the other process is connected to the out-end of the channel as is shown in Figure 1.

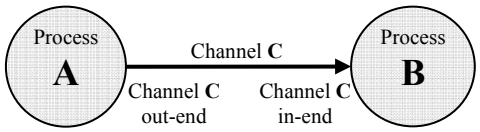

Figure 1. A simple example of a CSP network diagram.

In Figure 1 a channel is represented by an arrow to show the direction of communication. If bidirectional communication is needed an additional channel in the opposite direction should be added. Channels are unbuffered, so process A writes to the channel C and waits in an idle state for process B to be ready to read. In this way channels are unbuffered and communication occurs only when both processes are ready [10]. This simple channel is constructed in a way that data sent over it cannot be lost, provided the underlying message transport layer guarantees message delivery. In CSP based systems there are no buffers needed in the channels. Therefore the size of buffers becomes a property of a device that needs them for its own data handling, which makes system design simpler and more robust, and makes it easier to calculate memory needs.

A CSP process can have many channels connected to it. With the CSP 'alternative' programming structure it is possible to distinguish from which channel communication appeared. Therefore alternative captures non-deterministic channel behavior and permits selection between one or more input communications.

Summary

This section described a basic background of the research area. The fundamentals of ubiquitous computing, adaptation and ad-hoc networks have been described. CSP concepts have been explained as an introduction to Section 3 of this paper. A pervasive adaptive system presents many challenges and, to categorize and understand them better, an infrastructure defining software requirements is needed.

3 INFRASTRUCTURE FOR PERVASIVE SYSTEMS

Software requirements for pervasive computing were described in [11]. The software environment appropriate to support pervasive system must sustain application requirements such as:

mobility and distribution, adaptation, interoperability, component discovery, development and deployment, scalability and context awareness [11]. In this section we will describe those requirements and explain how CSP for Java (JCSP) [2]can be used to achieve pervasive system goals. JCSP is an environment allowing the development of Java applications following CSP principles. Use of Java based language enables executing processes on any device that can run a Java Virtual Machine (JVM).

Mobility and Distribution

Mobility and distribution are a natural requirement for a pervasive system. A pervasive environment is highly mobile, as users and devices can change their location; it is required for software to be mobile and distributed. Mechanisms associated with this requirement should be deployed in software and transparently for component developers [11]. Mobility should be achieved without thinking about synchronization or data migration [11]. Software for pervasive systems should have those mechanisms already deployed.

Π-calculus, a calculus for communicating systems, is a model for ubiquitous computing that proposed formal mobility constructs [12]. JCSP, following π-calculus model, offers mobility of processes, channels and code [13-15]. Process mobility enables creating processes and sending them over a channel. A mobile process can be connected and run in parallel with different processes on the node it was sent to. Mobility of channels enables sending channels with processes without a need of recreating the connection, therefore the input or output end of the channel changes its location [14]. Code mobility enables sending appropriate Java classes that are used by mobile process but are not present in a new location. Mobility in JCSP is presented and fully explained in the papers [13-15].

Adaptation

Adaptation in a pervasive environment is not only the ability to react to a changing environment but also to users' intents and needs. The pervasive infrastructure should apply adaptation to individual software components [11]. Depending on situation, some bindings should be reconfigured by adding, removing or replacing components [11]. Those processes can be reconfigured depending on adaptation techniques.

One of the main advantages of CSP system design is that simple processes can be composed into larger networks [10]. In computer science this approach is called separation of concerns [16], where concerns can represent features or functionalities of the system. In this design approach functionalities are implemented separately and composed into a larger architecture. As components of the system with different functionalities are implemented separately, the system can be more easily reconfigured by reordering and reconnecting processes, or by adding and removing processes. This type of adaptation is also supported by JSCP mobile processes and channels by moving components of the system between different devices.

Interoperability

A pervasive infrastructure is required to integrate diversity of components programmed using different languages into an infrastructure that can successfully interact and cooperate [11]. Integration of different components is a very difficult task and has been researched in Component-Based Software Engineering [17]. The ability to unify the connection mechanism would be very useful when considering a network of different components.

