MobiPADS: A Reflective Middleware for Context-Aware Mobile Computing

Alvin T.S. Chan, *Member*, *IEEE*, and Siu-Nam Chuang

Abstract—Traditionally, middleware technologies, such as CORBA, Java RMI, and Microsoft's DCOM, have provided a set of distributed computing services that essentially abstract the underlying network services to a monolithic "black box." In a mobile operating environment, the fundamental assumption of middleware abstracting a unified distributed service for all types of applications operating over a static network infrastructure is no longer valid. In particular, mobile applications are not able to leverage the benefits of adaptive computing to optimize its computation based on current contextual situations. In this paper, we introduce the **Mobi**le **P**latform for **A**ctively **D**eployable **S**ervice (MobiPADS) system. MobiPADS is designed to support context-aware processing by providing an executing platform to enable active service deployment and reconfiguration of the service composition in response to environments of varying contexts. Unlike most mobile middleware, MobiPADS supports dynamic adaptation at both the middleware and application layers to provide flexible configuration of resources to optimize the operations of mobile applications. Within the MobiPADS system, services (known as mobilets) are configured as chained service objects to provide augmented services to the underlying mobile applications so as to alleviate the adverse conditions of a wireless environment.

Index Terms—Middleware, mobile applications, mobile computing support services, mobile environments.

1 Introduction

The rapid growth of wireless technology, coupled with advances in mobile computing devices, such as PDAs, cellular phones, smart cards, etc., have fundamentally changed the landscape of distributed computing. Unlike fixed network computing, mobile computing operating in a wireless environment suffers from limited resources and often experiences constant changes in the availability of resources that significantly impacts the underlying performance of the system. Generally, to reduce its size and weight, mobile devices are often equipped with limited resources, such as a small display size, limited power, comparatively lower CPU speed, and memory size. In addition, the operating environment of these devices over a wireless network is often hostile and the availability of resources is changing, where the connection may be subjected to high error rates and temporary disconnections. The unique characteristics of mobile computing have resulted in a recent drive to revisit the fundamental design and challenges of distributed computing to include mobility support.

Middleware technology has played in important role in facilitating the development of distributed applications by abstracting the network to create a single-system image, thereby providing network transparency. Importantly, middleware technologies, such as CORBA [1], Java RMI [2], and Microsoft's DCOM [3], provide a set of distributed computing services that essentially abstract the underlying network services to a monolithic "black box [4]." With the black box abstraction, the middleware is able to hide the details of the underlying low-level network operations and interfaces, thus allowing the application developer to focus on the important aspects of implementing the application's logic. While the distributed application development model based on a monolithic middleware paradigm clearly benefits the applications in terms of ease of development and robustness, this may not be true in developing distributed mobile applications. In a mobile operating environment, the fundamental assumption of middleware to abstract a unified distributed service for all types of applications operating over a static network infrastructure is no longer valid [5]. In particular, mobile applications are not able to leverage the benefits of adaptive computing to optimize their computation based on current contextual situations. For example, a mobile Web client application browsing a graphical Web site can request that the middleware service progressively degrades the picture quality if it detects a drop in bandwidth availability or increased rates of error. The benefits of middleware in supporting context-awareness can be extended to provide mechanisms for the dynamic deployment and reconfiguration of underlying services to optimize the overall operations of the mobile application. These services may be configured to provide optimized delivery of contents to minimize bandwidth usage, alleviate the impact of error rates by locally performing error control over the wireless link, data compression on contents, and content transcoding. The placement of adaptation can occur either at the middleware layer or at the application. By providing adaptation at the middleware layer, the application is relieved from the need to monitor the environment and can leave the adaptation completely to the decision of the middleware. In this application-transparent approach, the middleware is designed to provide best effort adaptation to general mobile computing, where context information is completely hidden from the application. This approach significantly limits the amount of adaptation space available

• *The authors are with the Department of Computing, The Hong Kong Polytechnic University, Hong Kong SAR, People's Republic of China. E-mail: {cstschan, cssiunam}@comp.polyu.edu.hk.*

Manuscript received 31 Dec. 2002; revised 6 July 2003; accepted 5 Aug. 2003. Recommended for acceptance by M. Oivo and M. Morisio.
For information on obtaining reprints of this article, please send e-mail to: tse@computer.org, and reference IEEECS Log Number 118783.

to the middleware to optimize processing since in the event of adverse conditions, the application itself is in the best position to make critical decisions on operating conditions and, hence, on the adaptation strategy.

In this paper, we introduce the **Mobi**le **P**latform for **A**ctively **D**eployable **S**ervice (MobiPADS) system, which is a middleware that extends the WebPADS system [6]. MobiPADS supports the active deployment of services for mobile computing over a wireless environment. Importantly, MobiPADS is designed to support context-aware processing by providing an executing platform to enable active service deployment and reconfiguration [7] of the service mix in response to an environment where the context varies. Unlike most mobile middleware, MobiPADS supports dynamic adaptation at both the middleware and application layers to provide flexible configuration of resources to optimize the operations of mobile applications. Within the MobiPADS system, services (known as mobilets) are configured as chained service objects to provide augmented services and protocols to the underlying mobile applications so as to alleviate the adverse conditions of a wireless environment. An abstraction of service object interactions and configurations is expressed in a high-level declarative language written in XML format, while the dynamic composition of service objects is customizable directly by the applications to adapt to the operating context of the environment. The following section highlights the design issues of mobile middleware. It presents the concept of reflection and how this can be used to effectively model a highly configurable middleware system based on metaobject reflection techniques. The MobiPADS system employs the use of reflection and metaobject as the baseline technology to achieve robust and efficient adaptation of service objects in a context-aware middleware system for mobile computing.

2 Middleware for Mobile Environment

An important requirement of a middleware system to support mobile computing applications is the provision of a highly configurable and adaptive execution environment that dynamically reacts to changes in operating context. This requirement translates to the need for middleware to organize and implement its system components as a collection of services that are highly configurable and robust enough to enable the system itself to respond to the varying conditions in the environment. In addition, mobile applications are presented with open programming interfaces to enable application introspection and, if required, to reconfigure the underlying services to adapt to changes in the environment.

2.1 Reflective Middleware

Computational reflection [8] is a unique approach to achieving adaptation and reconfiguration in a mobile middleware system. In general, reflection characteristic refers to the ability of a system to monitor its computation and possibly change the semantics of the way it is performed. In other words, a reflective middleware possesses the unique ability to model itself through self-representation, such that manipulation of its behavior may be changed through *introspection* and *interception* [9]. In this case, introspection refers to the ability of the system to observe and, therefore, reason about its own state, while interception is the ability of the system to modify its own execution state or its own interpretation or meaning. A middleware system with self-representation is causally connected if changes made to the self-representation directly affect the implementation of the middleware. The opposite is true if changes to the middleware implementation will change the self-representation.

2.2 Service Adaptation and Configuration

Middleware systems operating over a wired environment are often static, so the service configuration between the components does not internally react to the executing context. To allow middleware systems to respond to changes in the operating context, it is necessary that notification mechanisms are in place to enable the operating system to monitor and trap context events. These events represent important signals to both the middleware and applications to perform reflective computation and, if necessary, to adapt and deploy a new service configuration to the system. The questions still remain: *How do the middleware and applications know what context information to react to, and how do they know how to react to the context changes?* There are three levels of adaptations that can be applied to the mobile middleware system:

Interservice adaptation refers to adaptation of the middleware system by reconfiguring the composition of the service objects. The reconfiguration is triggered by the occurrence of a composite set of events that represents changes in the context environment which signals the middleware to react to it. For example, the adaptation may be triggered by the occurrence of a drop in bandwidth availability and signal that the error rate is above a certain level. In this case, a reconfiguration of the service composition at the middleware is necessary to alleviate the conditions. This can be done by introducing a service object to perform content transcoding of images from color to grayscale in order to improve bandwidth utilization, and a service object to perform local error control mechanism over the wireless link. The service configuration profile associated with composite events can be maintained by the middleware or supplied by the application.

Intraservice adaptation refers to adaptation of the middleware system by allowing the service object to react to a set of composite events. The service object is required to subscribe to the contextual events of it is interested in being notified. Upon triggering the event, the service object can be programmed to change its behavior to react to the executing context. For example, a transcoding service object may be programmed to receive an event associated with the change of bandwidth availability. For a high availability of bandwidth, the service object may transfer full-colored graphical objects, while medium availability may result in lower color depth, and low availability may result in gray-scale image.

