Computing Migration Through Context Awareness and
Context Transmission

Zhaohui Cheng, Kenneth Chin, Yuan Luo
Middlesex University,
Bounds Green, London, UK N11, 2NQ
{m.z.cheng,k.chin,y.luo}@mdx.ac.uk

Abstract. With the development of wireless communication and portable devices, people can now enjoy
the benefits of mobile computing. But due to the nature and design of portable devices, it is unlikely
portable devices can provide certain functionalities with the same quality as fixed devices. So during
mobile computing, it is not trivial to attain computing within a fixed network environment with high-power
workstations. A means of discovering available computing resources and enabling manipulation of them to
cooperate with mobile devices to maintain the continuous computing with better quality within an active
environment is required. In this paper we present a novel architecture that facilitates the detection and
manipulation of resources using context awareness and context migration methods in mobile computing
environments.

1. Introduction

The emergence and development of two technologies, portable computers and wireless communication,
allow users to move about with computing power and network resources at hand. Computers are
shrinking in size, allowing many to be held by hand despite impressive computing capabilities. Pocket
PCs and wearable computers are now viable commercial products. At the same time, the bandwidth of
wireless links keeps increasing and new wireless technologies emerged to serve different
communication requirements. Bluetooth is used for personal area network within a short distance,
whereas wireless LAN is employed for high-speed communication in local area, and wireless cellular
networks are built to provide wide area mobile communication. These changes have increasingly
enabled people to access their personal information, corporate data, and public resources “anytime,
anywhere”. For example, now journalists can use personal digital assistants (PDAs) to edit newspaper
articles at press conferences and send the breaking news swiftly back to the headquarters to be
published.

But for the native characteristics of portable devices, such as battery resource, wireless communication
quality, display size, etc., it is unlikely on portable devices to provide the same computing services with
the same quality as on fixed devices with fixed network connections. We need a means of discovering
available computing resources in the surrounding and enabling cooperation with mobile devices to
maintain the continuous computing with better quality within the active environment.

The contribution of this paper is to present a novel architecture to enable the transfer of computation
among devices for better computing quality through context awareness and context migration methods
without interrupting the current service processing. To this end, the following of this paper is organized
as follows. Section 2 introduces a case study scenario to demonstrate the need and analyze the requirements for the novel approach. Section 3 discusses the related works. Section 4 gives an overview of the architecture and details the function of each component. Section 5 discusses the implementation details. Section 6 concludes and outlines future works.

2. Scenario and Motivations

To illustrate the main motivations and requirements of the new architecture, we introduce a case study scenario and try to show the inadequacy of traditional approaches and drill down the basic requirements for the possible solutions.

2.1. Case study scenario

On the way to the company headquarter in London, Susan notices that, according the schedule, one important conference in New York is holding and she must participate in this videoconference. So she takes out her PDA and starts up the conference software (Super NetMeeting) and connects to the video server. Her PDA supports 3rd generation wireless cellular network, wireless LAN and Bluetooth communications. Although Susan can use her PDA to follow the proceeding of the conference, she is not satisfied with the quality of display and expensive tariffs for the real-time video communication. Half an hour later, Susan arrives at her office. There is a big intelligent screen (IScreen) on the wall, which supports high-speed data communication and with an embedded videoconference system. Susan wants to use the IScreen to follow the conference with better display quality and at the same time without losing the important speech from the chief executive officer. The PDA transfers the video stream to the IScreen.

After the speech, the chief financial officer will deliver a financial report and Susan wants to show the report to Jennifer, one of her colleagues, at the other part of the building. When Susan takes the PDA and walks out of her office, the video stream of the conference is transferred back to the PDA. Passing through the corridor, when Susan enters Jennifer’s office, the video stream migrates to a workstation in the room, which has a conference software and network connection.

2.2. Requirements of Solutions

The above scenario consists of two parts. In the first part, the video stream transfers from the PDA to the IScreen, which concerns computing migration for better quality. Videoconference is a type of computing service, which has two major requirements: a basic computing device with the conference software and a network connection. How do we support conference-computing migration for better quality?

