*Communicating Process Architectures 2006*
*Peter Welch, Jon Kerridge, and Fred Barnes (Eds.)*
*IOS Press, 2006*



# Ubiquitous Access to Site Specific Services by Mobile Devices: the Process View

Jon KERRIDGE and Kevin CHALMERS
*Napier University, Edinburgh, EH10 5DT, Scotland*
{j.kerridge, k.chalmers}@napier.ac.uk

**Abstract.** The increasing availability of tri-band mobile devices with mobile phone, wi-fi and Bluetooth capability means that the opportunities for increased access by mobile devices to services provided within a smaller locality becomes feasible. This increase in availability might, however, be tempered by users switching off their devices as they are overloaded with a multitude of messages from a variety of sources. A wide range of opportunities can be realised if we can provide a managed environment in which people can access wireless services specific to a particular physical site or location in a ubiquitous manner, independent of the service, and they can also choose from which services they are willing to receive messages. These opportunities range from retail promotions as a person walks down the street, to shopper specific offers as people enter stores that utilise reward card systems, to information about bus arrivals at a bus stop, additional curatorial information within a museum and access to health records within a hospital environment. The CPA paradigm offers a real opportunity to provide such capability with mobile processes, rather than the current approach that, typically, gives users access to web pages.

## Introduction and Motivation

The JCSP framework [1, 2] together with the jcsp.mobile package [3] provides the underlying capability that permits the construction of a ubiquitous access environment to services that are provided at the wi-fi and Bluetooth ranges of 100m and 10m respectively. Service providers will make available a universal access channel that permits the initial interaction between the person's mobile device and the service provider's server at a specific site or physical location. The mobile device will contain only a single simple access process that will download client processes from any service provider offering this type of service. These client processes will then interact with the service provider's server to achieve some desired outcome of benefit to both the user and the service provider. Regardless of the service provider, the same simple access process residing in the mobile device will remain unaltered. The access method will enable a mobile device to manage the interaction between several service providers at the same time. Once an interaction has finished, the resources used within the mobile device will be automatically recovered for reuse. Location and context determination is no problem because that is determined solely by the user's mobile device detecting and receiving a signal from a service provider.

In order that a person can personalize the services with which they are willing to interact, the mobile device should also contain a preferences document describing the types of interaction in which the user is willing to participate. The user can modify this document. Thus a user can ensure that only services in which they are interested will cause an interaction between the mobile device and a service provider's server. The mechanism is thus one into which a user opts-in, rather than having to opt-out of either specific services or services specified in some generic manner.

The benefits of this style of interaction are that service providers will be able to create services that are specific to their organisation. It is thus possible to make this into a personal service rather than current technology, which tends to broadcast relatively unspecific data to a wide number of people. The proposed method means users can access many different services with only one universal and ubiquitous access mechanism that could be installed by the mobile device manufacturer.

The mechanism is analogous to that employed in coastal VHF radio communication between ships and the coast guards. Mariners can call the coast guard using a previously defined listening channel. The coast guard then informs the mariner to switch to a specific channel upon which they can continue their conversation privately. In this proposal the means by which a communication between mobile device and any server is started is identical, universal and independent of the communications technology. The server then transfers a specific client to the mobile device that makes that interaction unique.

The UASSS (Ubiquitous Access to Site Specific Services) concept is applicable to many and varied applications of which a non-exhaustive list includes:

- Supermarkets could make special offers to customers as they walk into the store based upon the previous spending habits of the shopper and current stock availability within the store. Shoppers could be informed of both price reductions specific to them but of new lines in which they might be interested.
- A person walking down the street could be informed that the latest edition of a magazine they regularly buy is now available at the newsagent they are passing.
- A museum or art gallery visitor could be given information about the objects, at various levels of detail, in their chosen language as they move round the displays.
- Within the home it could be used to replace all remote controls by a single device that downloads each product's controller as required. More importantly devices could download new controllers from the Internet automatically and these would be made available without any user intervention. Intelligent devices such as digital camera systems that detect new photographs have been taken that can be automatically downloaded onto the home PC become feasible as the person walks into their home.
- Within hospitals the UASSS could be used to access electronic patient records and other information depending upon the location and role of the person accessing the hospital's systems.
- On entering a bus stop a person could be informed of the expected arrival of the next bus but only for the buses that stop at that bus stop and for which the person has indicated they normally catch this bus in their preferences document.

## 1. Background

Much of the current work in context and location aware computing is addressing a different set of problems namely, how consistent services can be provided, securely, to a user as they move around the environment and how precise location information can be obtained. Many researchers are looking at ways of being able to confirm the specific location of a person using a number of different technologies[4, 5], for example GPS [6] and radio tags [7], then knowing the precise location route planning guidance can be given [8] assuming the system knows where the person wants to go. The UASSS does not require this level of sophistication; if an access point can be detected that provides the ubiquitous access capability then the location of the person is known sufficiently to undertake an interaction.

Workers [9, 10] have recognised the importance of a software engineering approach to building such systems and the difficulty in achieving this goal. UASSS uses a well proven software engineering approach based upon Hoare's CSP [11], which provides a compositional methodology for building highly parallel systems of which this application provides a prime example. Others are working on access control, especially as a person moves around [12, 13] and the effect this has on power consumption. Some have addressed the problem of advertising [14] or those wishing to work in groups accessing a common repository [15], yet again not areas the UASSS needs to address.

Several groups have suggested a set of scenarios, which motivate the research they have undertaken [16, 17], none of which includes any of the suggested scenarios given earlier. It has been suggested in [18] "*It is a big design challenge to design personalised location awareness so that it does not require too much effort on the part of the users*."

### *1.1 Ubiquitous Computing*

*"UC [Ubiquitous Computing] is fundamentally characterized by the connection of things in the world with computation"* [19]. With this statement, Weiser describes the underlying fundamental principle of Ubiquitous Computing – attaching computational elements to physical objects. Research is generally focused on thinking about computers in the world, taking account of the environment and trying to remove them from our perception [20]. Or another viewpoint is the interaction between the physical and computing worlds [21]. Computers are everywhere, but this does not truly give us computing ubiquity, as a toaster with a microchip is still just a toaster. Only when the toaster can connect to other devices does ubiquity occur [19]. For example, this could enable the toaster to talk to the alarm clock, having toast ready for us when we get up in the morning, overcoming a small problem (if it can be called such) instead of addressing the large ones that computing usually aspires to.