Application adaptation refers to adaptation of the application by reacting to a change in context information. The application is required to subscribe to a set of composite events that it is interested in monitoring. Upon the

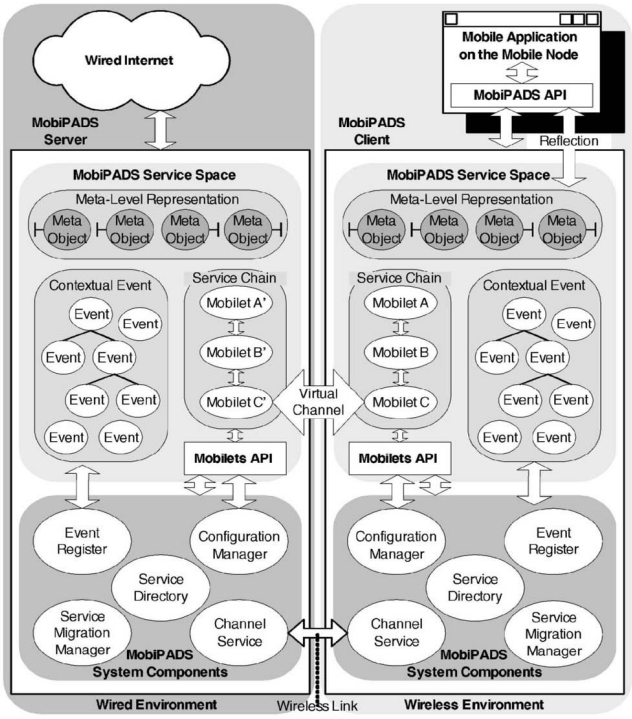

Fig. 1. The MobiPADS system architecture.

occurrence of the event, the application may choose to adapt its application logic to react to the executing context. For example, a video streaming application may be programmed to reduce the frame transmission rate upon detecting a drop in bandwidth availability. Alternatively, the application may directly interact with the underlying middleware to request interservice or intraservice adaptation.

3 System Overview

As shown in Fig. 1, the MobiPADS platform is composed of two agents: a MobiPADS server at the wired network and a MobiPADS client at a mobile device attached to the Internet through wireless or cellular networks. The two agents marshal the traffic over the wireless link and provide an optimal operating environment for mobile applications. The MobiPADS server is located at or close to the network of the wireless access point, to which the mobile device is connected. The MobiPADS server is designed to support multiple MobiPADS clients and is responsible for most of the optimization computations. The MobiPADS client is an intermediary that provides a comprehensive set of network and system services for mobile applications. These services enable ease of introduction of context-awareness and adaptation for mobile applications, so that the mobile application can adaptively react to varying context environments.

3.1 The MobiPADS Framework

Fig. 1 shows that each MobiPADS agent is composed of two parts: the system components and the MobiPADS service space. The system components provide essential services for the deployment, reconfiguration, and management of user service pairs—the mobilet pairs, which form the units of service for execution under a MobiPADS environment. The system components also provide common facilities that serve mobilets, which in turn provide value-added services to the wireless environment. These mobilets can be added, updated and removed dynamically.

In the MobiPADS service space, a series of mobilets is linked together to form a processing chain—*the service chain*, which allows mobile applications to benefit from the aggregated functionalities of a collection of mobilets. Mobilets access the services of the system components though the mobilet API, which also provides interfaces to allow the system components to communicate and configure the mobilets. To monitor the contextual changes, the MobiPADS employs composable event objects that report any contextual change to entities that subscribe to them. The composition of events can be initiated at start-up time and also allows runtime modification to the event compositions. At the top level of the service space, there is a set of metaobjects that reflects the configuration of the composite events and service chain, as well as the adaptation policies. Through the metaobjects, both the middleware and the mobile application can inspect and reconfigure the event compositions and service chain when adaptation is needed.

3.2 Mobilet Service Model

A Mobilet (pronounced as mo-be-let) is a service entity that can be downloaded, pushed or migrated to a MobiPADS platform for execution within an environment. The name, mobilet, bears a strong resemblance to applet. Applets are active codes executed within Web browsers while mobilets are active mobile codes that run within the MobiPADS environment.

Mobilets exist in pairs: a master mobilet resides at the MobiPADS client and a slave mobilet resides at the MobiPADS server. A pair of mobilets cooperates to provide a specific service. A typical case would be: While a slave mobilet shares a major portion of the processing burden, the master mobilet instructs the slave mobilet what actions to take and also presents the processed output to the MobiPADS client.

The mobilets are chained together on the client in a specified order, and the corresponding peer mobilets are chained together in a nested order on the server-side. The service chaining model of mobilet supports a general service composition paradigm that enables utmost flexibility in deploying service aggregation, while providing ease of reconfiguration in response to the varying characteristics of a wireless environment. A consistent synchronization and data flow model can be established through the abstraction of channel objects and the employment of data encapsulation between services.

In order to support a robust service configuration, all mobilets can be dynamically deployed across a MobiPADS client and server. In other words, it is possible for a mobile node to carry with it relevant mobilets as it travels across foreign domains. As the need arises, mobilets from the client can be dynamically pushed to a MobiPADS server and configured to operate in a coordinated manner. Conversely, it is possible for a MobiPADS server to push

mobilets to a MobiPADS client to actively install new services to operate across a wireless link.

3.3 System Components

The components of the MobiPADS system are briefly described, as follows:

- *Configuration Manager*: The configuration manager is responsible for negotiating the connection between the client and the server. It also has a service controller for initializing, interconnecting, and managing the service objects.
- *Service Migration Manager*: The service migration manager manages the process of importing and exporting service objects between the MobiPADS server and the MobiPADS client. It also cooperates with the service directory to activate, store, and keep track of the changes made to the active service objects.
- *Service directory*: The service directory records all the known service types. The object codes are stored in a service repository, which is used for service activation and service migration.
- *Event Register*: The event register allows objects to register for event sources. When an event occurs, the objects that have registered for that event source are notified. Event sources include various changes in network status, machine resources status, and connectivity status.
- *Channel Service*: The channel service provides virtual channels for the service objects to communicate. Instead of opening separate TCP connections for each message, messages are multiplexed into a single persistent TCP connection, which then eliminates the overheads of opening new TCP connections and avoids the slow-start effect on overall throughput [10].

3.4 Context-Aware and Adaptive Service

The MobiPADS achieves context-awareness by using an event notification model, which monitors the status of the interested context and reports the event to the subscribed entities. The event notification model supports subscription from all entities within the MobiPADS platform. These include the system components, the mobilets and the mobile application, which can all act as subscribers to the event service. The event notification model allows the composition of several primitive events to form a hierarchical composition of events in the form of an event graph. For example, a composite event graph called "high_resource" can be composed of events that monitor the network, battery, and CPU utilization, where simultaneous occurrence of the individual events will signal the triggering of the composite event. Once composed and subscribed, the event service is responsible for maintaining, monitoring, and analyzing the primitive events and for matching them against the flow of the event graphs.

With the subscription of contextual events that are of interest to the MobiPADS entities, various degrees of adaptations can be applied to dynamically respond to the changing contextual environment. For example, a service-chain may be reconfigured to add and delete services, while the mobilet itself may respond to the contextual events by changing its operating behavior. At a higher level, applications running over a MobiPADS execution environment may choose to reflect upon an existing service composition and to dynamically change the composition by accessing the metarepresentation of the metaobjects. Through the reflected metachain, both the MobiPADS system and the mobile application can identify the mobilets running in the current service chain. To reconfigure the current service chain, an adapted metachain that specifies the new configuration will have to be supplied to the configuration manager, which will adjust the current service chain accordingly.

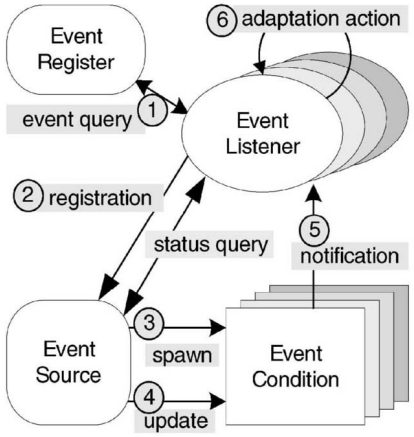

Fig. 2. The event—condition-action cycle.

4 Contextual Event Notification

One of the unique features of the MobiPADS system is the provision of a highly composable service chaining of mobilets that reacts and adapts to the varying characteristics of a wireless environment. To react dynamically to the changes in the environment, the MobiPADS system must provide an effective means to monitor the changes in the environment and to disseminate the relevant information to all of the interested entities.

4.1 Event Model

As shown in Fig. 2, the event model is comprised of three primary interacting entities: the event source, event listener, and event condition. The event source object is designed to decouple the binding of events from listeners and to provide the basic implementation for adding, removing and notifying event listeners. The object is actively deployable so that a MobiPADS client can push a new event source, upon its initialization, to a MobiPADS server. In other words, an event source, like a mobilet, is implemented as a serializable object that can be bound to a local context to provide abstract monitoring of the previously registered event on the server. An event listener is any object that is interested in a specific event and subscribes to that event.

