Event Engine for Adaptive Mobile Computing

Shiow-yang Wu H.S. Cinatit Chao
Department of Computer Science and Information Engineering
National Dong Hwa University
Hualien, Taiwan, R.O.C.
E-mail: showyang@csie.ndhu.edu.tw

Abstract

To cope with the highly resource constrained and dynamically changing mobile computing environment, we propose an architecture which employs an active event engine to detect current resource and environment status, inform registered applications about status changes, and provide a suite of actions for application adaptation. Preliminary implementation and evaluation results demonstrate that the event engine can successfully detect the registered events and invoke application specified actions to bring the system to a desired state.

1. Introduction

The operating condition of a mobile host is constantly subject to resource availability and environment variability such as changing *mobile support stations*, connection status, as well as terrain and weather. Effective information services in such a demanding environment is undoubtedly a great challenge to information system designers [2, 3, 11]. Our goal is to develop enabling technologies for building effective information systems in mobile computing environment.

To cope with the problem of change, a mobile application must first be aware of the change before proper action can be taken. We propose an architecture for application adaptation based on an active event engine for application awareness. The availability of resources on a mobile host and the status changes in the operating environment are modeled as events. Applications can register events of interest to be notified when the events occur. The event engine keeps monitoring the resource and environment status for the detection of registered events. Once the events occur, all registered applications of the corresponding events are notified for possible adaptive actions. Applications can either choose from a suite of predefined actions provided by our framework or supply customized actions for application adaptation. Being resource and environment aware, an application can easily optimize its operating mode and execution strategy to better utilize available resources and to cope with the changing environment. The relationship between the event engine, applications and the operating system on a mobile host is depicted in Figure 1. As a summary, the main contributions of this paper are:

- An architecture for adaptive mobile computing based on active event engine.
- An event language for specifying events of interest.
- API for applications to interact with the event engine.
- Change detection and event triggering mechanism for monitoring resource and environment status.
- A suite of responsive actions for application adaptation.

The rest of the paper is organized as follows. Section 2 provides a background survey of related issues and research work. Section 3 presents the event classification and event language for the applications to specify events of interest. Section 4 describes the structure and operation of our event engine which is central to the proposed architecture. Preliminary implementation and evaluation results presented in Section 5 demonstrate the feasibility of our framework toward the construction of highly adaptive mobile information systems. Section 6 concludes the paper.

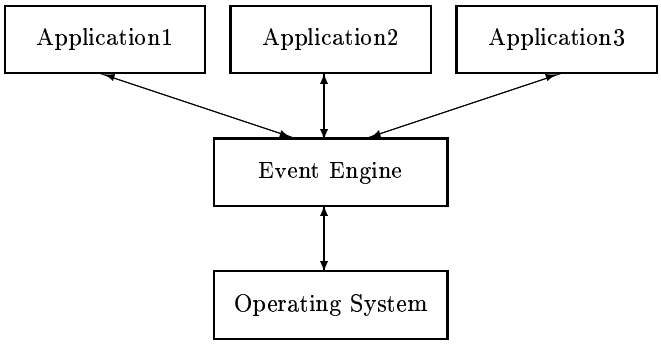

Fig. 1. The relationship between the event engine, applications and the operating system.

2. Related Work

The need for intelligent adaptation is considered essential for mobile data management [9, 16]. Similar ideas have been discussed under terms like context-aware [14], application-aware [13], environment-directed [17], and adaptive [8] information systems. Various techniques on dynamic power adaptation have been proposed for mobile computers [1, 7]. In general, the decision on when and how to adapt can be made by either the underlying system (*application-transparent adaptation*) or the application programs (*application-aware adaptation*) [12]. Application-transparent implies that no change is needed to existing applications which also means no application-specific feature can be exploited. On the other hand, the performance and flexibility offered by the application-aware adaptation may very well come with higher complexity and software development cost. The approach we propose in this paper is inspired by the work on active database systems which successfully augment traditional database systems with the capability of actively responding to changes [10, 18]. More specifically, we provide event modeling, specification, registration, and detection mechanisms, as well as proper actions for intelligent adaptation, which offers an active framework toward the construction of effective mobile information systems. Our framework is different from other event engine work such as [6, 15] in that our event modeling and triggering mechanisms have been specially tailored for mobile computing environment for which, as far as we know, no other similar system exists.