Some other descriptions tend to involve the idea of smart spaces [22], that incorporate devices within them that form a smart dust of components. However, this diverges somewhat from the idea of a device performing a task for us. Want provides a good example of how to view this [23]. When purchasing a drill, do you want a drill, or do you want to make a hole? In general, the answer will most likely be the latter, and a good tool should be removed from our awareness [24], something the computer rarely does. This is where Weiser started to develop the notion of Ubiquitous Computing, examining new methods of people relating to computers, and trying to enable computers to take the lead in this relationship. As a further level of complexity as to how to think of computer ubiquity, the European Information Society Technology research program seems to also be addressing the same areas as Ubiquitous and Pervasive Computing [25], but without using these terms explicitly.

Another viewpoint taken is that of everyday computing [26], which considers the scaling of Ubiquitous Computing. What this appears to actually mean is simply adding the concept of time, as well as removing the beginning or end of a system interaction by the user and allows task interruption. It is arguable if this is actually different from UbiComp; time is an important piece of contextual information, and how to scale an idea that is meant to be incorporated into everything is another question.

These ideas of task and tools are generally highly coupled. As technology evolves, so do tasks and vice versa [27]. As this occurs, Weiser puts forth that we should not be adapting to technology, rather it should be technology fitting to our lives, something that Weiser terms as Calm Technology [19]. Calmness can also be considered similar to the invisibility of computers [28].

To put Ubiquitous Computing into a better context, Kindberg [21] provides a set of examples of interactions which are either not ubiquitous, or are borderline ubiquitous. Accessing email with a laptop, or a collection of wirelessly connected laptops are (very obviously) not examples of UbiComp; a smart coffee cup, peer-to-peer games and the Internet are borderline. Weiser agrees with the analogy of the Internet as a ubiquitous entity, although the belief is that the focus should be moved from thin clients to thin servers [19], both for the Internet and other devices. In fact, Weiser considers the Internet and embedded processors as signalling the beginning of the Ubiquitous Computing era (with 8 billion embedded processor sales compared to only 15 million PC in 2000 [23], this era can surely not be too far away).

Lately, research does seem to concentrate more on the idea of mobility. Weiser states that this is not all that UbiComp means, but the current literature indicates a shift towards this idea. An example of this is Want's argument [23] that Personal Digital Assistants (PDAs) and mobile phones are the most useful devices for Ubiquitous Computing developers, although limited in the required computing capability, integration and interface requirements. Another example is Abowd [26], who claims inch-scale computing is here in the form of the PDA. Weiser describes entities smaller than this however [20], and also states the Ubiquitous Computing is addressing different ground than the PDA [24], which is merely the acquirement of a tool without thinking of its purpose. As an example, Weiser describes the lifting of a heavy box. Either you call an assistant to help, or you automatically, and unconsciously, become stronger. Computing generally focuses on the former, whereas UbiComp aims for the latter.

The emphasis on mobile computing has enabled some other interesting trends. Some major issues with trying to develop UbiComp systems were the resource constraints of power and processing on the sometimes small and mobile devices required. Weiser's work actually forced new metrics to be established [19], such as MIPS (Millions Instructions Per Second)/Watt and Bits/Sec/$m^3$. Due to improvements, most likely attributable to mobile computing as a whole than just UbiComp, Weiser experienced improvements of a hundred fold in MIPS/Watt in three years after a decade of no improvement.

### *1.2 Site Specific Services*

What is apparent from the literature is that Ubiquitous Computing relies on services; devices or people executing some external task through an intermediary device. This is most apparent in the area of context-awareness and location-awareness in particular (Location Aware Services - LAS, Location Based Services - LBS), the goal being to provide access to resources via services [29]. The hope is to create an environment with entities interacting to provide and use services between one another, thereby utilising local resources.

The main problem here is the terminology. LAS and LBS seem to be interchangeable, and are extremely dependant on location data. A better solution is to provide services based in a location, or a site specific service [30]. This is basically a stationary LBS as described by Ibach [29], but here we remove ourselves from the ambiguous term of Location Based Services. A site specific service is much easier to relate to for users in the real world. Site Specific Services (SSS) exist in the form of ticket machines, information kiosks, interactive product catalogues, and ATMs. Adding UbiComp to the equation allows multiple users to exploit these services at once, without the normal waiting to access the machine.

There are issues when providing this type of interaction. First of all, different users using the same device should discover different services from one another [31], or more

precisely a users role should determine the services available to them. Secondly, services need to either support the plethora of client hardware platforms (a futile task), or be developed using a framework that is platform independent – such as Sun Microsystems' Java or, to a lesser extent, Microsoft's .NET Framework. The third problem is establishing connections between devices and localized services [30]. Users will need to be aware when connection is possible, and connection will need to occur as quickly and efficiently as possible. A final point to consider in this non-exhaustive list is the storage of personalized information on some form of mobile device, to allow interactions to be as tailored as much as possible. When we view the environment as a collection of distributed sites, with some sites within sites, we do not necessarily want our personalized information travelling electronically from site to site, a more physical form is probably preferable. This removes a wealth of the privacy concerns that are apparent in the general field of UbiComp, but the single device is still a point of weakness, and security will have to be built around it. Another possible solution is to distribute information between device and servers [32], reducing the amount of personal information carried by the user, but requiring a higher level of trust of the service provider.

The consensus seems to be that providing site specific services is a good form of Ubiquitous Computing, although little application of this is apparent. One such system [14] involved messages sent to phones as users passed stores, and although technologically restricted (Bluetooth enabled mobile phones only) these ideas appear to have been implemented on a commercial scale [33] by Filter UK, who provide multimedia content at locations using Bluetooth and Java. Another system [30] provides a service explorer interface, allowing users to interact with site specific services via a mobile phone. This solution does appear to be a glorified web browser that provides a UI to a nearby terminal on a phone, which in itself is not a bad idea, but does not really allow the level of interactions required for Ubiquitous Computing. Ubiquitous Computing is considered one of the Grand Challenges in Computing Research [35], an area that Milner appears interested in applying techniques from the π-calculus to [36].

## 2. The Underpinning Architectural Process Structures

The two key underlying capability requirements of the UASSS are:

- the mobile device contains a process, called the access process, which can identify wireless access points using a variety of technologies, and
- the service provider makes available an access channel that is universally known by the same name.