JCSP is equipped with network capabilities using the Communicating Process Architectures Universal Network Protocol (CPAUNP) to communicate between processes [18]. The protocol enables communication independent from the data being sent, and work towards a standardized mechanism for connecting various CPA based network architectures is in progress [18]. In particular, the protocol enables communication between Java and non-Java based devices and the simpler devices do not have to implement a complete parallel environment.

Component Discovery

A component and service discovery framework is used to organize dynamic client-server applications [19]. There are many approaches to component and service discovery [19-22]. One approach is to have a centralized repository that keeps track of all the devices and services available in the network. Any member of the network can register itself with a repository. For example JINI is a Java-based environment [21] that provides a set of application programming interfaces and network protocols for service discovery [22] and is supported by a local repository Central JINI Lookup Service. This approach is suitable for wired networks where there are always at least one device remaining in the network making the repository available for new devices [20]. In ad-hoc networks, the topology is dynamic and devices have to perform their tasks with any set of equipment. There is no central control and assuming that the repository is always available would only make an ad-hoc network less flexible. A device that performs service discovery would have to find a repository server every time it visits some network. Typical devices in ad-hoc networks can have small memory capabilities, storing information about all needed resource servers would be difficult.

Use of a distributed approach is a solution that would better fit ad-hoc network characteristics. Konark is a service discovery and delivery protocol developed in University of Florida [20] that enables integration of ad-hoc network of devices using a distributed peer-to-peer mechanism that enables devices to advertise and discover services available in the network. Service discovery is based on Web Server mechanisms using SOAP messages; therefore all the devices are using a small version of a HTTP server [20]. This distributed approach is more suitable for ad-hoc networks, but still unsuitable for devices with small memory capabilities.

In JCSP, component discovery can be performed in a simpler way. For a device and service discovery the most important issue is establishing connection. In JCSP to create a connection between two nodes only the IP address and port number are required, therefore device discovery amounts to finding those two properties. This can be performed using simple, low level sockets and broadcast capability for a local network. This is explained in Section 5 of this paper. When this information is available, the next stage to perform service discovery is to exchange information about capabilities and functions deployed in devices. This approach relies on the dynamic connection capabilities in JCSP, which are the main focus of this paper and described in Section 4 and 5.

Development and Deployment

Components in pervasive systems are required to adapt to changing environmental conditions that requires the ability to redeploy and adapt at a runtime, without restarting devices or installing new versions of components. Rapid development using multi-agent systems can be a solution to reconfiguration problems [11]. An agent is a physical or virtual unit that can identify its environment and communicate; an agent is autonomous and has the ability to accomplish tasks and achieve its goals [23].

The interpretation of agent ideas in JCSP is presented in paper[24]. The mobile agent, based on the actor model [25], is a unit that is able to move around the network, connect to some or all of the nodes and perform some pre-defined tasks. A JCSP mobile agent is a specialization of the mobile process [24] described earlier in this paper. JCSP is a framework that can offer capabilities for rapid development using an interpretation of multi-agent systems.

Scalability

The number of devices and users in a pervasive environment is not limited. Therefore the number of interactions increases and also the number of devices increases. As a result scalability is a problem for pervasive systems [4].

CSP usability for a complex emergent system was described in [26]. The study was to model artificial blood platelets using CSP concepts. This software technology scales to millions of processes per processor [26], which shows that CSP based systems are capable of dealing with scalability trivially. Since in CSP processes offering some services are composed using channels into a system that offers a natural separation of concerns at the level of services. Another advantage is that composing new system using existing components becomes easier and maintenance requires less effort.

Scalability also depends on device capabilities, and hence a limitation of the pervasive system. In dynamic adaptive pervasive system the number of threads can vary depending on a situation. A JCSP system controlling LEGO NXT robot was presented in paper [27]. In JVM a process is represented as a thread. Due to JVM and device capabilities limitation the number of simple threads was restricted to 90. The number of threads on a JVM is limited by the memory used for a thread [27]. As a result of this, if the system runs on a limited device, only limited operations can be done.

Context Awareness

Another pervasive software requirement is context awareness. Transparency of the application can be achieved by enabling a pervasive system to make decisions based upon context taken from environment and user inputs [11].