In a mobile environment, a direct implementation of an event source is to report every change in the context that it monitors (for example, the bandwidth and error rate of a wireless link) as the context change over time. However, it is usually not the intention of an event listener to receive every event-update message, which may incur unnecessary

TABLE 1
Contextual Events for a Wireless Environment

Event Type	Description	Datatype / Units	Example
CPU	**The processor**		
CPU_type	The type of CPU	String / -	"Xcale PAX 250"
CPU_clockRate	The current CPU's frequency	Int /MHz	200
CPU_loading	The processor utilization level	Float /%	10.0
RAM	**The physical memory**		
RAM_size	Total size of the memory	Int /MB	64
RAM_freeSpace	Available free memory	Int /MB	40
Storage	**The secondary storage devices**		
Storage_name	The label or drive letter of the device	String / -	"Storage Card"
Storage_size	Total size of the logical device	Int /MB	512
Storage_freeSpace	Available free space of the device	Int /MB	302
Network	**The network interfaces**		
Network_name	The name of the network interface	String / -	"eth0"
Network_capacity	The max. capacity of the interface	Int /(bit/s)	1153436
Network_maxRate	The current max. data rate	Int /(bit/s)	341000
Network_delay	The measured round trip delay	Int /ms	50
Network_errorRate	The packet drop rate	Float /%	0.01
Network_disconnect	Report of disconnection	Int /(ms)	0
Network_handoff	Report of handoff between networks	Bool / -	False
Power	**The power level of the device**		
Power_maxUpTime	The maximum battery time	Int /mins	240
Power_remainUpTime	The remaining battery operation time	Int /mins	144
Power_batteryLevel	The remaining battery level	Float /%	60.0
Power_warning	Report of low battery level	Bool / -	False
System	**The MobiPADS system events**		
System_ClientCount	Number of clients connected	Int / -	15
System_MaxClient	Max. limit of client connections	Int / -	100
System_AllowMigrate	Report of migration support enabled	Bool / -	True
System_AllowDeploy	Report of active deployment enabled	Bool / -	True

computing overheads in processing these messages. Instead, the event listener should be provided with the option of constraining the scope associated with the status of an environmental context. To facilitate the provision of this option, the event listener is required to supply a description of the conditions under which the listener will be informed of any change. In this connection, an event-condition object is created that keeps track of the status of the event source, and issues a notification only when the specified condition is fulfilled. Each event listener has a corresponding event condition, and there can be multiple event-condition objects monitoring the same event source.

4.2 Contextual Information

Inspired by the work of the Global Grid Forum Performance Working Group [11], we have defined a list of context event types that are essential for adaptation in a wireless environment. As shown in Table 1, our list includes events from five sources: CPU, memory, storage device, network, and power. Besides these primitive contextual events, various information on the status of the MobiPADS system is represented by the contextual events generated by MobiPADS itself.

4.3 Event Composition

Thus far, we have described the abstractions of an event source and event condition to provide subscription and notification services by event listeners interested in the changes in the context. While this is sufficient for most applications, there are situations where more complex event compositions are required by service objects to monitor composite events occurring across various environmental contexts. To monitor the environmental changes in a more structured manner, an environment monitor object was

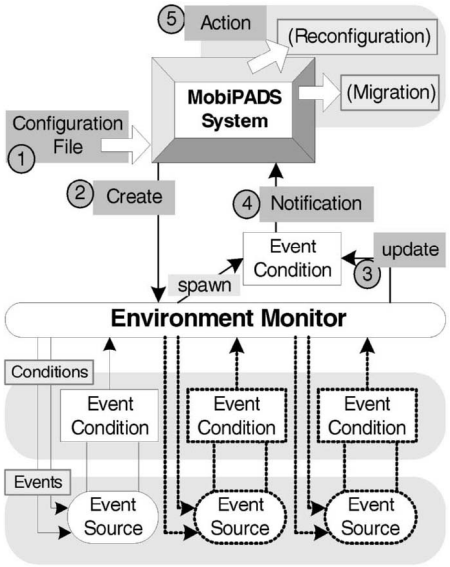

Fig. 3. Script-controlled event composition and adaptation.

created. As shown in Fig. 3, the environment monitor is a composite event object that inherits the characteristics of an event source object. Importantly, the environment monitor can be applied and bound to monitor multiple event sources. Thresholds and conditions can be set to compare against each reference value of a monitored event source. Only when *all* of the specified conditions are fulfilled will the environment monitor notify its listener, corresponding to an *AND* event model. We have also implemented the *OR* event model that will respond when any one of the specified conditions is satisfied. Furthermore, we can have different environment monitors representing different environmental setups. Adding another environment monitor that oversees all of these environment monitors can establish a hierarchical monitoring structure on the event sources, which can represent a flexible and detailed representation of a specific environmental setup. Fig. 4 shows a hierarchical composite event that consists of multiple levels of environment monitors. Using this structure, a very complex environmental setup can be composed.

Unlike the event systems [12], [13] for active databases, which are interested in what has already happened, our event system is interested in what is currently happening. In particular, an active database focuses on the history of the changes in data entities, whereas the event system of MobiPADS focuses on the current status of a wireless environment. Although the two models act in the same way when only a single event is involved, they behave differently when there is a composite event. In an active database, the action associated with a composite event will be carried out when the specified conditions of its primitive events are all fulfilled, irrespective of when the primitive events were generated. Some systems [13] allow a condition to be specified with an expiry period for an individual primitive event so that older events are discarded. However, these systems are still dealing with historical events.

When considering adaptation in a mobile environment, the status of monitored environmental contexts may change

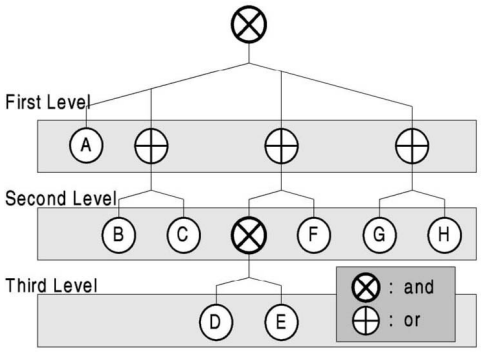

Fig. 4. Hierarchical event composition.

very rapidly. Therefore, a status report sampled a couple of seconds ago might no longer be valid. As such, it is important that before an environment monitor sends out any notification, it has to verify the concurrent occurrence of all of the primitive events that it has received. This ensures that the current environment does, in fact, reflect and satisfy the conditions. In other words, we need to ensure that the collection of interested events is obtained in real-time, within a prescribed time interval. This relates closely to a quasi-distributed snapshot. We employ a validation mechanism similar in vein to the distributed deadlock detection validation; the environment monitor queries all of the previously received events when these recorded events fulfill the conditions that trigger a notification. To ensure that all of these events are checked against the specified conditions, the records for all invalid events are removed. If the conditions can still be fulfilled after this, the notification is issued.

4.4 System Profile

To regulate the adaptive policies, the MobiPADS system utilizes XML to maintain a system profile that describes the metalevel configuration of the system components and entities. Fig. 5 shows the partial listing of document type definitions (DTD) of the system profile. Lines 1 to 3 allow the definition of the required events, specify the code-base of these events, and describe the events as either composite or primitive. Lines 4 to 8 define the relational operators for event composition, which allow the use of "AND," "OR," and "NOT" operators in manipulating the conditions. There is also a pair of "BEGIN" and "END" elements to enable the scope of the conditions to be specified, which supports the composition of a complex event. The environment element is specified, as shown in Fig. 5, on lines 9 to 13. It consists of one or more conditions, where each condition specifies an event, a relation operator and a reference value that will satisfy that condition.

For example, Fig. 6 shows a portion of a system profile, which contains a description of the service composition and reconfiguration. Lines 1 to 40 define four environments using composite events. Lines 2 to 16 define the first environment as "HighCPUAvailability," which specifies the following conditions:

(CPU_type = "XScale PAX 250") AND
[(CPU_clock_rate = 400 AND CPU_loading < 50%) OR
(CPU_clock_rate = 200 AND CPU_loading < 10%)] AND
(Power_warning = false)

```
1 <!ELEMENT EVENT (#PCDATA)>
2 <!ATTLIST EVENT codebase CDATA #IMPLIED code NMTOKEN #IMPLIED composite
  NMTOKEN "false">
3 <!ELEMENT PRIMITIVE_EVENT_LIST (EVENT)+>

4 <!ELEMENT BEGIN EMPTY>
5 <!ELEMENT END EMPTY>
6 <!ELEMENT AND EMPTY>
7 <!ELEMENT OR EMPTY>
8 <!ELEMENT NOT EMPTY>
9 <!ELEMENT RELATION (#PCDATA)>
10 <!ATTLIST RELATION operator NMTOKEN #REQUIRED>
11 <!ELEMENT CONDITION (EVENT, RELATION)>
12 <!ELEMENT ENVIRONMENT(NOT?,BEGIN*,NOT?,CONDITION,END*,(AND|OR)?)+>
13 <!ATTLIST ENVIRONMENT compositeName NMTOKEN #IMPLIED>
14 <!ELEMENT COMPOSITE_EVENT_LIST (ENVIRONMENT)+>

15 <!ELEMENT MOBILET (#PCDATA)>
16 <!ATTLIST MOBILET codebase CDATA #REQUIRED code NMTOKEN #REQUIRED>
17 <!ELEMENT CLIENT_META_CHAIN (MOBILET)+>
18 <!ELEMENT SERVER_META_CHAIN (MOBILET)+>
19 <!ELEMENT CHAIN_MAP(ENVIRONMENT,CLIENT_META_CHAIN, SERVER_META_CHAIN)>
20 <!ATTLIST CHAIN_MAP default NMTOKEN "false">
21 <!ELEMENT RECONFIGURATION (CHAIN_MAP)+>
22 <!ELEMENT MIGRATION (ENVIRONMENT)>

23 <!ELEMENT CONFIGURATION (PARAMETER, PRIMITIVE_EVENT_LIST,
  COMPOSITE_EVENT_LIST, RECONFIGURATION, MIGRATION)>
```

Fig. 5. The partial listing of the DTD of the system profile.