The first requirement implies that any mobile device not only contains the access process but also the infrastructure to run processes in a JCSP framework. This capability has been achieved and is reported elsewhere [3]. These capabilities could be added to a mobile device after initial purchase or could be incorporated into such devices when they are manufactured.

The second requirement means that the wider community, involving service providers, network providers and mobile device manufacturers have to agree on a single name by which access to this type of site specific service is initiated. For the purposes of this paper it shall be called "A". Once these requirements have been realised then the following simple set of interactions allows a mobile device to connect to any site specific service.

### *2.1 Detection of the Site Specific Service by the Mobile Device*

The access process (AP) in the mobile device detects the presence of either a wi-fi or Bluetooth wireless access point (WAP) that is transmitting data in the form of an IP address of a node upon which the address of a JCSPNet Channel Name Service (CNS) [2] is located. Most WAPs for security reasons do not transmit such data and thus by default do not participate in UASSS capability. Once AP has received the address of a CNS, it then initializes itself as a node in the service provider's network thereby becoming an ad-hoc member of the network. AP then creates an anonymous network channel input end, the location of which is sent to the server using the access channel "A", which is the only communication between the mobile device and the site specific server (SSS) using "A".

### *2.2 Communication of an Initial Mobile Process from Server to Mobile Device*

The SSS receives the location of the mobile device's network channel input end and uses that to create a network output channel. Over this channel it now communicates the initial mobile process (IMP) that is to be executed within the mobile device. On reading the IMP, the mobile device, causes it to be executed. This IMP could be the only process that is transferred but in other cases this IMP might, by means of a user interaction, determine the user's specific requirement and cause the transfer of yet further processes. These transfers will take place over channels that are private and hence known only to the SSS and the IMP. The IMP achieves the ubiquitous nature of the interaction by ensuring a user only becomes involved in any interaction with a service provider identified in a preferences document. The IMP will thus interrogate the preferences document and will discard any communication from a service provider that does not appear therein.

The mobile device, by means of the IMP, is now able to communicate with the SSS. If further channels are required for this interaction then these can be passed as properties of the mobile process and dynamic anonymous channel connections can be created in the same manner as the channel used to send the initial process. The way this capability can be utilized is now explained by a simple application.

## 3. The Meeting System

Consider a train station or airport in which people congregate waiting for the departure of trains and flights that might be delayed. If you are traveling within a group it would be sensible to meet together during the delay. The management of the transport infrastructure has set up a means whereby such ad hoc meetings can be registered and subsequently accessed by other members of the group, provided they have previously agreed on a name by which they will recognize the group. One person registers the meeting name and a location with the service. Subsequent members of the group may try to create the same meeting so they are told that it has already been created and where it is located. Others, typically those that arrive close to the original departure time will just try to find the meeting, possibly knowing that they are unlikely to be the first person to arrive. Anyone trying to find a meeting that has not yet been created will be told that the meeting has not yet been registered.

### *3.1 The Access Process (AP)*

Listing 1 gives the Java code for the Access Process, which is generic and applicable to all sites offering the UASSS capability. This representation assumes access to only a single

SSS and that we know the IP address of the machine running the CNS. The AP, as well as the remainder of the system utilizes the jcsp.mobile package [3]. The AP is specified in Java as this process has to run on a mobile device that will only have access to a (limited) subset of the full Java environment made available on mobile devices.

```
01  public class AccessProcess {
02
03    private static ProcessChannelInput processReceive;
04
05    public static void main(String[] args) {
06      String CNS_IP = Ask.string("Enter IP address of CNS: ");
07      Mobile.init(Node.getInstance().init(new TCPIPNodeFactory(CNS_IP)));
08      String processService = "A";
09      NetChannelLocation serverLoc = CNS.resolve(processService);
10      NetChannelOutput toServer = NetChannelEnd.createOne2Net(serverLoc);
11      processReceive = Mobile.createNet2One();
12      toServer.write(processReceive.getChannelLocation());
13      MobileProcess theProcess = (MobileProcess)processReceive.read();
14      new ProcessManager(theProcess).run();
15    }
16  }
```

Listing 1. The Access Process

The channel `processReceive` {line 3} is the network channel upon which the IMP will be received. The IP address of the CNS is determined by means of a console based user interaction {line 6} but in practice would be determined automatically by the device. The mobile device is then initialized as a member of the network that is the same as that of the CNS {7}. The name of the access channel "A" is defined in `processService` {8}. The location of the input end of the access channel is then `resolve`d because this process writes to a process in the SSS {9} and then the output end is created {10}. The `processRecieve` channel is then created {11} and its channel location written to the process that inputs messages on the "A" channel {12}. The AP now reads the IMP as `theProcess` {13} and this is then run using a ProcessManager {14}, a JCSP class that permits a process to be spawned concurrently with the AP process. At this point the IMP will be executed and thus the processing becomes specific to the site from which it has been downloaded.

### *3.2 The Architecture of the Site Specific Service Server*

Figure 1 shows the outline of the architecture required to support a SSS, which has been specialised to the Meeting Organiser application. Some of the processes may be executed on different processes within the network but that is of no concern for this explanation. The input end of the access channel "A" is connected to the IMPServer, which is responsible for obtaining and then communicating an instance of an IMP process to a mobile device. The IMP instance is obtained from an IMPSender process. The named network channels "N" and "F" are used by the Meeting Organiser to manage the communication of instances of the NewMeeting and FindMeeting client processes. Each client service known to the SSS has its own pair of Server and Sender processes. The Sender manages a list of available client processes, such that when a service client has completed it can be reused.

In this case the Meeting Database is relatively simple and just receives requests on its requestChannels. The required response channel is allocated dynamically because the input end of the channel is created within a mobile device. Once an interaction has completed, the MeetingDatabase process can inform the appropriate Sender process which service client is available for reuse, using one of the Reuse channels.