A context aware system has to take into consideration many signals from different sensors and services providing information [11]. As the nature of the environment is nondeterministic, components from the pervasive system have to be able to make nondeterministic choice between received signals.

In JCSP a nondeterministic choice is performed by alternating upon channel inputs. JCSP based systems can wait for many inputs and trigger actions according to signal received. The concurrent nature of a system based on channels and processes simplifies sending and receiving signals.

Context awareness is also associated with adapting to changing environmental conditions. The topology of the environment is also dynamic because of mobility of devices. Pervasive devices should recognize changes in the environment and adapt to it. The JCSP dynamic connection capability is a main subject of this work.

Another important issue when considering pervasive and mobile environment is energy management. Most of mobile devices are battery charged and saving energy when device is not in use would be an advantage.

CSP based processes are designed to stay in idle state when they wait for a communication from other processes and use no CPU power. Only a communication from particular channel, which process waits for, can wake it up. This way many processes in CSP based system wait for a communication form a channel or a set of channels in a state that consumes less energy than programs that would continuously send signals to sender to check its availability.

Summary

The JCSP features, described in this section, are useful when constructing a pervasive system. Mobility and distribution are supported by mobility of channels, processes and code. Adaptation techniques can be applied with ease by reconfiguring a JCSP system by reordering and reconnecting processes. Interoperability may be achieved by using the Communicating Process Architectures Universal Network Protocol (CPAUNP) between different CSP based components. Development and deployment can be accomplished by using JCSP dynamic connection capabilities and mobility of components. Scalability was shown on an example of a CSP based system presented in [26]. Context awareness by adapting to environmental conditions can be resolved using JCSP dynamic connections, which is the main focus of this paper and described in the following sections.

4. JCSP CONNECTIONS

JCSP offers local and network channels. Local channels are used to connect processes working in one device, so are executed in one JVM. This kind of channel is based on CSP principles. Network channels are used for communication between nodes on a network. JCSP connections can be divided into two categories: static and dynamic. This section introduces JCSP network channels and dynamic connection capabilities.

JCSP Network Channels

Network channels enable communication between two network nodes. Communication that uses network channels is based on CPAUNP that was developed and verified at Napier University[28]. To establish connection between two nodes only an IP address and port number of the remote node are required.

Managing the connection is easier if it is possible to check the state of the channel. The JCSP networking package enables this at the application or user process level. If the connection is lost the process trying to send data is informed, so new connections between processes can be established. If the connection was already established, the process trying to create it again is informed by the mechanisms embedded in the network architecture.

Dynamic JCSP

The JCSP package [29] enables dynamic creation of the basic components of CSP based systems: processes and channels.

Dynamic Channel Creation

In JCSP dynamic channels can be created if they are needed to establish a new connection between two processes. Both local and network channels can be dynamically created. The number of dynamic channels on a node can be large, with this number of connections it is possible to dedicate one connection to one task or type of signal (e.g. turn on/off a light). This way we can establish many connections from one node or process to another. Therefore control over many functions in the device can be performed using simple non-deterministic alternation on channel inputs. Dynamic channel creation is a very useful capability when establishing a connection between devices that know very little about each other. The connection will help devices to explore each other's potential.

Dynamic Processes Creation

JCSP libraries enable dynamic creation of an instance of a process that can be activated and connected to a network node using dynamic channels. A new process can be initialized with some data, then perform a defined task working concurrently with processes on the node. The process can be disconnected and destroyed when its task is finished. JCSP also enables dynamically created processes to be mobile[13], which makes JCSP based systems even more dynamic.

Verified Model

Concurrent systems can cause many problems, and it is necessary to have a good understanding how they behave and how to accommodate non-determinism and avoid livelock and deadlock during the design stage of the system [10], especially with systems that have dynamic topology. A tool for software verification would be useful for testing purposes and improve software reliability.