3. The Event Language

The events in an active data base system represent any change to the database itself or the way it is being used. We model changes to the resource availability and environment status as events. Events in our system can be *primitive* or *composite* [4, 5]. A primitive event is to model a certain level of change on a single source (such as disk capacity, free memory, bandwidth, etc.). Primitive events can be combined using *event operators* to form composite events. We classify the events of interest in a mobile environment into *resource events*, *mobility events*, and *environment events* as depicted in Figure 2.

Resource events model the change in resource availability on a mobile host. An application must be aware of the current resource status in order to better utilize the usually limited resources. In particular, we have identified the available disk space, free memory, battery power, and the wireless bandwidth level as the primary resources for monitoring and detection. For disk space, we monitor the free disk space left on the hard drive in MB. For memory, we measure both the available free memory space in MB and in percentage with respect to the total memory on the mobile host. Similarly for the battery power, we monitor the estimated power left in seconds remaining as well as in percentage with respect to the fully charged battery. For bandwidth, we measure the current level in Kb/second to enable the applications to be aware of the connection quality and status.

Mobility events are related to the change in position of the mobile host under consideration. This is a unique characteristic in mobile computing which has no direct counterpart in traditional distributed computing environment. We propose to monitor the current position, speed, and direction as three key parameters for representing the

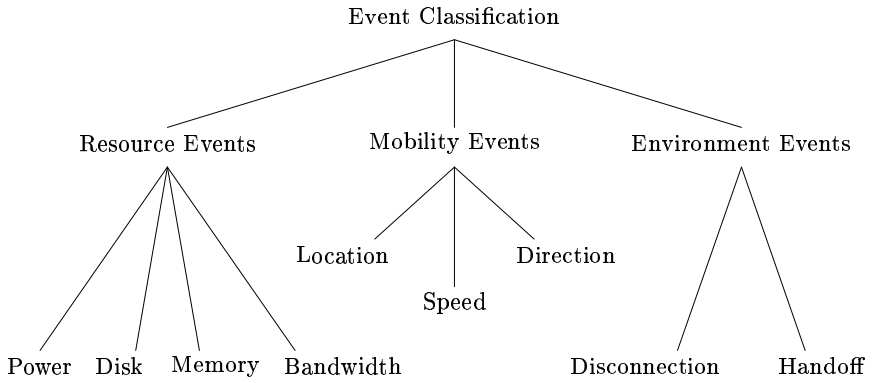

Fig. 2. Event Classification.

mobile host's current mobility status. As a first step, we use the current cell ID within which the mobile host resides as its current position. The speed is represented as km/second or as symbolic levels such as low, medium, and high. The direction is the current direction of movement of the mobile host. This is especially useful in conjunction with speed to estimate the mobility pattern of the mobile host in the near future.

Environment events are changes to the operating environment that are expected to have significant effect on the mobile host. We have identified disconnection and handoff as the two most distinctive state changes to monitor for the mobile client. This allows applications to take preventive actions in response to these changes.

Resource, mobility, and environment events are primitive events since each event of the above types is detected directly from status change of a single source. Primitive events alone are not expressive enough to express various situations of interest. We therefore provide *event operators* to combine primitive events into composite events. To simplify our system design and implementation, we provide only the AND and OR operators for combining events. For example, the composite event "E1 AND E2" occurs only when both E1 and E2 occur. Similarly, the composite event "E1 OR E2" occurs when any one of the events occurs.

Primitive events are specified using the following three formats:

ON <event_type> <op> <value> <unit>
ON <event_type> LEVEL <level>
ON <environment_event_type>

where <event_type> is the name of a resource or mobility event. <op> is a relational operator such as $>, <, =$ etc. <value> and <unit> are used together to specify the exact value threshold to be detected by the event engine. The second form allows user-defined symbolic levels to represent value ranges of interest. The third form is used to denote environment events for which only the occurrences of the events are of interest.

Some examples should make the event specification formats clear. The resource event

"ON Bandwidth < 100 kbps"

will be triggered whenever the current wireless bandwidth goes down below 100 kbps. Similarly, the mobility event

"ON Speed LEVEL high"

will be triggered if the current speed of the mobile host reaches a level defined by the user as high. Take another example, the environment event

"ON Disconnection"

will be triggered if the mobile host is disconnected from the server.

Primitive events as specified above can be further combined using the event operators to form composite events. For example, the event

ON Disk < 100 mb AND ON Handoff

will be triggered only when both the free disk space is low and the mobile host is in the handoff process.