The first requirement (A) is that PDA must sense the surrounding environment to find out a possible computing host that satisfies the computing service requirement with better quality. To satisfy the requirement, the mobile devices must express the service requirements in some way, and similarly the computing environment also need to have representation of its abilities. Furthermore, mobile devices must be able to sense changes in the environment, according to its current locality, and dynamically make decision whether or not to migrate the computing service.

The second requirement (B) is that the PDA and the new computing host must support computing migration. Computing migration need the new host to rebuild the service’s current running context,
including application context and communication context. So the system must be able to present the possible contexts and support contexts transfer “on the air”. In this scenario, network communication must support changes of end entities, which can be fulfilled by new protocols such as Mobile IPv6 [1]. This topic is out of the scope of this paper.

In the second part, the video stream moves back to the PDA when Susan leaves her office, which mandates that computing service must follow personal movements. In fact, this requirement (C) can be satisfied by the context migration for requirement B, with the support of a new concept: Service Anchor. In this paper, personal mobility is based on the assumption that a person always takes a mobile device. That mobile device is the service anchor. The service moves back and forth between service anchor and the environment, when the anchor device moves around.

2.3. Inadequacy of Traditional Approach: Context-aware Mobile Code

Context-aware computing, brought forth firstly by the researchers at Olivetti Research Ltd. [2] and Xerox PARC Laboratory [3], tries to explore and provide an infrastructure to support context-aware applications. Such applications can take advantage of the dynamic environmental characteristics to serve user needs effectively and adaptively without consuming too much of the user’s attention. Context [4], representing the environmental characteristics, is the set of environmental states and settings that either determines an application’s behavior or in which an application event occurs and is interesting to the user. The context could include computing context, user context, physical context, time context, etc. Currently, some researchers try to combine another major mobile computing technology, mobile agent, with context-aware computing to support application migration [5,6]. A mobile agent [7] can be seen as a self-contained packet, including a piece of code and state information, which is capable of moving to and executing at a host. The agent may travel to more than one host in order to complete its task. Although we can use mobile agent to enable some sort of computing migration, there are some problems using context-aware mobile code in the above scenario. First, mobile agents must be written in portable code to execute in mobile agent execution environments on heterogeneous host platforms. It is difficult to implement agents that need to make changes to the host kernel states, which is required in the above scenario for continuous communication. Second, it is complex to support large tasks with multiple agents and frequent transmission of large pieces of mobile agents will waste bandwidth.

In fact, the computer service on the new host indicated in requirement A does not need to be the same with the one on the mobile device. The two services could be on different operating systems with different implementations, but supporting the same function. Our solution is based on more general view on computing service. We focus on computing service migration, instead of application migration. First, we decouple the service and its implementation. Many communication-related services have different implementations on different platforms, but these implementations are service compatible. For example, we can use Internet Explorer on Microsoft Windows™, Netscape on Solaris, and Mozilla on Linux to browse the same web site. In fact, even for some complex services, we can find compatible implementations, such as a word processor. Open Office can edit the files created by Microsoft Office™. Second, we decouple the applications and their running contexts, resulting in a new kind of application, context-rebuildable application. This is the difference between Super Netmeeting used by Susan and Microsoft Netmeeting™. The application running context, different from the process context
in the operating system, is the persistent Information related with the current instance of a service, for example, for web browser, the running context could be just the current web page’s universal resource location (URL); for telnet, the running context is just the current TCP connection. Many applications could be modified easily to context-rebuildable applications, such as telnet, web browser, Real Player™, etc. In the following part, the computing service refers to the context-rebuildable computing service.

3. Related Works

In context-aware computing field, the most similar researches to ours are Call Forwarding [2] and Teleporting [3,8]. In Call Forwarding system, PBX will forward the telephone calls to the destination user’s nearest phone, with the help of location system taken with the user. Teleporting system uses virtual terminal software to map the user interface dynamically onto the surrounding computer and communication facilities. The difference between our work and these pioneer researches is that we create a new architecture based on a more general view of computing and this architecture supports context migration “on air” and new context-rebuildable applications to support continuous computing with better quality. On the other hand, they focused on location-aware application, cooperating with active badge for location sensing. Location-Oriented multimedia in [6] presents a framework to support mobile multimedia applications using mobile code, which also uses the videoconference scenario. And many research on context representation and transmission are also on-going. Schilit in [9] uses key-value pair to hold the actual context data. [6,10] employ Standard Generic Markup Language (SGML) to model the context and write resource advertisement. Other methods such as event-driven [11], tuple space [12,13] are also tried to express and exchange context information.