Spin is an open-source software tool that can be used for formal verification of concurrent and distributed software systems [30]. Spin can be used both as a simulator or verification tool. Spin model is a logic model checker, and accepts a specification language called Promela (Process Meta-Language). Spin is similar to JCSP model, so processes and channels are included. Importantly, Spin incorporates a concept of mobility in its design that makes its use more appropriate for the mobile distributed systems that will occur in pervasive adaptive systems. Moreover a Spin model can verify a system as a whole, including dynamic and static sections. This capability is useful when checking if a system with dynamic topology is deadlock and livelock free.

Summary

A pervasive adaptive environment can be built using components of a system working concurrently. Therefore a CSP model can be used when building a pervasive adaptive system. As deadlock and livelock can be avoided by design and the software can be verified using a Spin model, the advantages of the CSP model can be used when designing and implementing a pervasive adaptive system. Every device from the system can run processes communicating with processes on different devices. Processes on one device can be grouped in sets and cooperate. Every process can be responsible for different device capabilities, communication or synchronization with different devices.

5 DYNAMIC CONNECTIONS EXPERIMENT

This section presents an experiment with dynamic connections for synchronization using a JCSP based system. Synchronization on connection is an important issue when considering connections between devices. CSP processes can be dedicated to controlling the quality of communication between processes. Those additional processes will exclusively manage only one connection between devices.

The devices in this experiment are placed in different rooms. These devices have various capabilities and different types of communication links: wired or wireless. The experiment is to enable communication between devices from different rooms.

These tasks can be achieved using the CSP concepts and functions that JCSP offers. Priority and scheduling on channels can be accomplished by using guards and timers.

Aim of the Experiment

The aim of this experiment is to create a set of processes working on different machines performing: device and service discovery, synchronization on connection and dynamic creation of exclusive channels for data flow.

Method

Tasks of the system are to discover existing devices in rooms, create connections to perform service discovery and send commands between devices. Moreover the system is designed to create exclusive connections between devices for data flow. There are devices from the system that are created to enable communication between rooms.

Description of the system

The experiment will be carried out with two rooms and three types of devices. Rooms have different sets of devices. There are three types of devices: A, B and C shown on Figure 2.

Device type A is a device that needs to send some type of data. The device can connect to devices from Room 1 including device type B. Device type A has no knowledge about services offered in other rooms, but can recognize that device type B from its room can provide useful information about devices in other rooms.

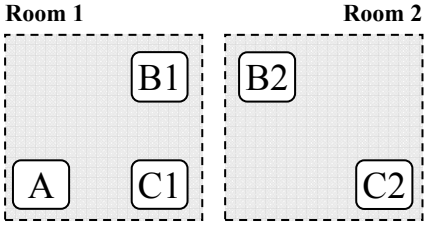

Figure 2. System design datagram.

Devices of type B are present in both rooms. A type B device is aware of devices and their capabilities in the same room as themselves and can also connect with other type B devices. Device type B performs an information service about devices outside its own room, and on request can find specific devices and arrange exclusive connections between devices. Device type

B is not a repository; devices from the system can connect to other devices from the room without asking device type B. Every device has its own device discovery capability, but connection outside the room is performed using device type B. Therefore device type B is another capability of the system, extending the connection scope, but communication inside of the room can occur without it.

Device type C is able to receive streams of data, and has specific channel inputs for processing particular types of data. In this experiment there are devices of type C in both rooms. Their capabilities are different, so depending on the data type to be sent, one of them will be used.

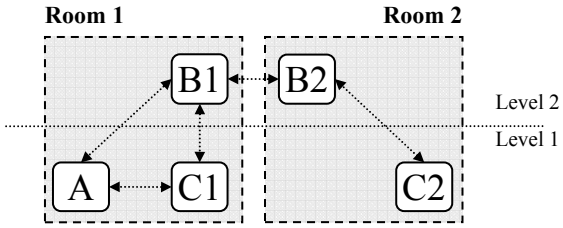

Figure 3. Connections established using a discovery service.

All the devices from the system can recognize the presence of different devices. A device discovery service is present in every device and provides information about all of the existing devices (Figure 3). Devices as shown in Figure 3 are divided into levels. Levels are used to manage initial connections between devices. Devices from level 1 can only connect other devices from the same room and have to request connection with devices from level above. This way device from lower level requests a connection with devices from one level above, and that manages the order of communication and reduces number of requests sent during the initialization stage. In Level 2 there are type B devices with high priority, that recognize devices from its own room and other devices from Level 2. In Level 1 there are devices type A and C that have only knowledge about devices in their own room and store this knowledge as a local repository.