Lines 17 to 29 show another definition of an environment "LowCPUAvailability," which specifies the following conditions:

(CPU_type = "XScale PAX 250") AND
[(CPU_clock_rate = 400 AND CPU_loading > 80%) OR
(CPU_clock_rate = 200 AND CPU_loading > 60%)]

Lines 30 to 39 show two other environments, which are "HighBandwidth" and "LowBandwidth."

```
1  <COMPOSITE_EVENT_LIST>
2  <ENVIRONMENT compositeName="HighCPUAvailability">
3      <CONDITION> <EVENT> CPU_type </EVENT>
4      <RELATION operator="EQUAL">XScale PAX 250</RELATION></CONDITION> <AND/>
5    <BEGIN/> <CONDITION> <EVENT> CPU_clockRate </EVENT>
6              <RELATION operator="EQUAL"> 400 </RELATION> </CONDITION>
7      <AND/>  <CONDITION> <EVENT> CPU_loading </EVENT>
8              <RELATION operator="LESS_THAN"> 50.0 </RELATION> </CONDITION>
9      <OR/>  <CONDITION> <EVENT> CPU_clockRate </EVENT>
10             <RELATION operator="EQUAL"> 200 </RELATION> </CONDITION>
11     <AND/> <CONDITION> <EVENT> CPU_loading </EVENT>
12            <RELATION operator="LESS_THAN"> 10.0 </RELATION> </CONDITION>
13   <END/>  <AND/>
14   <CONDITION> <EVENT> Power_warning </EVENT>
15   <RELATION operator="EQUAL"> false </RELATION> </CONDITION>
16 </ENVIRONMENT>

17 <ENVIRONMENT compositeName="LowCPUAvailability">
18     <CONDITION> <EVENT> CPU_type </EVENT>
19     <RELATION operator="EQUAL"> XScale PAX 250 </RELATION> </CONDITION> <AND/>
20     <BEGIN/> <CONDITION> <EVENT> CPU_clockRate </EVENT>
21              <RELATION operator="EQUAL"> 400 </RELATION> </CONDITION>
22      <AND/>  <CONDITION> <EVENT> CPU_loading </EVENT>
23              <RELATION operator="GREATER_THAN"> 80.0 </RELATION> </CONDITION>
24      <OR/>  <CONDITION> <EVENT> CPU_clockRate </EVENT>
25              <RELATION operator="EQUAL"> 200 </RELATION> </CONDITION>
26      <AND/>  <CONDITION> <EVENT> CPU_loading </EVENT>
27             <RELATION operator="GREATER_THAN"> 60.0 </RELATION> </CONDITION>
28     <END/>
29 </ENVIRONMENT>

30 <ENVIRONMENT compositeName="HighBandwidth">
31     <CONDITION> <EVENT> Network_maxRate </EVENT>
32     <RELATION operator="GREATER_THAN"> 20000 </RELATION> </CONDITION> <AND/>
33     <CONDITION> <EVENT> Network_delay </EVENT>
34     <RELATION operator="LESS_THAN"> 100 </RELATION> </CONDITION>
35 </ENVIRONMENT>

36 <ENVIRONMENT compositeName="LowBandwidth">
37     <CONDITION> <EVENT> Network_maxRate </EVENT>
38     <RELATION operator="LESS_THAN"> 5000 </RELATION> </CONDITION>
39 </ENVIRONMENT>
40 </COMPOSITE_EVENT_LIST>
```

Fig. 6. XML segment specifying context composition.

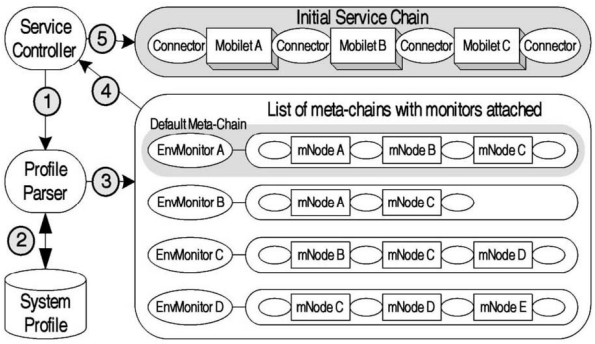

Fig. 7. Establishment of the metachains and initial service chain.

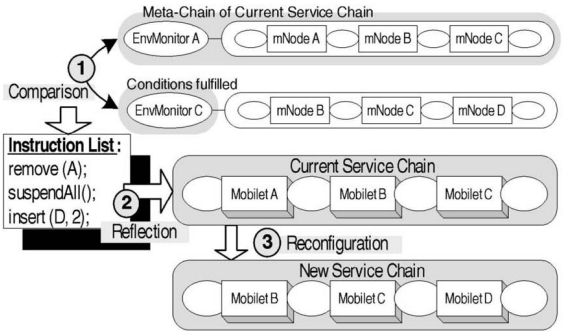

Fig. 8. Service reconfiguration process.

5 Dynamic Service Reconfiguration

To adapt dynamically to the changes in an environment, MobiPADS employs the environment monitor and event system to monitor and communicate the changes. After changes have been detected, the MobiPADS system can respond in two ways. The first way it can respond is by reconfiguring the current service chain. By adding and removing mobilets within the service chain, the optimum set of mobilets can be selected based on the constrained environment. The second way it can respond is by communicating the changes in the environment to each of the mobilets so that they can readjust their service provision to adapt to the mobile environment.

5.1 Active Deployment

As the number and variety of services being provided though the Internet continue to grow, a static service configuration that offers a limited set of functionalities will not be able to cope with the increasing demands placed upon these services. The MobiPADS system must be able to be updated dynamically with newly developed and upgraded functionalities, to ensure compatibility and to best match the operating environment.

To actively deploy the service code, the MobiPADS client has to carry with it the service code of the mobilets that it needs. The argument in favor of this approach is that the MobiPADS client should know in advance about the mobilets that it requires, whereas the MobiPADS server is considered to be more passive and has little information concerning the requested mobilets. Thus, it is a straightforward idea that the MobiPADS client should carry with it the service code of the mobilets and push this code to the MobiPADS server if necessary. However, the disadvantage of this approach is that the active service deployment procedure takes a significant time to complete under limited bandwidth and when the data size of the service codes is large.

5.2 Adaptation Policies

Fig. 7 shows the conceptual initialization procedures of the service chain. When a MobiPADS client starts executing, the service controller (of the configuration manager) invokes the profile parser, which will then load and process the system profile. Based on the description in the system profile, the default metachain is created, as well as a number of alternative metachains, which are specified in the DTD of the system profile, on line 17-18 in Fig. 5. The list of metachains is then returned to the service controller, which will then deploy the default service chain and the event monitors of the metachains. Each metachain is attached to an environment monitor, which regulates the time and conditions that determine when the reconfiguration is to take place. When all of the conditions of a specific environment monitor are fulfilled, the corresponding metachain will be reflected onto the current service chain, and reconfiguration will take place.

5.3 Service Chain Reconfiguration

Service chain reconfiguration takes place when the context environment changes to a state that fulfills all of the conditions of a specific environment monitor. The corresponding metachain will then be reflected onto the current service chain to best adapt to the changes. Fig. 8 shows the procedures of service chain reconfiguration when an environment monitor *EnvMonitor_C* is qualified for reconfiguration. First, the eligible metachain will be compared to the active metachain (of the current service chain), so that a list of instructions is generated to perform the actual operations needed for reconfiguring the current service chain. As shown in Fig. 8, the active metachain consists of *mNodes_A*, *mNodes_B*, and *mNodes_C*, while the new metachain consists of *mNode_B*, *mNode_C*, and *mNode_D*. The configuration manager compares each mNode in the active metachain to the new metachain, and any unmatched mNode in the active metachain will be marked for deletion. In this case, the mNode_A is not found in the new meta chain, thus the instruction *remove(A)* is generated. After all mNodes in the active metachain are compared, a suspension instruction *suspendAll()* is added. Then the comparison is repeated but reversed: Each mNode in the new metachain is compared to the active metachain, and any unmatched mNode in the new metachain will be marked for addition. Subsequently, the instruction *insert(D, 2)* is generated, where 2 is the insert index position in the current service chain. Last, the list of the instructions is passed to the service controller, in which the actual service chain reconfiguration operations are carried out.

5.3.1 Service Deletion

The deletion-addition process reconfigures the current service composition to match the description of the new metachain. The service deletion process checks each mobilet in the current service composition against the new

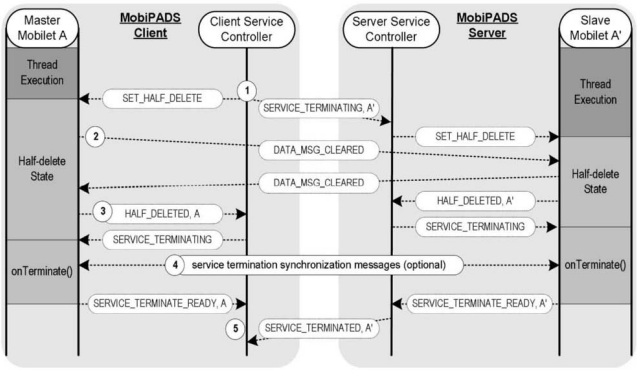

Fig. 9. Service deletion synchronization process.

metachain. Any mobilet that does not exist in the new metachain is removed. To remove a pair of mobilets, a *service deletion synchronization process* is introduced to guarantee a smooth message flow, which is shown in Fig. 9. As there may be unprocessed messages in the message queues of a mobilet pair that is to be deleted, as well as messages in transit that are generated by this mobilet pair, such messages must be handled before the mobilet pair is destroyed; otherwise, data may be lost.