Any application can register its events of interest to the event engine. Events specified by applications must be carefully managed to reduce redundancy and improve efficiency. For such purposes, registered events are organized into an *event forest* as depicted in Figure 3. All primitive events are at the leaf nodes. Others are either OR nodes or AND nodes. Applications registered for the same event(s) can share the same event node(s). We keep the application's ID and the associated actions with the event node(s) that the application registered. These nodes are depicted as shaded nodes in the figure. The event forest is used by the event engine for managing registered events as well as for event triggering to be discussed in the next section.

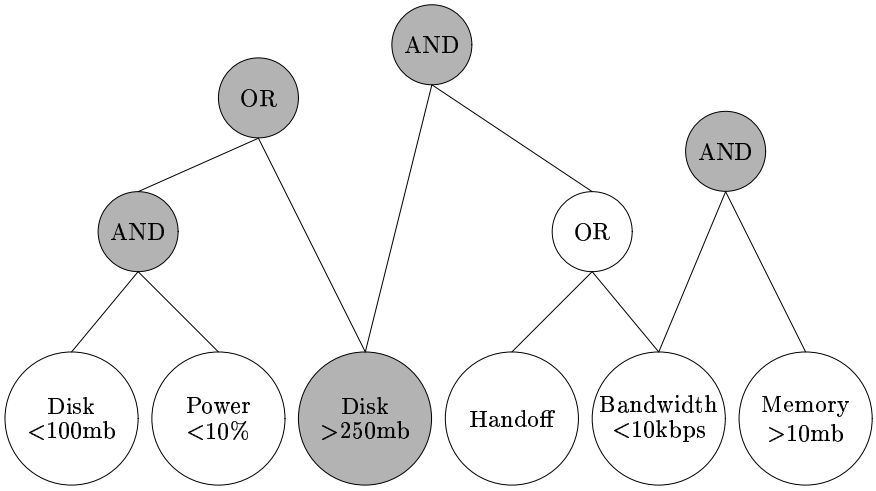

Fig. 3. The Event Forest.

4. The Event Engine

The event engine plays the central role in our adaptive mobile computing framework. It consists of an *event register*, a *change detector*, an *event detector*, and an *action activator* as depicted in Figure 4. Applications register their events of interest through the event register which then classifies the events and builds the event forest as described in the previous section. The change detector is responsible for detecting changes on the resource or environment status that are of interest to the registered applications. The event detector periodically compares the status changes with registered events to determine if any event(s) occurs. Once a registered event occurs, the action activator triggers the associated actions to respond to the changes. The process of event detection and action triggering is done by marking and traversing the event forest as follows:

- On each event detection period, all leaf nodes are checked first to determine if any primitive event occurs. A leaf node is marked if the corresponding event occurs.

- For each marked node, if there are application IDs stored with it, the action activator is invoked to inform the applications about the event occurrence and to trigger the corresponding registered actions.
- Once a marked node is processed, it is unmarked to prevent unnecessary repeated triggering. The marker is then passed upward to its parent.
- An OR node is considered marked if it receives at least one marker from any of its children. An AND node is considered marked if it has received markers from all its children.
- The process repeats until no more node need to be processed in the event forest.

We note that if an application terminates, all its records are removed from the event forest such that the subsequent event detection and action triggering do not perform wasted work on applications that are no longer exist. From the experience of our preliminary implementation to be discussed in Section 5, we found the above approach to be a simple yet efficient way of managing events and actions.