The system consists of processes running concurrently on different devices communicating over TCP/IP links. Every device is equipped with main process responsible for key system decision making procedure and set of device discovery and information processes: Discovery Server and Discovery Client connected in an internal network (Figure 4).

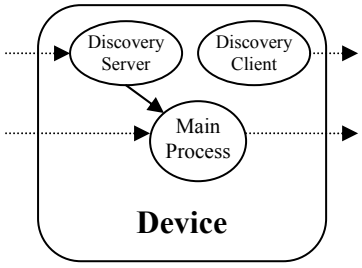

Figure 4. Devices internal network.

The network of processes in a device is initialized with localization and device capability data. The main process is responsible for establishing initial connections between the device and others, managing signals and data received by the device, send and receive commands. Moreover main processes in devices type B are responsible for dynamic creation of an instance of a process that can be connected to the network of processes already existing on the device, to manage exclusive connection between devices type A and C. The device discovery and information processes are responsible for informing other devices about their existence and receive information about other devices.

Network diagram

Let's consider a scenario that device A wants to send a particular type of data for processing to device C2. Device A first searches its own repository in order to find a suitable device. If an appropriate device is not present the device asks device B1 to provide a channel input, of a particular type, for sending data. Device B1 searches its repository and asks device B2 to fulfill a request. B2 checks the capabilities of devices in Room 2 and finds that device C2 is available to perform a request (Figure 5).

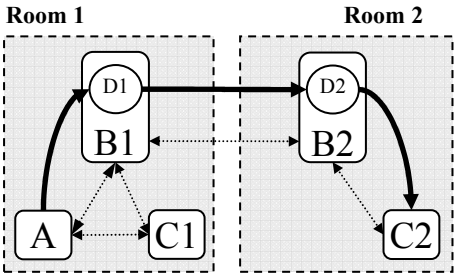

Figure 5. System design datagram.

Device B2 sends a request to device C2 for a channel input and asks it to be ready for input data. Next it creates a new process D2 (Figure 5) that will be exclusively responsible for a connection between B1 and C2. Device B1 also dynamically creates a process D1 so device A can connect to device C2. When an exclusive connection is ready B1 sends directions for device A to send data (Figure 5). When processes D1 and D2 finish passing data they are disconnected and destroyed.

In the system devices A and C2 can communicate although they are in different rooms. Devices B1 and B2 are responsible for initiating connections not maintaining them. When an exclusive connection is established processes D1 and D2 are responsible for it, devices B1 and B2 can perform different tasks and respond to requests from different devices.

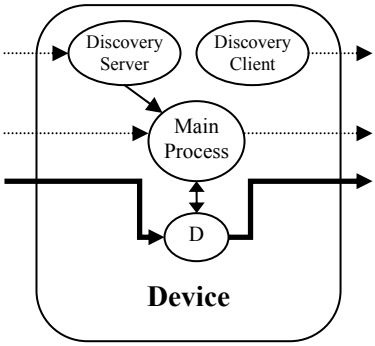

Figure 6. Exclusive connection management.

In the device internal network process D is connected to the main process (Figure 6). The exclusive connection initialized by the main process is now managed by dynamically created process D.

The device discovery capability is present in every device. As shown in Figure 6 there are two processes in every device responsible for finding new devices and recognizing that a device is no longer in use. Process DiscoveryClient sends an UDP (User Datagram Protocol) packet with information about itself to a broadcast address. DiscoveryServer process creates socket and waits for a communication. When communication occurs DiscoveryServer remembers the IP address of the sender and, if it's a new device, stores it in a local list. Processes run in parallel and every device can simultaneously send and receive packets.

Equipment and Libraries

Devices from the presented system are connected to an ad-hoc network and run Java scripts, communicating over TCP/IP network provided by a wireless router. Every device is a Personal Digital Assistant (PDA) Dell Axim X5 with Microsoft® Pocket PC operating system, processor Intel® PXA255/400MHz, RAM capacity of 64MB and IBM J9 Java Virtual Machine. Devices used in the system are in medium size, as it was defined for the purpose of this paper. The reason to use those machines is to show that the system can be run on devices with limited memory capabilities.