To start the service deletion synchronization process, the client service controller commands the master mobilet A (referred to as mobilet A in the following discussion) to enter the *half-delete* state by sending the message SET_HALF_ DELETE. At the same time, it also informs the server service controller to delete the slave mobilet A′ (referred as mobilet A′ in the following discussion), which in turn takes the same steps as mobilet A.

After finishing the *onTerminate()* method call, mobilet A returns a SERVICE_TERMINATE _READY message to the client service controller, which can then destroy mobilet A. Similarly, mobilet A′ reports to the server service controller, which in turn informs the client service controller about the deletion of mobilet A′. At this point, the deletion of the mobilet pair A and A′ is complete.

As the service deletion synchronization process ensures data integrity and no interfere with the operations of other mobilet pairs, deletions of multiple mobilet pairs can be carried out simultaneously. This means that the half-delete state and the *onTerminate()* method call of different mobilet pairs can overlap with each other to operate concurrently.

5.3.2 Service Suspension

After all unwanted mobilets have been removed in the service deletion synchronization process, the service chain will undergo a service suspension. The service suspension takes place only after the service deletion. The reason for this approach is, as follows: Before a pair of mobilets is actually deleted, intermediate messages generated by either mobilet must be processed by the corresponding mobilet at the other side. Suspension of the service chain will freeze the message flow, including these intermediate messages. Deleting the mobilet pair would then render these intermediate messages meaningless and cause data loss. Therefore, during service deletion, we must allow continuous execution of the service chain and let the targeted mobilet pair finish processing all the intermediate messages.

After completing the service deletion synchronization process, the service composition is suspended for service addition. The need for suspension arises because a new pair of mobilets has to be added to the service composition simultaneously. Otherwise, without the corresponding mobilet acting at the receiving end, the messages generated by the new mobilet will be meaningless. One thing worth noting is that, during suspension, a mobilet is still free to receive messages using its message queue, thus avoiding the loss of messages and the blocking of other components.

5.3.3 Service Addition

The service addition takes place first at the MobiPADS client side. The new metachain will be used to check against the current service chain. Any newly required mobilets will be initialized and then inserted into the current service chain. After completing the client side reconfiguration, the client service controller instructs the server service controller to carry out the corresponding service addition of the peer mobilets. Once the server service controller completes the service addition, it will resume the service chain operation and inform the client service controller to also resume its service chain operation. The execution threads of all the mobilets will then be resumed and service provision will recommence.

5.4 Mobilet Reconfiguration

Simply adding or removing mobilets within the service chain is not adequate to adapt to the changes in a wireless environment. To allow a finer-grained adaptation, the MobiPADS programming platform should allow reconfiguration at the level of individual mobilets. To develop a reconfigurable mobilet, the service object can leverage and dynamically extend the event system and environment monitor. In particular, the mobilet can subscribe to an event and allow it to react to the event messages by adjusting its internal parameters to best adapt to the changes in the environment. If necessary, the mobilet may change the subscription of EnvMonitor to adapt to the changing requirements of context monitoring. An example is an image transcoding mobilet that provides different levels for the compression ratio. It can use a different compression ratio, based on the reported bandwidth from an event source that monitors the bandwidth. However, it is not desirable to have the mobilet adapt to the contextual changes implicitly. Rather, a set of operation modes should be defined, which will allow external entities to override the adaptation logic of the mobilet and enforce a specific mode of operation.

6 Application Adaptation

While the adaptation policies within MobiPADS offers some degree of context adaptation, at times mobile applications are still in the best position to make critical decisions on the operating context and, hence, the adaptation strategy. For this reason, the MobiPADS provides the mobile application with an extensive set of APIs and reflective interfaces. Through the metalevel object representation of the internal event system and service reconfiguration mechanism, a mobile application can access the

TABLE 2
Relationships between Adaptation Initiators and Reconfiguration Entities

	Reconfiguration Entity		
Adaptation Initiator	Mobile Application	Service Chain	Mobilet
Mobile Application	✓ To adapt to the dynamic wireless environment, a mobile application can respond to the contextual event and adjust its behavior accordingly.	✓ Using the application profile, a mobile application can specify the service configurations under different environments. The mobile application can also change the current configuration directly.	✓ By supplying suitable parameters to a mobilet, a mobile application can fine-tune the subtle behavior of individual mobilets, so that the most suitable mode of operation is selected.
MobiPADS System	The MobiPADS system cannot reconfigure the mobile application directly. It can only supply the mobile application with the context.	✓ According to the MobiPADS system profile, the configuration of the current service chain is actively adjusted to suit the existing environment.	✓ If specified in the system profile, the reconfiguration process can also switch the mode of operation of an individual mobilet by supplying suitable parameters.
Mobilet	A mobilet cannot affect the mobile application directly.	A mobilet cannot affect the whole service chain directly. However, it can disable itself, and perform no action in the service chain.	✓ A mobilet can subscribe to a contextual event and react to environmental changes by switching its mode of operation.

contextual information, service configuration and adaptation strategy of the MobiPADS, and modify these entities to obtain optimal service provision from the MobiPADS.

To have a clearer picture of the capabilities and roles of each entity in the MobiPADS platform in supportting adaptation, Table 2 shows the relationships between adaptation initiators and reconfiguration entities. There are three entities that can subscribe and react to contextual events: the *mobile application*, the *MobiPADS system* itself, and the *mobilet* running within the MobiPADS. On the other hand, there are three entities that can be reconfigured to adapt to the changes in the context environment: the mobile application, the service chain of MobiPADS and the mobilet within the service chain.

A mobile application can respond to the contextual changes by changing its internal logic, or by changing the configuration of the service chain or even the behavior of individual mobilets within the service chain. By contrast, the MobiPADS cannot alter the internal logic of the mobile application, but can only supply the mobile application with the contextual information. Within a service entity, the mobilet can only change its own operation logic to adapt to the changes, while not affecting the other mobilets or the overall service chain.

6.1 Reflective Adaptation

To support the development of context-aware mobile applications, the MobiPADS exposes three metalevel objects that abstract the contextual information and the service adaptation of the system, which are shown in Table 3. Through these metaobjects, the mobile application can subscribe to the contextual changes and is highly flexible in selecting and adjusting the adaptation policy of the MobiPADS. The three metaobjects are the *Context* metaobject, the *Configuration* metaobject, and the *Adaptation* metaobject. The *Context* metaobject allows examination and modification of the context composition and allows a mobile application to initiate subscription to a contextual event. A mobile application can also create new context composition through the *Context* metaobject. The *Configuration* metaobject reflects the configuration of the current service chain that serves the mobile application. Through the *Configuration* metaobject, a mobile application can add or remove mobilets in the service chain, as well as change the operation mode of individual mobilets. The *Adaptation* metaobject reflects the adaptation policies of the MobiPADS. Through the *Adaptation* metaobject, a mobile application can modify the service composition of metachains and also the context composition of environment monitors. Table 3 also shows the interface of the *ContextNode*, which represents the nodes that assemble the context composition tree, and the interface *ContextListener* that is used for subscribing and receiving notifications of contextual events.

6.2 A Case Example

To have a clearer understanding of the metaobjects, we give an example in Fig. 10 to show how a mobile Web application can 1) adjust the adaptation policy of MobiPADS and 2) adjust its internal logic and the service chain of MobiPADS, in response to environmental changes. Using the MobiPADS API, the application may add a new metachain to MobiPADS, which specifies that when the system experiences limit in *network bandwidth* but with sufficient *battery power*, a service chain with *image transcoding* and *compression* services shall be configured. Using transcoding and compression, the data volume of all images and text passing through the service chain are reduced to achieve a faster transfer time over the wireless link, which, in turn, reduces the delay experienced by the user. The application also subscribes to an event on network disconnection. The application receive notification when the network disconnects, upon which the application will insert an *asynchronous data* service to the current service chain, such that the MobiPADS system will be able to serve continuously, even the network connection of the mobile device is lost.