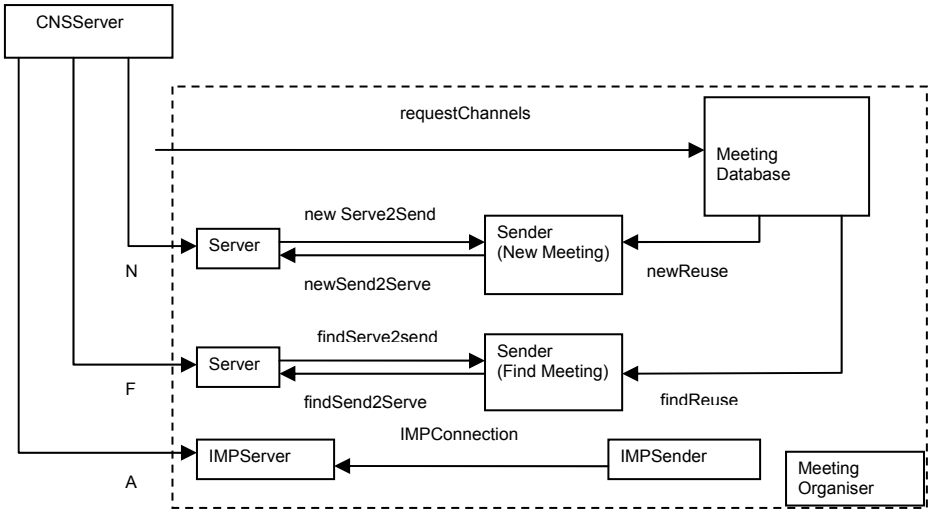


Figure 1. Architecture of the Meeting Organiser Server

### *3.3 The Initial Mobile Process (IMP)*

Each mobile process within the UASSS, typically comprises two processes, one that provides the functional capability, with the second providing the user interface. The IMP (Listing 2) is no exception {17-26}. The `events` channel {19} provides the interface between the user interface and the capability process. In this case no configuration channel is required as the interface is used merely for input events from the interface to the capability process. The code has been created using the Groovy Parallel formulation [34]. The use of Groovy has no impact on the processing as the `jcsp.mobile` package automatically downloads any classes from the server that are not present on the mobile device, including any specifically needed by the Groovy system.

```
17  class InitialMobileProcess extends MobileProcess {
18    void run () {
19      def events = Channel.createAny2One()
20      def processList = [
21                new InitialClientCapability ( eventChannel : events.in() ) ,
22                new InitialClientUserInterface ( buttonEvent : events.out() )
23                        ]
24      new PAR (processList).run()
25    }
26  }
```

Listing 2. The InitialMobileProcess Structure

#### *3.3.1 Initial Client Capability Process*

The `InitialClientCapability` process (Listing 3) waits to read an `eventType` {31} from the user interface process and then depending upon whether the user has pressed the "Create New Meeting" or "Find Existing Meeting" button sets the `serviceName` to "N" or "F" as required {34, 37}. These service names are private to the SSS and are connected to respective Servers as shown in Figure 1. The `serviceName` is resolved with the CNS {39} so that we can create a network output channel from this process to the required server {40}. An anonymous mobile process input channel `processReceive` is then created {41} and its location written to the server {42}. This message indicates that a mobile device requires an instance of the specific process and also informs the server which network

address should be used to communicate the mobile client process. The process is then read {43} and executed {44}. When the loaded process terminates all the resources used by the mobile device are automatically recovered.

```
27  class InitialClientCapability implements CSProcess {
28    @Property ChannelInput eventChannel
29
30    void run () {
31      def eventType = eventChannel.read()
32      def serviceName = null
33      if ( eventType == "Create New Meeting" ) {
34        serviceName = "N"
35      }
36      else {
37        serviceName = "F"
38      }
39      def serverLoc = CNS.resolve(serviceName)
40      def toServer = NetChannelEnd.createOne2Net(serverLoc)
41      def processReceive = Mobile.createNet2One()
42      toServer.write(processReceive.getChannelLocation())
43      def theProcess = (MobileProcess)processReceive.read()
44      new ProcessManager(theProcess).run();
45    }
46  }
```

Listing 3. The Initial Client Capability Process

### *3.3.2 The Initial Client User Interface*

The `InitialClientUserInterface` is a simple interface with two buttons called "Create New Meeting" and "Find Existing Meeting". The interface uses the active AWT components that are part of JCSP.

## *3.4 The New Meeting Client*

For the purposes of explanation we shall describe the New Meeting Client only.

### *3.4.1 The Meeting Data Object*

`MeetingData` (Listing 4) has been created to send data from the mobile device and also to return results of the request from the `MeetingOrganiser` to the user client process. It has to implement the `Serializable` interface because it is communicated over the network. In general, care must be taken accessing objects communicated between processes to ensure no aliasing problems. In this case, this problem is alleviated because *all* communication is over a network, so the underlying system makes a deep copy of any object.

The property `returnChannel` {48} holds the net location of an input channel that is used to return results to the user process in the mobile device. The `clientId` {49} is the identity of the client process used by this user process. The properties `meetingName` {50} and `MeetingPlace` {51} are text strings input by the user of the mobile device to name the meeting and the place they are to meet. The property `attendees` {52} indicates the number of people that have already joined the group of people that form the meeting.

```
47  class MeetingData implements Serializable {
48    @Property returnChannel
49    @Property clientId
50    @Property meetingName
51    @Property meetingPlace
52    @Property attendees
53  }
```

Listing 4. The MeetingData Object

### 3.4.2 New Meeting Client Process

The process follows the same pattern used before in that it comprises capability and user interface processes as shown in Listing 5 and extends `MobileProcess` [3].

The property `clientServerLocation` {55} is the location of a network input channel that forms one of the requestChannels shown in Figure 1. The property `clientId` {56} indicates which of the available `NewMeetingClientProcesses` has been allocated to this user's mobile device. It will subsequently be used to indicate that the process can be reused once the interaction has completed.

Connecting `NewMeetingClientCapability` to `NewMeetingClientUserInterface`, a set of event and configuration channels is defined {59}. The `clientProcessList` is then defined {60-69} comprising the two processes. The processes are passed property values that connect the capability process to the user interface and vice versa. The process is then executed by means of a **`PAR`** {70}.

```
54  class NewMeetingClientProcess extends MobileProcess {
55    @Property NetChannelLocation clientServerLocation
56    @Property int clientId
57
58    void run () {
59      // define event and configuration channels between processes
60      def clientProcessList = [
61        new NewMeetingClientCapability (
62          clientId : clientId,
63          clientServerLocation : clientServerLocation,
64          // and interface channel connections
65          )
66        new NewMeetingClientUserInterface (
67          // interface channel connections
68          )
69      ]
70      new PAR ( clientProcessList ).run()
71    }
72  }
```

Listing 5. The New Meeting Client Process

### 3.4.3 The New Meeting Client Capability

Listing 6 shows the relatively simple process associated with creating a new meeting. The property `clientServerLocation` {74} is the connection to the requestChannel shown in Figure 1 and `clientId` {75} is the number of the client process being used.