The Personal Digital Assistants used in the experiment have Standard Java 2 Platform Micro Edition (J2Me) 1.3 libraries.

Resulting system

The system consists of application classes with code and only limited additional JCSP libraries. The size will be a sum of those two sizes and is presented in Table 1.

Application classes:	48.5 KB
Additional JCSP libraries:	628.0 KB
Total:	**676.5 KB**

Table1. Size of the system

There are many packages in JCSP libraries, available at [2], the system only uses three of them: *jcsp.lang*, *jcsp.net2* and *jcsp.util*. The size of those libraries can be reduced, to separate classes that are used in the application. The JCSP Robot Edition (JCSPre) is a set of classes collected to support the system that controls LEGO NXT Robots [27]. The size of the *jcsp.lang* package was reduced from 223 KB to 20KB.

Devices in the system have different capabilities and perform various sets of tasks. Sizes of particular sets of classes for different devices are presented in Table 2.

Device type:	Application classes:	Additional JCSP libraries:	**Total:**
A	21.7 KB	628.0 KB	**649.7 KB**
B	24.4 KB	628.0 KB	**652.4 KB**
C	19.6 KB	628.0 KB	**647.6 KB**

Table 2. Size of the system on particular devices

The static size of the system on a particular device is no more than 700 KB (Table 2), assuming the presence of Standard JavaMe 1.3 libraries. The system is small in size, although it operates at a high level of abstraction. As the system is dynamic and adapts to a particular situation and changes in environment, the size of the system at runtime is larger than its static size.

Summary

We present a concept in which, the idea of a system is based on simple routing techniques. All devices from a proposed system have information about other devices' availability and can connect within a room with other devices. Only when a device with required capabilities is not present in the room, or a device cannot be directly connected, a device type B is used to help to create a dynamic connection between devices. This kind of connection will be virtual and devices on both sides will not be aware that it is not direct.

The system presented in this experiment dynamically creates a network of devices in rooms that discover other devices, connect appropriate ones, perform service discovery and creates dynamic connections to perform some tasks using available devices. The system has no central control and devices have different capabilities and can perform different tasks. Adaptation technique is designed to connect appropriate devices and enable creation of exclusive connections between devices for data flow. Device type B adapts to a request from devices type A by creating a mobile processes to manage requested connection type.

The system is small in size, but does operate at a high level of abstraction and organizes a dynamic adaptive pervasive environment. It performs device and service discovery, creates dynamic connections, supports sending commands between devices and manages an exclusive connection for data flow.

6 CONCLUSION AND FURTHER WORK

Pervasive adaptive computing creates new challenges for software development. This paper introduces JCSP dynamic connection capabilities and explores its usability for pervasive adaptation. We have described the infrastructure for software development in pervasive systems and useful JCSP capabilities have been presented.

Scalability in a CSP based system was demonstrated by a system modeling artificial blood platelets [26]. Mobility and distribution are supported by JCSP through mobility of channels, processes and code. Adaptation can be achieved by reconfiguring a JCSP system that allows adding, removing, reordering and reconnecting processes. Interoperability can be achieved using CPAUNP protocol between different CSP components not necessarily Java based. Readapting to changing environment at runtime can be accomplished by using JCSP dynamic connection capabilities and mobility of components.

The experiment of a simple pervasive adaptive environment was presented. The application dynamically creates a network of devices in rooms that discover other devices, connect appropriate ones, perform service discovery and create dynamic connections to perform data flow using available devices.

JCSP may not be the answer to all the challenges in pervasive adaptive computing, but some of it features might be useful. The system built using JCSP concepts is small in size and is appropriate for devices with small memory capacity. In CSP based systems simple processes can be composed into larger networks. As the system can be verified using the Spin model it

is appropriate also for large scale architectures consisting of small devices.

Interoperability is still an unsolved problem, but research in this area is in progress. JCSP advantages can be used in some parts of a pervasive system. The ability to create a system consisting of various types of components, implemented in different languages would be useful.