Fig. 11 shows the sample code of the mobile Web application. On lines 2-16, the *setAdaptation()* method shows the way to add a new metachain to the MobiPADS. Lines 3-5 created three *ContextNodes*. Line 6 instructs *node2* to send a notification when the maximum network transfer rate is lower than 10,000 bits per second. Line 7 instructs *node3* to report when the battery power level is higher than 50 percent. Lines 8-10 add *node2* and *node3* to *node1* as the child node, and set the relationship of these two nodes as conjunction (AND condition). Lines 11-12 add this newly created context tree to the MobiPADS, and name it "User_lowNetwork HighPower." Last, on lines 13-15, the "User_lowNetwork

TABLE 3
Reflective Mobipads API for Context-Aware Mobile Applications

Interfaces	Description
interface Context {	*The meta-object that reflects the Context Notification System.*
String[] **listAllContext**();	Lists all contextual events available in the platform.
String[] **listAllActiveContext**();	Lists all contextual events that have been subscribed.
ContextNode **getContextComposition**(String ctxName);	Gets the composition of a specific contextual event.
void **setContextComposition**(String ctxName, ContextNode node);	Updates the composition of a specific contextual event.
void **newContextComposition**(String ctxName, ContextNode node);	Creates a new contextual event with a new composition.
void **subscribeContext**(ContextListener listener, String ctxName, String relation, String refValue);	Subscribes to a specific contextual event; must provide the listener with the context and specify the condition when the listener should be notified.
void **unsubscribeContext**(ContextListener listener);	Removes a listener from the subscription list.
}	
interface ContextNode {	*// A node object in the composition tree of a composite event.*
boolean **isLeave**();	Checks if the current node is a leave node.
String **getContextName**();	Gets the contextual event that this node monitors (for leave nodes only).
String **getRelation**();	Gets the relation operator for the contextual event (for leave nodes only).
String **getReferenceValue**();	Gets the relation operand for the contextual event (for leave nodes only).
void **setContext**(String ctxName, String relation, String refValue);	Updates the monitoring condition for this node (for leave nodes only).
ContextNode[] **getChildNode**();	Gets the child nodes for the current node (for non-leave nodes only).
void **addChildNode**(ContextNode node);	Adds a child node to the current node (for non-leave nodes only).
void **removeChildNode**(int position);	Removes a child node from the current node (for non-leave nodes only).
boolean **isChildConjunction**();	Checks the correlation among child nodes (true: AND; false: OR).
void **setChildConjunction**(boolean isConjunction);	Changes the correlation among child nodes (true: AND; false: OR).
}	
interface ContextListener {	*// To subscribe to a contextual event, an object has to implement this interface.*
void **notifyContext**(String time, String detail);	When the contextual condition is fulfilled, this method will be invoked.
}	
interface Configuration {	*// The meta-object that reflects the current service chain of the MobiPADS.*
String[] **listAvailableService**();	Lists all available services that can be deployed on the MobiPADS.
String[] **listServiceMode**(String serviceName);	Lists the descriptions of the operation modes of a service.
String[] **listServiceChain**();	Lisst the service names of the mobilets in the current service chain.
void **insertService**(int position, String serviceName);	Inserts a mobilet into the current service chain.
void **removeService**(String serviceName);	Removes a mobilet from the current service chain.
int **getServiceMode**(String serviceName);	Gets the operation mode of a mobilet in the current service chain.
void **setServiceMode**(String serviceName, int mode);	Updates the operation mode of a mobilet in the current service chain.
}	
interface Adaptation {	*// The meta-object that reflects the reconfigurable meta-chains.*
String[] **listAvailableService**();	Lists all available services that can be deployed on the MobiPADS.
String[] **listServiceMode**(String serviceName);	Lists the descriptions of the operation modes of a service.
int **getMetaChainCount**();	Shows the number of meta-chains associated with the current application.
String **getEnvMonitor**(int index);	Gets the environment monitor (a composite event) of a meta-chain.
String[] **getMetaChain**(int index);	Gets the list of services of a meta-chain.
int[] **getServiceMode**(int index);	Gets the list of operation modes of the services of a meta-chain.
void **addMetaChain**(String ctxName, String[] metaChain, int[] mode);	Creates a new meta-chain (an environment monitor with a service list).
void **insertService**(int metaChainIndex, int position, String serviceName, int serviceMode);	Inserts a service into a meta-chain.
void **removeService**(int metaChainIndex, String serviceName);	Removes a service from a meta-chain.
void **updateServiceMode**(int metaChainIndex, int position, String serviceName, int serviceMode);	Changes the operation mode of a specific service in a specific meta-chain.
}	

HighPower" Contextual event is used as an event monitor for a new metachain, which consists of "HTP_Image Transcoding" and "HTTP_Compression" mobilet pairs. As such, the MobiPADS will be able to reconfigure the service chain to provide the "HTTP_Image Transcoding" and the "HTTP_Compression" service when the network is low in bandwidth and the battery level is high.

On lines 17-26, the *MyListener* class shows the way to subscribe to a contextual event and react to the contextual change by modifying the current service chain. To receive the contextual event of MobiPADS, the mobile application has to provide an object that implements the *ContextListener* interface and use it to subscribe to a specific contextual event. Line 20 shows the subscription of the contextual

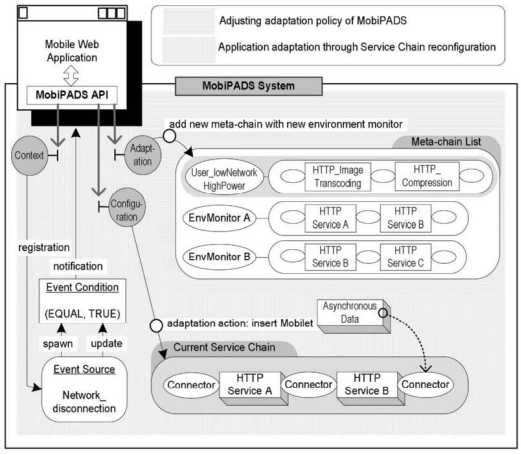

Fig. 10. A case example for context-aware mobile application.

```
1 public class SampleApp {
2 public void setAdaptation() {
3   ContextNode node1 = new ContextNodeImp();
4   ContextNode node2 = new ContextNodeImp();
5   ContextNode node3 = new ContextNodeImp();
6   node2.setContext("Network_maxRate", "LESS_THAN", "10000");
7   node3.setContext("Power_batteryLevel", "GREATER_THAN", "50.0");
8   node1.setChildConjunction(true);
9   node1.addChildNode(node2);
10  node1.addChildNode(node3);
11  Context ctx = MobiPADS.getContext();
12  ctx.newContextComposition(node1, "User_lowNetworkHighPower");
13  Adaptation apt = MobiPADS.getAdaptation();
14  String[] mChain = {"HTTP_ImageTranscoding", "HTTP_Compression"};
15  apt.addMetaChain("User_lowNetworkHighPower", mChain);
16 }
17 class MyListener implements ContextListener {
18   public MyListener () {
19     Context ctx = MobiPADS.getContext();
20     ctx.subscribeContext(this,"Network_disconnect","EQUAL","TRUE");
21   }
22   public void notifyContext(String time, String detail) {
23     Configuration cfg = MobiPADS.getConfiguration();
24     cfg.insertService(-1, "AsynchronousData");
25   }
26 }
27 }
```

Fig. 11. Sample code context-aware mobile application.

event "Network_disconnect." Lines 22-25 show the *notifyContext()* method, which will be called back when the event occurs. Line 24 shows that the mobile application adapts to the change by inserting an "AsynchronousData" service into the current service chain; Index -1 means at the last position of the service chain.

7 Experimental Methodology and Test Bed

Dynamic reconfiguration aims to maximize the performance of Web access under a vigorously changing wireless environment. However, the reconfiguration process of service composition brings a certain number of performance penalties that are unavoidable. It is important to justify the use of reconfiguration so that the operations overhead is comparatively small compared with the performance gain in the reconfigured service composition. In this section, our focus is on the reconfiguration time and the suspension time. Reconfiguration time indicates the level of responsiveness to the environment. In particular, reconfiguration time is the time taken for the MobiPADS system to adapt to changes in the wireless environment. Suspension time is the duration in which all execution threads of the mobilets are paused, and the service provision of the service composition is temporarily suspended. In other words, suspension time is the amount of time in which a user will find the MobiPADS system inactive due to reconfiguration.

7.1 Reconfiguration and Suspension Time

In this experiment, three PC machines were used to act as a MobiPADS server, a mobile node and a router. The router is a Linux machine that runs NIST net [14], which emulates numerous complex performance scenarios, including: tunable packet delay distributions, congestion and background loss, bandwidth limitation, packet reordering, and packet duplication. The two other PCs are connected through this router to emulate the dynamics of a wireless environment.

In this experiment, we had the MobiPADS server already running. At the Linux router, the round trip time (RTT) between the MobiPADS client and the MobiPADS server was set to 100 ms. The MobiPADS client was then started, which set up a service composition by coordinating with the MobiPADS server. Then the MobiPADS client initiated the reconfiguration process.

The initialization time (T) is the summation of the following components:

- α: establishing a TCP socket between the MobiPADS client and server.
- 2 RTT: delay incurred in the exchange of messages during initialization.
- $\frac{\beta}{B}$: transmission time of the metachain, where β is the data size and B is the bandwidth.
- $\frac{\gamma}{B}$: transmission time of the mobilet code request, where γ is the data size of the request.
- $\frac{\delta}{B}$: transmission time of the mobilet codes, where δ is the data size of the codes.
- C: all computational overheads introduced by MobiPADS, such as the execution environment, the system services, and internal object messaging, excluding the time used for the transfer of data over the network.

The initialization time can then be represented as:

$$T = \alpha + \frac{\beta + \gamma + \delta}{B} + 2\mathrm{RTT} + C. \tag{1}$$

In this experiment, as the TCP socket establishment uses three way handshaking [15], which take three control messages to establish the connection between the sender and receiver. Without considering other overheads, each control message takes half RTT, thus we assume that α, the TCP connection establishment time $\cong 1.5$ RTT, so that:

$$T = \frac{\beta + \gamma + \delta}{B} + 3.5\ \mathrm{RTT} + C. \tag{2}$$

Reconfiguration involves the following factors:

- $\frac{\beta+\gamma+\delta}{B} + 3.5\ \mathrm{RTT} + C$: active deployment phase of (2).
- $\mathrm{Max}(\mathrm{m}_1, \mathrm{m}_2, \ldots, \mathrm{m}_\mathrm{i}, \ldots, \mathrm{m}_\mathrm{j}) + \mathrm{RTT}$: deletion phase, as shown in Fig. 9, where mi is the deletion synchronization time of mobilet i, and j is the number of mobilets to be deleted. This factor is based on $\max(\mathrm{m}_{1,2,\ldots,\mathrm{i},\ldots,\mathrm{j}})$ and not $\Sigma \mathrm{m}_\mathrm{i}$ due to the fact that the service deletions of all unwanted mobilets are carried out in parallel, thus only the longest service deletion time will be counted. The RTT component represents service termination messaging between the client and server service controllers.
- $2\sum_{i=1}^{k} n_i + \mathrm{RTT}$: addition phase, as described in Section 5.3.3, where n is the mobilet initialization time and k is the number of mobilet pairs to be added. The coefficient of two is needed since the mobilet addition is carried out sequentially at the MobiPADS client and server. The RTT is for service addition messaging between the MobiPADS client and server.