The event and configuration channels used to create the connection between the capability and user interface are defined. A network output channel is then created that connects the capability process to the meeting organizer service is then created using the `clientServerLocation` property {79}. The network channel used to return the results from the server to this client is then defined {80}. Lines {81-86} defined an instance of `MeetingData`, which is then populated with the necessary data values. The property `returnChannel` {82} contains the location of the input network channel and the `meetingName` {84} and `meetingPlace` {85} properties are obtained directly from the user interface. The `clientData` is then written to the meeting organizer server {86}. The mobile device then reads the `replyData` {87}.

The value of `attendees` indicates whether this meeting has been created or whether the meeting had already been created {88-95}. In either case the user is told where the meeting is taking place. The number of `attendees` already at the place is also written to the user interface.

```
73  class  NewMeetingClientCapability implements CSProcess {
74    @Property NetChannelLocation clientServerLocation
75    @Property int clientId
76    // event and configuration channels that connect UI to NMCC
77
78    void run () {
79      def client2Server = Mobile.createOne2Net(clientServerLocation)
80      def server2Client = Mobile.createNet2One()
81      def clientData = new MeetingData()
82      clientData.returnChannel = server2Client.getChannelLocation()
83      clientData.clientId = clientId
84      clientData.meetingName = meetingNameEvent.read()
85      clientData.meetingPlace = meetingLocationEvent.read()
86      client2Server.write(clientData)
87      def replyData = (MeetingData) server2Client.read()
88      if ( replyData.attendees == 1 ) {
89        registeredConfigure.write("Registered")
90      }
91      else {
92        registeredConfigure.write("ALREADY Registered")
93      }
94      registeredLocationConfigure.write(replyData.meetingPlace)
95      attendeesConfigure.write(new String (" " + replyData.attendees) )
96    }
97  }
```

Listing 6. New Meeting Client Capability

### *3.4.4 New Meeting Client User Interface*

The user interface process is simply a collection of interface components connected to the capability process by a set of event and configuration channels. The interface is created as an ActiveClosingFrame. The input container comprises two labels and two `ActiveTextEnterFields` that are used to enter the name of the meeting and the place where the people are to congregate. The response container gives the user feedback as to whether the meeting was created or if somebody had already registered the meeting and its location. The active components that make up the interface are added to a process list and then invoked using a **`PAR`**.

## *3.5 The Processes Contained Within the Meeting Organiser Site Specific Server*

Figure 1 shows the processes contained within the SSS, which shall now be described. The server and sender processes are generic and thus independent of the particular client that is to be communicated.

### *3.5.1 The Server Process*

The Server process (Listing 7) has properties that include its channel {100,101} connections to its related sender process and the name of the service it provides. The `serviceName` {202} is the name of the private network channel that an IMP will use to obtain a mobile client for a specific service.

The server is defined {105} as an instance of `MultiMobileProcessServer`, defined in jcsp.mobile [3]. The server is then initialised with the channel connections to its sender process and the `serviceName` {106} and invoked {107}. Such a server waits for a request on a network channel with the same name as `serviceName`. The request takes the form of a network channel to which it can write a mobile client process. The server process then signals its need for an instance of the mobile client process on its `toSender` channel. It

then reads the mobile client process from its `fromSender` channel and communicates it to the mobile device using the network channel identified in the original request.

```
98  public class Server implements CSProcess {
99
100   @Property ChannelInput fromSender
101   @Property ChannelOutput toSender
102   @Property String serviceName
103
104   void run() {
105     def theServer = new MultiMobileProcessServer()
106     theServer.init(serviceName, fromSender, toSender)
107     new PAR ([theServer]).run()
108   }
109 }
```

Listing 7. The Generic Server process

### *3.5.2 The Sender Process*

The nature of the `Sender` process (Listing 8) will vary depending upon the requirements of the service being provided. The properties identify the channels that connect the `Sender` to its related `Server` process {112,113} and the channel upon which it receives inputs informing it that a client can be reused {114}. It receives a `List` of `clients` {115} that have been allocated to this server. Typically these client processes will all be instances of the same mobile client process, which have the required network input channel locations embedded in each instance. Network channels that are used to output results from the service to the mobile device have to be created dynamically see {80, 82, 86}.

```
110 public class Sender implements CSProcess {
111
112   @Property ChannelOutput toServer
113   @Property ChannelInput fromServer
114   @Property ChannelInput reuse
115   @Property List clients
116   void run() {
117     def serviceUnavailable = new NoServiceClientProcess()
118     def n = clients.size()
119     def clientsAvailable = []
120     for (i in 0 ..< n) {
121       clientsAvailable.add(clients[i])
122     }
123     def alt = new ALT ([reuse, fromServer])
124     def index, use, client
125     while (true) {
126       index = alt.select()
127       if (index == 0 ) {
128         use = reuse.read()
129         clientsAvailable.add(clients[use])
130       }
131       else {
132         fromServer.read()
133         if (clientsAvailable.size() > 0 ) {
134           client = clientsAvailable.pop()
135           toServer.write(client)
136         }
137         else {
138           toServer.write(serviceUnavailable)
139         }
140       }
141     }
142   }
143 }
```

Listing 8. The Generic Sender Process

In this case the `Sender` implements a simple quality of service capability by sending a `NoServiceClientProcess` {117, 138} if none of the actual client process instances can be used. Each of the client processes is added to a list of available clients {119-122}.

The Sender alternates over the `reuse` and `fromServer` channels {123} and the never ending loop {125} either adds a client process to the `clientsAvailable` list, in the case of an input on the `reuse` channel {127-130}. Otherwise, a request from a mobile device is read from an IMP and either a client or the service unavailable process is sent {132-136}.