4. Architecture

4.1. Overview

This architecture tries to enable computing migration between the mobile device and its surrounding computing environment. To satisfy requirement A mentioned in section 2, we make use of the context-aware technology to sense the surrounding computing environment to find out the possible new host for the computing service. To satisfy requirement B, we take advantage of context migration to rebuild the running environment on the new host. Decoupling the running context and the implementation of services enables computing services to easily transfer between different computing devices. By introducing service anchor, the architecture realizes the personal mobility to meet requirement C.
The computing service is started up on the mobile device, which is called the service anchor. When the service anchor moves around, it can sense the environment’s computing context, which could be computing devices, network facilities, location information, context-rebuildable service implementations, etc. If the anchor’s context agent (ACA) senses the existence of context agent in the surrounding environment, it will send the current computing service’s requirements and location information to the environment context agent (ECA), under the security protection. ECA, by comparing its abilities and ACA's requirements, finds out if there are hosts available in usage cost field that can provide the same computing service with better quality. If such hosts are available, ECA will select a batch of candidates from them as the possible new computing hosts based on its policy. The decision criteria in the polices could include security requirements, the distance between current anchor device and possible candidate host, possible candidate host’s resource usage, etc. ECA sends to ACA the candidates, in priority order. ACA selects one or some candidates from the ECA's offer, based on its own policy, which could include security requirement, affordable charge, distance, etc. then transfers the computing context to the selected host either automatically or with the person’s involvement. The new host will construct the running context and start up the related application for the service. While the just started context-rebuildable application on the new host is serving the computing requirement, the anchor device also makes contact with the new host to enable the service to transfer back when the anchor moves out of the current environment. Our architecture also supports context-aware mobile code, if the mobile code running environment is registered as one context ability.

4.2. Components’ Functions

The overall architecture is shown in figure 1. Each component of the service anchor and the environment has a different role and in the following part, we will look at each component in turn.

- **Context Agent**: The pair of agents, through transmitting contexts, negotiate the contract for computing service migration. ACA sends the service requirements to ECA, and ECA returns the
candidate hosts based on the devices’ ability, security requirement and the available location information. After selecting one candidate from ECA’s offer, ACA transfers the application context and communication context to the new computing host. After migration, ACA still contacts ECA to enable the service to move back on demand. It could be only one ECA in the whole surrounding environment that administrates all the devices, and it is also possible to use separate ECA on each device.

- **Context Policy:** The policy on the ECA is used to find the truly available devices from the possible candidate set which includes all the devices whose abilities can satisfy the service’s requirement. Sometimes, the device’s ability is strong enough, but some required non-preemptive resource is already being used. Some devices do not allow the anchor device to access the services for security reasons. Some devices are too far away from the anchor’s current position. ECA selects the proper devices as the migration candidates, according to the preset rules. ACA’s policy component selects one host or a host set from ECA’s offer based on security, expense and location considerations. The policy on ACA can prevent the context transferring to an un-secure host, and provides the user with the best performance and expense ratio selection.