Thus, the reconfiguration time (T_2) can be represented as:

$$T_2 = \frac{\beta + \gamma + \delta}{B} + 2\sum_{i=1}^{k} n_i + \max(\mathrm{m}_{1,2,\ldots,\mathrm{i},\ldots,\mathrm{j}}) + 5.5\ \mathrm{RTT} + \mathrm{C}. \tag{3}$$

In this experiment, since the tested mobilets are null mobilets (a null mobilet refers to a normal mobilet that inherits the characteristics of a mobilet class, but contains no logic processing routine), the deletion time of every mobilet is almost the same. A similar effect is seen for the addition time of every mobilet. Therefore, (3) can be rewritten as:

$$T_2 = \frac{\beta + \gamma + \delta}{B} + 2kn + \mathrm{m} + 5.5\ \mathrm{RTT} + \mathrm{C}. \tag{4}$$

The service composition maintains the service provision throughout the whole dynamic reconfiguration process, except for the addition phase. Therefore, the suspension time (S) is the time of the addition phase:

$$S = 2kn + \mathrm{RTT}. \tag{5}$$

7.2 Measurement Analysis

We measured reconfiguration time under different bandwidths for different mobilet sizes and different numbers of mobilets to be deleted and added. In this experiment, we

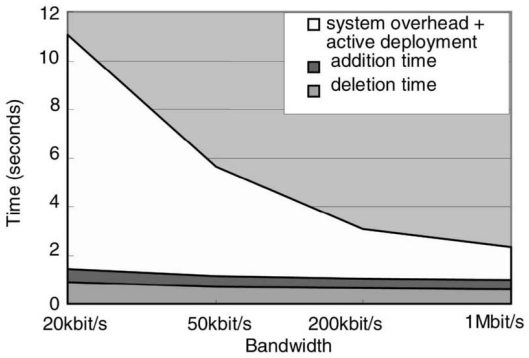

Fig. 12. Cumulative time for reconfiguration with active deployment [mobilets of 5 KB x (5 deletions + 5 additions)].

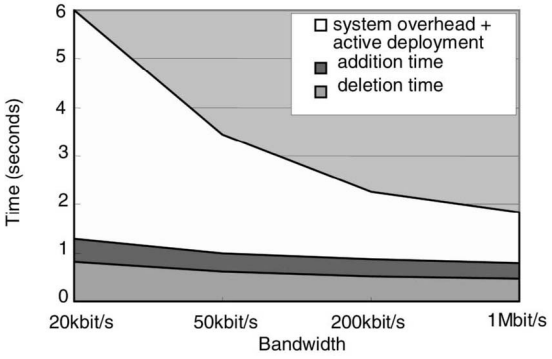

Fig. 14. Cumulative time for reconfiguration with active deployment [mobilets of 5 KB x (2 deletions + 2 additions)].

tested the system under the bandwidth of 20 kbps, 50 kbps, 200 kbps, and 1 Mbps. Mobilets with sizes of 5 KB were used. In the experiment, the deletion time, addition time and total reconfiguration time were measured. Based on these three data items, we deduced the system overhead and active deployment time. This experiment has four parts, and the results are plotted in Figs. 12, 13, 14, and 15 corresponding to the following reconfiguration arrangements:

1. Five mobilet deletions and five mobilet additions and the newly added mobilets needing active deployment.
2. Same as 1, but without active deployment.
3. Two mobilet deletions and two mobilet additions, and the newly added mobilets needing active deployment.
4. Same as 3, but without active deployment.

Comparing Fig. 12 with Fig. 13, it was found that, when active deployment is needed, the reconfiguration time is dominated by the active deployment process. Figs. 14 and 15 show similar behaviors. When active deployment is needed, the active deployment process and other system overheads contribute 56.3 to 87 percent of the overall reconfiguration time. Figs. 13 and 15 show that, without active deployment, the reconfiguration time is relatively insensitive to variations in bandwidth. Such a characteristic is due to the fact that, during the reconfiguration process, only a few control messages are exchanged over the network, and these messages are small in size. Without active deployment, the system overhead is relatively constant between 21.4 and 28.5 percent of the overall reconfiguration time. The suspension time, which is represented by the service addition time, is between 0.528 to 0.316 seconds. The short suspension time is hardly noticeable by users.

8 Related Work

In the past, the development of middleware platforms has taken the approach of a black box concept, where the detailed operations of invocations, dispatching and instantiation of processes at the remote end are completely hidden away from the programmers developing the applications. For example, in platforms such as CORBA and DCOM, the middleware abstracts the distributed applications as a unified collection of objects operating over a virtual bus. The use of language-independent IDL (Interface Definition Language) to specify the object's interfaces and boundaries enables the separation of the object implementations from the services provided by the middleware. Importantly, the application objects are completely oblivious to the detailed operations of the middleware, and the architecture does not present any option for the application to inspect or alter the middleware services. In a mobile computing environment, it is desirable for middleware systems to support some degree of configurability to enable mobile applications to adaptively reconfigure and optimize the services in response to varying context environments.

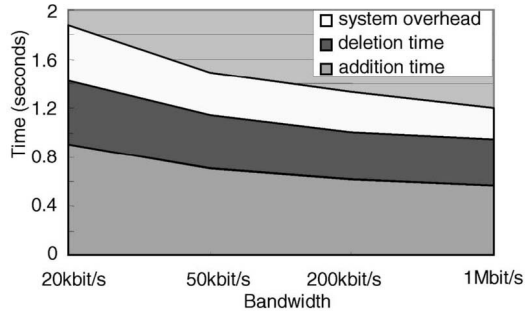

Fig. 13. Cumulative time for reconfiguration without active deployment [mobilets of 5 KB x (5 deletions + 5 additions)].

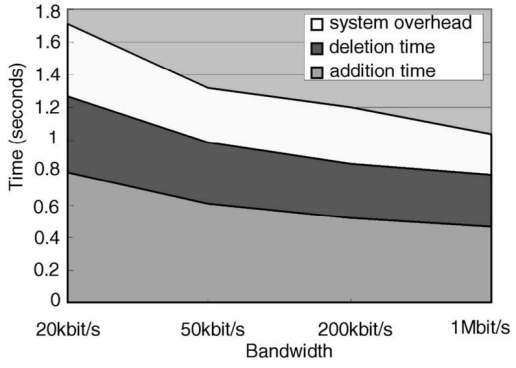

Fig. 15. Cumulative time for reconfiguration without active deployment [mobilets of 5 KB x (2 deletions + 2 additions)].

The need to create a more robust and configurable middleware system is realized in the development of the Open ORB [16]. The design and implementation of Open ORB is based on a reflective middleware platform. Access to the underlying platform is achieved through the metainterface, which exposes the metaspace that represents the support environment for the component. The Open ORB architecture is designed as a general middleware that supports system reconfigurations by allowing applications to inspect (through reflection) and to adapt (through reification) the behavior of the underlying system components. The architecture does not have specific mechanisms to support mobile computing applications so that context awareness can be seamlessly integrated into the programming model to facilitate dynamic configuration and deployment of mobile services. The Open ORB component composition employs the component graph to represent the complex relationship between the bindings of the components. While it is a generic and flexible model, the model requires careful handling and management of the component placements, dependencies and underlying structure. If unconstrained, it may lead to binding inconsistency and potential operation deadlocks. By contrast, MobiPADS uses the concept of service chaining as the underlying technique to support flexible composition of the mobile services offered by the middleware by dividing the system services into a service chain of mobilets. The organization of the system services into mobilets provides a clean separation between service implementations and specifications. The communication between mobilet services is based on the notion of object encapsulation, where data pertaining to a service is encapsulated in a header object, while the encapsulation protocol remains open-ended to allow arbitrary data to be sent to the peer mobilet service. The peer-to-peer mobilets are therefore responsible for implementing their own service protocols to define the format of the encapsulated object data. The separation of peer-to-peer protocols between services and open-ended interfaces between mobilets minimizes the dynamic dependencies, while promoting ease of maintenance and reconfiguration of mobilets.

In addition to dynamically configurable middleware, there are systems that have been developed to directly address issues pertaining to mobile computing for a specific environment. For example, the Mowgli middleware [17] designed specifically to operate over the WWW, separates the communication path into two communication subsystems: One is wired and the other is wireless. Based on the Mowgli architecture, the Mowgli WWW is a set of Web services that enhances the Web browsing experience in a mobile WAN environment. The Mowgli WWW Agent and Proxy is a pair of mediators that serves HTTP. The agent and proxy communicate with each other using Mowgli HTTP (MHTTP), which is optimized for the wireless environment. The Mowgli WWW predictive uploads the embedded object to the mobile client, providing compresses data, intelligently filters the large embedded object, and manages the cache. The WebExpress [18] is another split connection approach that aims at improving the Web browsing experience in a wireless environment. It features a pair of agents at the ends of the wireless link. Unlike Mowgli WWW, WebExpress uses normal TCP/IP protocols on the wireless network, and its enhancements focus on improving HTTP across the wireless link. The optimization techniques of WebExpress include caching, differencing of CGI requests and HTTP protocol optimization.