### *3.5.3 The Meeting Process*

```
144 class Meeting implements CSProcess {
145   @Property List requestChannels
146   @Property ChannelOutput nReuse
147   @Property int newClients
148   @Property ChannelOutput fReuse
149   @Property int findClients
150
151   void run() {
152     def meetingMap = [ : ]
153     def alt = new ALT (requestChannels)
154     def newMeeting = new MeetingData()
155     def findMeeting = new MeetingData()
156     def replyData = new MeetingData()
157     def index
158     while (true) {
159       index = alt.select()
160       switch (index) {
161         case 0 ..< newClients :
162           newMeeting = requestChannels[index].read()
163           def reply = Mobile.createOne2Net(newMeeting.returnChannel)
164           if ( meetingMap.containsKey(newMeeting.meetingName ) ) {
165             replyData = meetingMap.get(newMeeting.meetingName)
166             replyData.attendees = replyData.attendees + 1
167           }
168           else {
169             replyData = newMeeting
170             replyData.attendees = 1
171           }
172           meetingMap.put ( replyData.meetingName, replyData)
173           reply.write(replyData)
174           nReuse.write(replyData.clientId)
175           break
176         // case to deal with find meeting requests
177       } // end of switch
178     } // end of while
179   } // end of run()
180 }
```

Listing 9. The Meeting Database Process

The `Meeting` process (Listing 9) implements the meeting database shown in Figure 1. Its properties {145-149} are directly related to that diagram and comprise the requestChannels, and the two channels by which client processes that can be reused are identified. The number of new and find meeting client processes is also required. A map, `meetingMap` {152} is used to hold the name of the meeting, as its key, and the meeting location, the map entry. It is presumed that the `requestChannels` list has been ordered such that all new meeting client requests are in the first part of the list and these are followed by the find meeting requests. Hence the `Meeting` process has to alternate over the two parts of the list as shown {161, 176}.

In the case of new meeting requests {162-175} the request is read into `newMeeting` {162}. A network `reply` channel is created from the `returnChannel` attribute of the

`MeetingData` object {163}. If the `meetingMap` already contains an entry for this `meetingName` {164} then the number of attendees is incremented in the replyData object {165-166}, otherwise it is set to 1 {170}. The entry in the `meetingMap` is then refreshed {172}. The reply is written back {173} to the mobile device that is currently executing the mobile process `NewMeetingClientProcess` that is able to interpret the reply appropriately {88}. The `clientId` of the client undertaking this interaction can then be written on the `nReuse` channel {174} so that this client can be reused. In this simple case the interaction between mobile device and server is such that only one request and reply are required and thus the mobile client process can be reused immediately. The implementation of the find meeting processing is similar.

### *3.5.4 The Meeting Organiser*

The Meeting Organiser (Listing 10) is a Groovy script that instantiates the server components given in Figure 1.

```
181   def CNS_IP = Ask.string("Enter IP address of CNS: ")
182   Mobile.init(Node.getInstance().init(new TCPIPNodeFactory(CNS_IP)))
183
184   def nSize = Ask.Int("Number of Concurrent New Meeting Clients? ", 1, 2)
185   def fSize = Ask.Int("Number of Concurrent Find Meeting Clients? ", 1, 3)
186   def newRequestLocations = []
187   def netChannels = []
188   for (i in 0 ..< nSize) {
189     def c = Mobile.createNet2One()
190     netChannels << c
191     newRequestLocations << c.getChannelLocation()
192   }
193   def NMCList = []
194   for (i in 0 ..< nSize) {
195     NMCList << new NewMeetingClientProcess ( clientServerLocation :
196                                  newRequestLocations[i], clientId : i )
197   }
198   // now do the same for the Find Meeting Clients
199   def newServe2Send = Channel.createOne2One()
200   def newSend2Serve = Channel.createOne2One()
201   def newReuse = Channel.createOne2One()
202
203   //and similarly for the connections between findSender and findServer
204   def IMPConnection = Channel.createOne2One()
205   def processList = [
206     new IMPSender(toAccessServer:IMPConnection.out()),
207     new IMPServer(fromAccessSender:IMPConnection.in()),
208     new Server( fromSender:newSend2Serve.in(),
209                 toSender:newServe2Send.out(), serviceName: "N"),
210     new Sender( toServer:newSend2Serve.out(),
211                 fromServer:newServe2Send.in(),
212                 reuse:newReuse.in(), clients: NMCList),
213     new Server( fromSender:findSend2Serve.in(),
214                 toSender:findServe2Send.out(), serviceName: "F"),
215     new Sender( toServer:findSend2Serve.out(),
216                 fromServer:findServe2Send.in(),
217                 reuse:findReuse.in(), clients: FMCList),
218     new Meeting( requestChannels : netChannels,
219                  nReuse : newReuse.out(), newClients : nSize,
220                  fReuse : findReuse.out(), findClients : fSize ) ]
221
222   new PAR(processList).run()
```

Listing 10. Meeting Organiser

The IP address of the CNS is obtained {181} and used to initiate this component as a node in the network {182}. The number of new and find meeting clients are then obtained {184,185}. An empty list `newRequestLocations` is defined {186} that will be used to hold location information for request channels allocated to new meeting client processes. Similarly, the list `netChannels` {187} is used to hold all the request channels for both new and find meeting clients. The loop {188-192} creates a net input channel and then appends (<<) it to the list `netChannels` and then its net location is appended to `newRequestLocations`. The list `NMCList` {193} holds the collection of `NewMeetingClientProcesse`s which are appended {195} with the required element of `newRequestLocations` and the identification number of the client. The same sequence of operations is the undertaken for the find meeting client processes (not shown).

The internal channels required to connect the processes are now defined {199-201} and also for the connection between the IMP sender and server processes {204}. The sender and server processes associated with IMP are similar to the generic `Server` and `Sender` processes (3.5.1, 3.5.2) but the IMP process is always available and hence there is no need to request an instance.

The list of processes needed to instantiate the meeting organiser system is then created {205-221} and invoked {222}. Of special note is the fact that the `Meeting` process is passed `netChannels` for the property `requestChannels`, over which it can alternate. However, each client process is passed the location of the equivalent `netChannel` because the mobile process has to define the channel locally and needs a net channel location to achieve this {79, 195-196}.

## 4. Evaluation

The UASSS is a generic capability that is fundamentally quite simple and can be applied in a universal manner. The requirements are that a mobile device contains a cut down version of the JCSP [2], sufficient for the purposes of running JCSP processes and the active AWT components. The only process that is executed within the mobile device is a relatively simple access process (AP). Furthermore the normal Java virtual machine present in a mobile device is very limited and we wish to reduce the amount of additional software footprint on the device [37]. The use of Java as the basic processing mechanism for most mobile devices, including mobile phones makes this choice relatively simple. Hence other language platforms have not been considered for this application. Perhaps of greater interest is that the required classes can be obtained easily by the mobile device from the server on an as needed basis by the dynamic class loading capability and this includes any additional classes required by the use of Groovy. Once the interaction is complete these additional classes can be automatically removed. Should the mechanism become ubiquitous then it would make interactions more efficient if most of the user interface and other commonly used classes were added to the basic software maintained within the mobile device.