- **Context Environment**
  - **Application Requirement Context:** A database stores the requirements (hardware and software specification) for all the installed context-rebuildable applications in the system. When one service implementation is installed, the implementation should register the basic requirements for the running environment when fulfilling such service. For example, the video conference service needs a network connection with speed greater than 1.5Mbps, a camera, a microphone, a speaker, a screen and one application supporting Real-Time Transport Protocol (RTP).
  - **Application Context:** A database holds the application level running contexts in which context-rebuildable applications are running. These contexts include persistent information related to the service’s instance. For the videoconference software, it could include access password, session password.
  - **Location Context:** The current location of the anchor device in ACA and available devices’ location in the environment on ECA. In many context-aware computing researches, location sensing and location context is the main point. This kind of application is based on pervasive computing assumptions [14] and researchers try to improve the accuracy of location sensing [15]. This architecture can make use of the advanced location system to improve the system automation, but it can also use some basic location sensing technologies such as the broadcast of access points in the wireless LAN for proximate location and people involvement in the final host selection.
  - **Communication Context:** We separate the communication related information from the application level running context, because it’s more difficult to be transferred than the application level context. The communication context includes the necessary information that can be used to reconstruct the communication session under the kernel part. For example, a TCP session needs remote entity’s address and port number, the two current sequence numbers and the window size. To enable communication transfer, the remote entity or some proxy along the communication path must support special protocol such as Mobile IPv6 which supports end entity changes.
  - **Application Ability Context:** A database stores the services implemented using
context-rebuildable applications and their locations in the environment. The information in the database will be compared with the anchor’s service requirement to find out the possible host candidates. For the former scenario, the item in the database could be (videoconference service, Super Netmeeting, RTP).

- **Hardware Context:** A database saves the hosts’ hardware information including network connections, which is related to the context-rebuildable applications. The hosts should register this kind of information with the ECA.

- **Context Security:** The security schemes deal with the authentication, authorization and accounting (AAA). The context security must provide mutual authentication between ACA and ECA and protect the context transmission. In some circumstances, it would be necessary to charge for computing migration. So accounting service should be supported.

- **Communication Context Transmission Primitive:** Two primitives are used to transfer communication context. Down Call is used to force the kernel to transmit the communication context to the new computing host. Up Call will be triggered after the host receives the context and rebuilds the necessary structures such as socket in the kernel.

5. Design and Implementation Details

5.1. Context Representation

We use the simple but powerful enough key-value pair to express the application requirement, running context, environment abilities, hardware information and communication contexts. Each context information is described by an Attribute Value Pair (AVP) and a serial of attribute pairs (Attribute List) can form a group with ‘Group’ type to express one complex context who has more than one attribute. An AVP with value of type ‘Group’ implies that the Data field is actually a sequence of AVPs. It is possible to include an AVP with a ‘Group’ type within a Group type, that is, to nest them, to support context hierarchy. Applications can define new AVPs to express their own contexts. This method is compatible with the registry format in Windows. The Attribute List’s formal specification is:

```
Context=Context Header ; Attribute List
Context Header="Context Header" ; Len
Attribute List= AVP | AVP; Attribute List
AVP=Type: Len: Key: Type: Len: Value
Value=Single Value | (Attribute List)
Type=Char | Short Integer | Integer | Long | Unsigned Short Integer | Unsigned Long | IPv4 Address | IPv6 Address | String | Bits | NULL | Group
Len=bit size of the following element after Len field.
Single Value= the normal value that doesn’t contain any other Attribute List.
```

Figure 2. AVP Format

For easy explanation, the specification uses semicolon as the separator between AVPs, and bracket to express the border of AVP group. In the implementation, Len field can help the interpreter find the borders of AVPs and groups. Every Type field uses 8 bits and Len field occupies 32 bits in the expression.
One concrete example about TCP context shows how this key-value pair scheme works. The key for Communication Context is pre-defined character ‘c’, which includes a handle expressing the reference to the context whose key name is ‘h’, and the TCP related information whose key is ‘p’. The TCP related information is an AVP group. The key names in the group are “ra” for remote address, “rp” for remote port, “sq” for sequence number, “ac” for acknowledge number and “wd” for window size. The context AVP expression is:
“Context Header”:992:Char:8:’c’:Group:912:(Char:8:’h’:Unsigned Integer:32:0x001; Char:8:’p’:Group:704:(String:16:”ra”:IPv6 Address:128:0x3ffe::1; String:16:”rp”:Unsigned Short Integer:16:38; String:16:”sq”:Unsigned Integer:32:0xf0; String:16:”ac”: Unsigned Integer:32:0xf; String:16:”wd”:Unsigned Short Integer:16:0xf))