Middleware to address file systems operating over a wireless environment was first introduced in the Coda system [19]. In particular, Coda middleware was designed to deal with the problems of file accesses that arise out of disconnected operations. The work shows how Coda exploits the caching of data to apply it to improving data availability operating in a hostile or even disconnected wireless environment. In the event of disconnection, the middleware serves file system requests based on the contents of its cache. Upon reestablishment of the connection, the middleware is responsible for performing data synchronizations and for propagating modifications to the servers.

In the Rover toolkit [20], the middleware supports the development of both mobile-transparent and mobile-aware applications. The key idea of Rover is the introduction of the relocatable object (RDO) and the queued remote procedure call (QRPC). Object codes extended from RDO can be easily relocated from the server to mobile client (or vice-versa) to allow disconnected operations. On the other hand, QRPC permits applications to perform remote procedural calls even when connection is unavailable, by queueing the calls locally and serves as the connection is reestablished.

As mobile computing continues to gain popularity, there is an increasing need to develop and extend middleware to support adaptations brought about by the dynamic characteristics of the wireless environment. Previous research has mainly focused on developing middleware systems to address specific issues and operating environments. These middleware have functionalities that are static, so that they require a priori installation of services to the underlying systems. Once deployed, it is difficult for the middleware to evolve to match the varying characteristics of mobile applications. In fact, most of these middleware do not allow applications to actively interact with the services and to adapt these services to the evolving needs of applications. The concept of configurable and active service deployment applied in the MobiPADS offers a new paradigm for the next generation of context-aware mobile middleware.

9 Conclusions

The growing importance of mobile computing has given rise to a need to revisit the design requirements of future middleware to cope with the diverse challenges of operating over a dynamic context. The fundamental assumption of a static operating environment, which resulted in a monolithic "black-box" approach to implementing existing middleware, is invalidated in a mobile computing environment. An important requirement in the formulation of a context-aware middleware is the need to devise suitable control mechanisms that allow applications to directly participate in resource adaptation in response to the dynamic operating environment. In this paper, we have presented the overall architectural design of the MobiPADS system. The MobiPADS represents a reflective-based mobile middleware that is designed to support the active deployment of augmented services for mobile computing.

The underlying MobiPADS is implemented as a collection of active-service entities, known as a mobilets, which are constructed as a series of primitive services that form a service-chain composition. The reflective model provides metainterfaces for applications to directly participate in computation adaptation in response to the changing context. Through the metalevel object representation of the internal event system and service reconfiguration mechanism, a mobile application can access contextual information, the service configuration and adaptation strategy of MobiPADS, and examine and modify these entities to obtain optimal service provision from the MobiPADS.

The entire MobiPADS system is implemented and deployed in a Java platform. To demonstrate the functionality of the middleware and verify its interactions between components, we have developed and implemented an adaptive Web access application [7] running over the MobiPADS execution environment. Importantly, the complete implementation of the MobiPADS and the application framework provides us with a unique opportunity to truly exercise the system, which provided us with important insights on the complex interactions between the different software components and objects. The design choice of modeling the platform with a crisp boundary of classes to promote clear distribution of responsibilities and code reusability needs to be balanced against the impact on computation overheads incurred in instantiating too many objects. Further experiments are planned to evaluate the performance of the system in terms of the computational overheads incur in executing MobiPADS platform on both the server and client proxies. This would provide us with insights on the performance issues of MobiPADS in supporting concurrent threads of mobilet chains across different platforms such as PCs, PDAs, J2ME, and notebooks. As part of our future work, we intend to investigate ways to improve the performance in terms of supporting more mobilets without significantly reducing the overall processing rate. We would also like to investigate the support of more flexible compositions of the mobilets, while providing support for consistency checking, security, and access control.

ACKNOWLEDGMENTS

This project is supported by the HK Polytechnic University Central Research Grant G-V893 and RGC Competitive Grant B-Q453.

REFERENCES

[1] R. Ben-Natan, *CORBA: A Guide to Common Object Request Broker Architecture.* McGraw-Hill, Inc., 1995.
[2] T. Downing, *Java RMI: Remote Method Invocation.* Foster City, Calif.: IDG Books Worldwide, 1998.
[3] T.L. Thai, *Learning DCOM.* Sebastopol, Calif.: O'Reilly, 1999.
[4] F. Kon, F. Costa, G. Blair, and R.H. Campbell, "Adaptive Middleware: The Case for Reflective Middleware," *Comm. ACM,* June 2002.
[5] L. Capra, "Mobile Computing Middleware for Context-Aware Applications," *Proc. 24th Int'l Conf. Software Eng. (ICSE 2002),* May 2002.
[6] S.N. Chuang, A.T.S. Chan, and J. Cao, "Dynamic Service Composition for Wireless Web Access," *Proc. Int'l Conf. Parallel Processing (ICPP 2002),* pp. 429-436, Aug. 2002.
[7] S.-N. Chuang, A.T.S. Chan, J. Cao, and R. Cheung, "Dynamic Service Reconfiguration for Wireless Web Access," *Proc. 12th Int'l World Wide Web Conf. (WWW2003),* pp. 58-67, May 2003.
[8] B.C. Smith, "Reflection and Semantics in Lisp," *Proc. POPL Conf.,* pp. 23-35, 1984.
[9] N. Parlavantzas, G. Coulson, M. Clarke, and G. Blair, "Towards a Reflective Component Based Middleware Architecture," *Proc. Workshop Reflection and Metalevel Architecture,* 2000.
[10] M. Liljeberg, T. Alanko, M. Kojo, H. Laamanen, and K. Raatikainen, "Optimizing Word-Wide Web for Weakly Connected Mobile Workstations: An Indirect Approach," *Proc. Second Int'l Workshop Services in Distributed and Networked Environments,* pp. 132-139, 1995.
[11] W. Smith, D. Gunter, and D. Quesnel, "A Simple XML Producer-Consumer Protocol," Global Grid Forum Working Document GWD-Perf-8-2, July 2001.
[12] N.H. Gehani, H.V. Jagadish, and O. Shmueli, "Composite Event Specification in Active Databases: Model & Implementation," *Proc. 18th Conf. Very Large Data Bases,* pp. 327-338, Aug. 1992.
[13] S. Chakravarthy and D. Mishra, "Snoop: An Expressive Event Specification Language for Active Databases," Technical Report UF-CIS-TR-93-007, Computer and Information Sciences Dept., Univ. of Florida, 1993.
[14] "NIST Home Page,"Nat'l Inst. of Standards and Technology, http://snad.ncsl.nist.gov/itg/nistnet, 2002.
[15] "Transmission Control Protocol,"RFC 793, J.B. Postel, ed., USC/Information Sciences Inst., Sept. 1981.
[16] G.S. Blair, G. Coulson, A. Andersen, L. Blair, M. Clarke, F. Costa, H. Duran-Limon, T. Fitzpatrick, L. Johnston, R. Moreira, N. Parlavantzas, and K. Saikoski, "The Design and Implementation of Open ORB Version 2," *IEEE Distributed Systems Online J.,* vol. 2, no. 6, 2001.
[17] M. Kojo, T. Alanko, M. Liljeberg, and K. Raatikainen, "Enhanced Communication Services for Mobile TCP/IP Networking," Technical Report No. C-1995-15, Dept. of Computer Science, Univ. of Helsinki, Series of Publications C, Apr. 1995.
[18] B. Hausel and D. Lindquist, "WebExpress: A System for Optimizing Web Browsing in a Wireless Environment," *Proc. ACM Int'l Conf. Mobile Computing and Networking (MobiCom'96),* Nov. 1996.
[19] J. Kistler and M. Satyanarayanan, "Disconnected Operation in the Coda File System," *ACM Trans. Computer Systems,* vol. 6, no. 1, pp. 1-25, Feb. 1992.
[20] F. Kaashoek, "Mobile Computing with the Rover Toolkit," *IEEE Trans. Computers,* pp. 337-352, Mar. 1997.

Alvin T.S. Chan graduated from the University of New South Wales with a PhD degree in 1995 and was subsequently employed as a research scientist by the CSIRO, Australia. He is currently an assistant professor at the Hong Kong Polytechnic University. From 1997 to 1998, he was employed by the Centre for Wireless Communications at the National University of Singapore as a Program Manager. He is one of the founding members and director of a university spin-off company, Information Access Technology Limited. He is an active consultant and has been providing consultancy services to both local and overseas companies. His research interests include mobile computing, context-aware computing, and smart card applications. He is s member of the IEEE.

Siu-Nam Chuang received the MPhil degree from The Hong Kong Polytechnic University in 2002. He was then employed as a research assistant by The Hong Kong Polytechnic University for the subsequent year. He is currently a PhD student in the Department of Computing at The Hong Kong Polytechnic University. His research interests include context-aware mobile computing, adaptive computing, and mobile middleware.