Any organization can set up a SSS to which they provide WAP access. The only requirement being that the IP address of the node upon which a CNS is running has to be made publicly available to be discoverable by mobile devices. This does have security implications but this can be overcome by the use of a firewall and the connection through the firewall is programmed by the SSS provider. The service uses a number of generic components and design patterns.

Each specific service that has its own client requires a Server and Sender process pair. The Initial Mobile Process requires a specialization of the Sender-Server pair so that a copy

of the IMP is made readily available to any mobile device. Each client process itself comprises two processes; a capability and a user interface. Some form of repository needs to be provided which is accessed by the mobile client processes.

The advantage of the approach is that both the client and service processes are written by the same organization and so they have complete control over the interaction that is permitted. This means that the normal security resulting from the use of Java classes becomes immediately available because if a mobile client or the server processes receive a class for which they have no definition then one of two things can happen. In the case of the server, it should know all its class definitions and thus this can be immediately discarded because the mobile client is trying to access the server using a class that is not known to the server. In the case of the mobile device if it requires access to a class definition it can request it from the server. If the server has no knowledge of the class then this could be considered an unauthorized class access and processing should be discontinued.

The JCSP network [38] version contains the concept of filtered channels, which are used internally to enable the loading of classes over the network by the dynamic class loading capability. This could be easily used to provide a means of encrypting data sent over the channels used by the mobile processes and the server.

The full capability of UASSS will only be achieved if a suitable business model can be developed that allows mobile phone network operators some form of income when a mobile device that normally accesses their network is used for non-network activity, as it is in this case. Currently, wi-fi and Bluetooth are used to provide continuity of connection as the mobile phone user moves around. In particular, many of the IP communication ports are disabled and this would need to be changed were the UASSS to be fully exploited.

## 5. Current and Future Work

Current work is focused on the ability to manage interaction with more than one service provider and hence mitigate the effect of overlapping WAPs. We shall be developing a means of permitting users to specify the type of interactions in which they are willing to participate. In this manner the interactions become ubiquitous because if the user specifies that they do not wish to participate in a specific type of interaction then they will not be informed even of the existence of such an interaction. Users will only be aware and prompted for interaction when the IMP has detected a situation where the user is willing to participate. The IMP will need to be extended to access this document of preferences maintained on the user's mobile device to determine whether or not to interact. Such a preferences document will need to be easily maintainable by a user.

Currently a student is preparing a demonstration of the operation of this technology in supermarkets where customers receive offers personalized to the owner of the mobile device as they enter the store. A further student is developing an application applicable to use in hospitals. The goal in this case is to make data available from an electronic patient record and other sources to clinicians, nurses and other staff on the basis of their location and their role within the hospital. In this case the mobile device will provide an interface to a SQL database.

## References


[1] P. H. Welch and J. M. R. Martin, "A CSP Model for Java Multithreading," in P. Nixon and I. Ritchie (Eds.), *Software Engineering for Parallel and Distributed Systems,* pp. 114-122. IEEE Computer Society Press, June 2000.

[2] P. H. Welch, J. R. Aldous, and J. Foster, "CSP Networking for Java (*JCSP.net*)," in P. M. A. Sloot, C. J. Kenneth Tan, J. J. Dongarra, and A. G. Hoekstra (Eds.), *Proceedings of International Conference Computational Science – ICCS 2002, Lecture Notes in Computer Science 2330,* pp. 695-708. Springer Berlin / Heidelberg.

[3] K. Chalmers and J. Kerridge, "jcsp.mobile: A Package Enabling Mobile Processes and Channels," in J. Broenink, H. Roebbers, J. Sunter, P. H. Welch, and D. Wood (Eds.), *Communicating Process Architectures 2005 (WoTUG- 28),* IOS Press, Amsterdam, The Netherlands, 2005.

[4] J. Nord, K. Synnes, and P. Pames, "An Architecture for Location Aware Applications," in *Proceedings of the 35th Annual Hawaii International Conference on System Sciences 2002, HICSS,* pp. 3805 – 3810, IEEE Computer Society Press.

[5] Y.-C. Chen, Y.-J. Chan and C.-W. She, "Enabling Location-Based Services in Wireless LAN Hotspots," *ACM International Journal of Network Management*, 15(3), pp. 163-175, 2005.

[6] N. Marmasse and C. Schmandt, "Location-aware Information Delivery with comMotion," in P. Thomas and H. W. Gellersen (Eds.), *Proceedings Handheld and Ubiquitous Computing: Second International Symposium, HUC 2000, Lecture Notes in Computer Science 1927,* pp. 157. Springer Berlin / Heidelberg.

[7] A. Harter, A. Hopper, P. Steggles, A. Ward, and P. Webster, "The Anatomy of a Context-Aware Application," *Wireless Networks*, 8(2-3), pp. 187-197, 2002.

[8] T. Bohnenberger, A. Jameson, A. Krüger, and A. Butz, "Location-Aware Shopping Assistance: Evaluation of a Design Theoretic Approach," in F. Paternò (Ed.), *Proceedings Mobile Human-Computer Interaction: 4th International Symposium, Mobile HCI 2002, Lecture Notes in Computer Science 2411,* pp. 155-169. Springer Berlin / Heidelberg, 2002.

[9] W. Griswold, R. Boyer, S. Brown, and T. Truong, "A Component Architecture for an Extensible, Highly Integrated Context-Aware Computing Infrastructure," in *Proceedings 25th International Conference on Software Engineering (ICSE '03)*, pp. 363. IEEE Computer Society Press, 2003.

[10] P. Fahy and S. Clarke, "CASS – Middleware for Mobile Context-Aware Applications," presented at MobiSys 2004 Workshop on Context Awareness, 2004. Available at http://sigmobile.org/mobisys/2004/ context_awareness/papers/cass12f.pdf.

[11] A. Abdallah, C. Jones, and J. Sanders, *Communicating Sequential Processes: The First 25 Years*: Springer, 2005.