The size of the system presented in the experiment is measured in a static way, but the actual size of it at the runtime has not been calculated. The size of the system is crucial when considering using devices with vary small memory capabilities, so experiments to measure the runtime system size have to be performed.

ACKNOWLEDGMENT

This work is based on *jcsp.net2* package with the Communicating Process Architectures Universal Network Protocol developed by Chalmers [28] at Napier University.

REFERENCES

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

[2] P.H. Welch, P.D. Austin, The JCSP Home Page. http://www.cs.ukc.ac.uk/projects/ofa/jcsp/, 1999.

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

[4] M. Satyanarayanan, Pervasive Computing: Vision and Challenges, IEEE personal communications, 2001, pp. 10-17.

[5] C. Elliott, B. Heile, Self-Organizing, Self-Healing Wireless Networks, Aerospace Conference Proceedings, IEEE, 2000, pp. 355-362.

[6] A. Rashid, G. Kortuem, Adaptation as an aspect in pervasive computing, OOPSLA 2004 Workshop on Building Software for Pervasive Computing, Vancouver, British Columbia, Canada, 2004.

[7] R.H. Katz, Adaptation and mobility in wireless information systems, IEEE Communications Magazine 40, 2002, pp. 102-114.

[8] A.W. Roscoe, C.A.R. Hoare, R. Bird, The Theory and Practice of Concurrency, Prentice Hall PTR, Upper Saddle River, NJ, USA, 1997.

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

[10] J. Kerridge, Lecture Notes and Forthcoming Text Book, Private Communication, Napier University, Edinburgh, 2007.

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

[12] R. Milner, J. Parrow, D. Walker, A Calculus of Mobile Processes, Part I, Information and Computation 100, 1992, pp. 1-40.

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

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

[15] J. Kerridge, K. Chalmers, Ubiquitous Access to Site Specific Services by Mobile Devices: the Process View, Communicating Process Architectures, 2006.

[16] H. Ossher, P. Tarr, Using multidimensional separation of concerns to (re)shape evolving software, Communications of the ACM 44(2001) pp. 43-50.

[17] G.T. Heineman, W.T. Councill, Component-Based Software Engineering: Putting the Pieces Together, Addison-Wesley Professional, 2001.

[18] K. Chalmers, J. Kerridge, I. Romdhani, A Critique of JCSP Networking, in: I. Press, (Ed), Communicating Process Architectures, 2008.

[19] G.R.I. Golden, M. Spencer, Service and Device Discovery, McGraw-Hill Professional, 2002.

[20] S. Helal, N. Desai, V. Verma, L. Choonhwa, Konark - A Service Discovery and Delivery Protocol for Ad-Hoc Networks, Wireless Communications and Networking, 2003. WCNC 2003. 2003 IEEE, 2003.

[21] R. Singh, P. Bhargava, S. Kain, State of the art smart spaces: application models and software infrastructure, Ubiquity 7, 2006, pp. 2-9.

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

[23] J. Ferber, Multi-Agent System: An Introduction to Distributed Artificial Intelligence, Harlow: Addison Wesley Longman, 1999.

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

[25] G. Agha, C. Hewitt, Concurrent programming using actors: Exploiting large-scale parallelism Foundations of Software Technology and Theoretical Computer Science 206/1985, Springer Berlin / Heidelberg, 1985.

[26] P.H. Welch, F.R.M. Barnes, F.A.C. Polack, Communicating Complex Systems, 11th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS'06), 2006.

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

[28] K. Chalmers, Investigating Communicating Sequential Processes for Java to Support Ubiquitous Computing, Napier University, 2008.

[29] P.H. Welch, J.R. Aldous, J. Foster, CSP Networking for Java (JCSP.net), in: P.M.A. Sloot, C.J.K. Tan, J.J. Dongarra, A.G. Hoekstra, (Eds), International Conference Computational Science - ICCS 2330, Springer Berlin / Heilderberg, Amsterdam, The Netherlands, 2002, pp. 695-708.

[30] G.J. Holzmann, The SPIN Model Checker: Primer and Reference Manual, Addison-Wesley, 2003.