<table>
<thead>
<tr>
<th>Rule #1: In ECA</th>
</tr>
</thead>
<tbody>
<tr>
<td>IF Host allow SA access AND Host’s Resource is available THEN</td>
</tr>
<tr>
<td>SET Host as a Candidate</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Rule #2: In SCA</th>
</tr>
</thead>
<tbody>
<tr>
<td>IF Candidate is Secure for Service AND Loan of Candidate &lt; 10 dollars per hour AND Distance between two devices &lt; 10 meters THEN</td>
</tr>
<tr>
<td>SET Candidate to Selection Set</td>
</tr>
</tbody>
</table>

Figure 3. Policy Rules Example

We express migration policies by lists of rules, which follow the form: IF conditions THEN actions. See Figure 3 for an example. The policy engine will refer other contexts information, such as the security context, resource usage status, location information, etc. to determine the migration destination candidates. For security context, now we use static configuration file, such as host.allow, host.deny, user.allow, user.deny, service.allow, and service.deny. Pre-shared security information is used in authentication method and Diffie-Hellman key agreement to establish session keys. In the future work, we will introduce the work of security policy researches into the architecture.

5.2. Workflow in the Architecture

Figure 4. describes the basic steps for a migratory computing service to move forth and back between the surrounding environment and the mobile service anchor. This architecture doesn’t apply the high accurate location system, instead, uses the proximate selection method. People are involved in the destination host chosen.
1. ECA broadcasts its existence
2. I am Michael, need your help
3. Mutual Authentication and Session Key Establishment
4. Transfers the service requirements
5. Compares the ability and hardware context with the requirements to find the possible candidates
6. Based on policies to find the candidates and send the candidate set to ACA
7. Based on policies to select object hosts from ECA’s offer
8. People selects a new destination host
9. Transfers application context and communication context to new host
10. Receives and reconstructs the context and starts up the service
11. Requires the host to transfer back the service
12. Transfers application context and communication context
13. Receives and reconstructs the context and starts up the service

Figure 4. Computing Migration Workflow

5.3. Interfaces in the Architecture

The programmer makes the computing service migratory by including the following interfaces:

1. **Register_Requirement(Application Name, Requirement Context)**
   When the application is being installed, the upper two inferences are called to register the service’s requirements and this implementation’s ability.

2. **Register_Ability(Application Name, Ability Context)**
   When the application starts normally and the running context has been established, the application uses `Register_Context` to register its context, which will also update the used resource’s status.

3. **Register_Context(Application Name, Instance Handle, Running Context)**
   When the application starts within the migration mode, the application gets the migrated context through `Retrieve_Context`.

4. **Running Context Retrieve_Context(Application Name, Instance Handle)**
   When the application starts within the migration mode, the application gets the migrated context through `Retrieve_Context`.

5. **Down_Call(Migration Handle, Communication Context Reference, Security Context)**
   `Down_Call` requires the operating system to transfer the communication context under the security scheme protection indicated by Security Context. Migration Handle is used to distinguish migrated application. If the application implemented under Unix using socket for communication, Communication Context Reference is the socket handle and the system call `ioctl` with extension can be used as `Down_Call`.

6. **Communication Context Up_Call(Source Address, Migration Handle, Security Context)**
   `Up_Call` consists of two parts. The first part will use the Source Address, Migration Handle and
Security Context to ask the kernel to accept and rebuild the communication context for the application whose handle is Migration Handle migrated from the host with Source Address under protection using Security Context. In the second part of this interface, the system informs the application the rebuilt communication context, if using socket, which could be socket handles. For the existence application, one alternative method is using another application to collect the contexts from the available system information and call the upper interfaces and ECA can rebuild the context by setting system configuration files or environment variables, even command line for the migrated application.

6. Conclusions and Future Works

In the mobile computing world, it is a big challenge to take advantage of the mobile device’s surrounding computing environment to meet users’ computing needs. We have presented a new architecture, which enable the computing services to migrate between a mobile device and its environment through employing context-awareness and context transmission methods. Our architecture is based on a more general view on computing service, which decouples the service and its implementation and also the implementation and its running context. Our architecture can also be used to support context-aware mobile code, if the mobile code running environment is registered as a kind of context ability. As a novel method, our work needs further improvements. A better policy process is needed, and wild scale security scheme is still a big challenge. The architecture needs to integrate advanced location technologies to improve the automatic migration. We need further explore the essence of computing service for migration ability.

References