[12] A. Friday, M. Wu, J. Finney, S. Schmid, K. Cheverst, and N. Davies, "Network Layer Access Control for Context-Aware IPv6 Applications," *Wireless Networks*, 9(4), pp. 299-309, 2003.

[13] C. Efstratiou, K. Cheverst, N. Davies, and A. Friday, "An Architecture for the Effective Support of Adaptive Context-Aware Applications," in K.-L. Tan, M. J. Franklin, J. C.-S. Lui (Eds.), *Proceedings Mobile Data Management: Second International Conference, MDM 2001, Lecture Notes in Computer Science 1987,* p. 15. Springer Berlin / Heidelberg, 2001.

[14] L. Aalto, N. Göthlin, J. Korhonen, and T. Ojala, "Bluetooth and WAP Push Based Location-Aware Mobile Advertising System," in *Proceedings of the 2nd International Conference on Mobile Systems, Applications, and Services,* pp. 49-58. ACM Press, NY, USA, 2004.

[15] O. Conlan, R. Power, S. Higel, D. O'Sullivan, and K. Barrett, "Next Generation Context-Aware Adaptive Services," in *Proceedings of the 1st International Symposium on Information and Communication Technologies, ACM International Conference Proceedings Series,* 49, pp. 205-212. Trinity College Dublin, 2003.

[16] A. Friday, N. Davies, N. Wallbank, E. Catterall, and S. Pink, "Supporting Service Discovery, Querying and Interaction in Ubiquitous Computing Applications," *ACM Baltzer Wireless Networks (WINET)*, 10(6), pp. 631-641, 2004.

[17] S. Duri, A. Cole, J. Munosn, and J. Christensen, "An Approach to Providing a Seamless End-user Experience for Location-aware Applications," in *Proceedings of the 1st International Workshop on Mobile Commerce*, pp. 20-25. ACM Press, NY, USA, 2001.

[18] E. Kaasinen, "User Needs for Location-Aware Mobile Services," *Personal and Ubiquitous Computing*, 7(1), pp. 70-79, 2003.

[19] M. Weiser and J. S. Brown, "The Coming Age of Calm Technology," 1996. Available at http://www.johnseelybrown.com/calmtech.pdf.

[20] M. Weiser, "The Computer for the 21st Century," in *Scientific American*, (September), 70-81, 1991.
[21] T. Kindberg and A. Fox, "System Software for Ubiquitous Computing," *IEEE Pervasive Computing*, 1(1), pp. 70-81, 2002.
[22] R. Campbell, J. Al-Muhtadi, P. Naldurg, G. Sampemane, and M. D. Mickunas, "Towards Security and Privacy for Pervasive Computing," in *Revised Papers International Software Security – Theories and Systems, Lecture Notes in Computer Science 2609,* pp. 1-15. Springer Berlin / Heidelberg. 2003
[23] R. Want, T. Pering, G. Borriello, and K. I. Farkas, "Disappearing Hardware," *IEEE Pervasive Computing*, 1(1), pp. 36-47, 2002.
[24] M. Weiser, "Some Computer Science Issues in Ubiquitous Computing," *Communications of the ACM*, 36(7), pp. 75-84, 1993.
[25] G. Roussos, A. J. Marsh, and S. Maglavera, "Enabling Pervasive Computing with Smart Phones," *IEEE Pervasive Computing*, 4(2), pp. 20-27, 2005.
[26] G. D. Abowd and E. D. Mynatt, "Charting Past, Present, and Future Research in Ubiquitous Computing," *ACM Transactions on Computer-Human Interaction (TOCHI)*, 7(1), pp. 29-58, 2000.
[27] I. A. Junglas and C. Spitzmuller, "A Research Model for Studying Privacy Concerns Pertaining to Location-Based Services," in *Proceedings of the 38th Annual Hawaii International Conference on System Sciences,* p. 180. IEEE Computer Society Press, 2005.
[28] D. A. Norman, *The Invisible Computer: Why Good Products can Fail, the Personal Computer is so Complex, and Information Appliances are the Solution*. Cambridge, MA: MIT Press, 1998.
[29] P. Ibach and M. Horbank, "Highly Available Location-based Services in Mobile Environments," in M. Malek, M. Reitenspie, and J. Kaiser (Eds.), *Revised Selected Papers Service Availability: First International Service Availability Symposium, ISAS 2004, Lecture Notes in Computer Science 3335,* p. 134. Springer Berlin / Heidelberg, 2005.
[30] E. Toye, R. Sharp, A. Madhavapeddy, and D. Scott, "Using Smart Phones to Access Site-Specific Services," *IEEE Pervasive Computing*, 4(2), pp. 60-66, 2005.
[31] F. Zhu, M. W. Mutkas, and L. M. Ni, "Service discovery in Pervasive Computing Environments," *IEEE Pervasive Computing*, 4(4), pp. 81-90, 2005.
[32] A. Agostini, C. Bettini, N. Cesa-Bianchi, D. Maggiorini, and D. Riboni, "Integrated Profile and Policy Management for Mobile-orientated Internet Services," TR-WEBMINDS-04, Web-Minds, Milan, Italy, 2003. Available at http://webmind.dico.unimi.it/arch/TR03.pdf.
[33] E. de Lara and K. I. Farkas, "New Products," *IEEE Pervasive Computing*, 4(4), pp. 4-7, 2005.
[34] J. Kerridge, K. Barclay, and J. Savage, "Groovy Parallel! A Return to the Spirit of occam?," in J. Broenink, H. Roebbers, J. Sunter, P. H. Welch, and D. Wood (Eds.), *Communicating Process Architectures 2005 (WoTUG- 28),* IOS Press, Amsterdam, The Netherlands, 2005.
[35] "Grand Challenges in Computing Research," British Computing Society, 2004. Available at http://www.ukcrc.org.uk/gcresearch.pdf.
[36] R. Milner, "Ubiquitous Computing: Shall we Understand It?" British Computing Society, 2006. Available at http://www.bcs.org/upload/amaxus_pdf/amaxus_conWebDoc_2578.pdf.
[37] K. Chalmers J.Kerridge and I Romdhani, "Performance Evaluation of JCSP Micro Edition: JCSPme, Proceedings of CPA 2006, Peter Welch, Jon Kerridge, and Fred Barnes (eds), IOS Press, 2006, pages 31-40
[38] JCSP download web site, www.jcsp.org