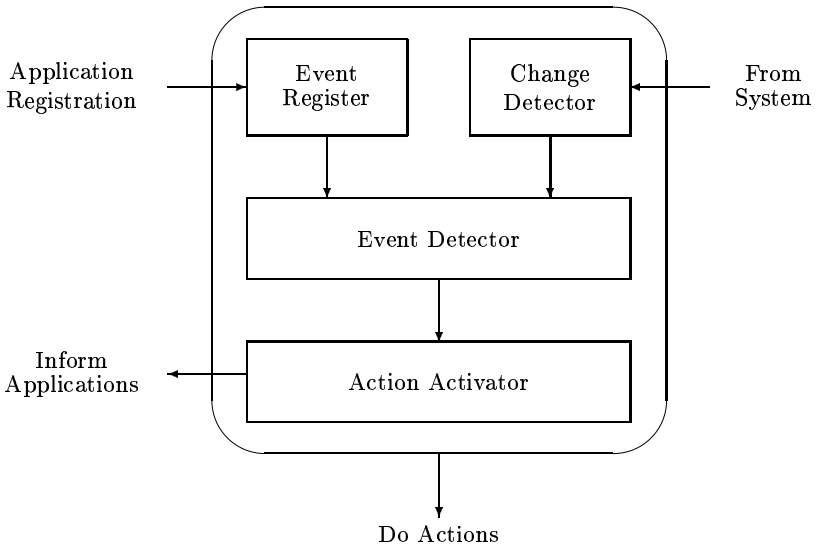

Fig. 4. The Event Engine.

For applications to specify proper actions to respond to status changes, we also provide a simple action specification format as follows.

DO <action> [<parameters>]

The <action> is the name of a built-in or user-defined action. An action can be invoked with optional parameters if required. For example, the following action instructs the event engine to trigger a power suspend action to save the energy.

DO power_suspend

And the action that follows directs the event engine to trigger a save action to protect valuable file.

DO save C:\DOC\budget.doc

5. Implementation and Evaluation

To evaluate the feasibility and performance of our approach, we have finished a preliminary implementation of the event engine that handles resource events. The event engine is implemented on a Windows98 based PC

platform using MFC and experimented on a Pentium 266 notebook. As a first step, we have developed an event engine API for the applications to register events of interest and the actions to respond. The API is in a function format that can be invoked in any C/C++ application program.

`Registration(` *APid*, *Event*, *Action* `)`

The *APid* is a unique identifier to represent the application that are registering. Both *Event* and *Action* are character strings to specify the event to be detected and the associated actions to perform once the event occurs. To simplify the implementation, we adopt a prefix notation for specifying composite events. For example, the event

ON Disk < 20mb AND ON Power < 10%

is specified as

```
"AND ( disk < 20 mb ) ( power < 10 % )"
```

The action is also specified similarly as the following example.

```
"(power_suspend)"
```

Whenever an application invokes the `Registration` API, the event register immediately records the ID, event, actions, and incorporates them into the current event forest. An application can register multiple event-action pairs by invoking the `Registration` API multiple times. Once registered, the event engine will start detecting the specified events for the application immediately.

To evaluate the performance of the event engine, we develop a simple text editor application which registers events and actions with the event engine. We then conduct a series of experiments on each resource event type to see if the event engine can successfully detect the specified events and trigger the desired actions. We are also interested in observing that, after invoking responsive actions, how fast the system can adjust itself to reach a desired state.

The first set of experiments is on the effectiveness of power adaptation. For each experiment, the editor application registers power event on a percentage of power remained on the battery and requests for the triggering of power suspension action once the power level goes below the specified percentage. We then measure how long the mobile host lasts from power on till power exhausted completely. The result depicted in Figure 5 demonstrates that power suspension is quite effective in keeping the mobile host on. For example, if the action is activated when power level goes below 40%, the mobile host can last for more than 5 hours.

The second experiment is on memory adaptation. In order to control the memory usage, the experiment follows a sequence of steps that mimics a typical operation by opening a number of applications, one of which is our editor application which registers a memory event to trigger adaptive actions when the memory usage goes above 54MB. The actions are to close the system tray first, and then clear the clipboard. The sequence of steps are as follows.

1. Start the Windows98 and the System Monitor.
2. Start up the File Manager.
3. Open the editor application.
4. Copy a number of text and image files.
5. Open the ACDSee32 graphic application.
6. When the memory usage goes above 54MB, trigger the action to close the system tray first.
7. Then clear the clipboard.

The Windows98 System Monitor is activated to display the memory usage on the screen for recording. Figure 6 demonstrates the result as displayed on the System Monitor. It is quite clear from the figure that once the physical memory usage goes above 54MB, the event engine immediately triggers the adaptive actions which quickly bring the system back to a state where the memory usage is below 50MB.

Figure 7 displays the change in resource availability along each operating steps. We can observe the significant increase in available resources after the event engine triggers the adaptive actions.

The last set of experiments is on bandwidth adaptation. In mobile environment, bandwidth is considered a scarce resource and is consistently subject to variation as well as frequent disconnection. It is therefore necessary to manage the bandwidth carefully. We use the Web browser as an example, and test the effectiveness of the

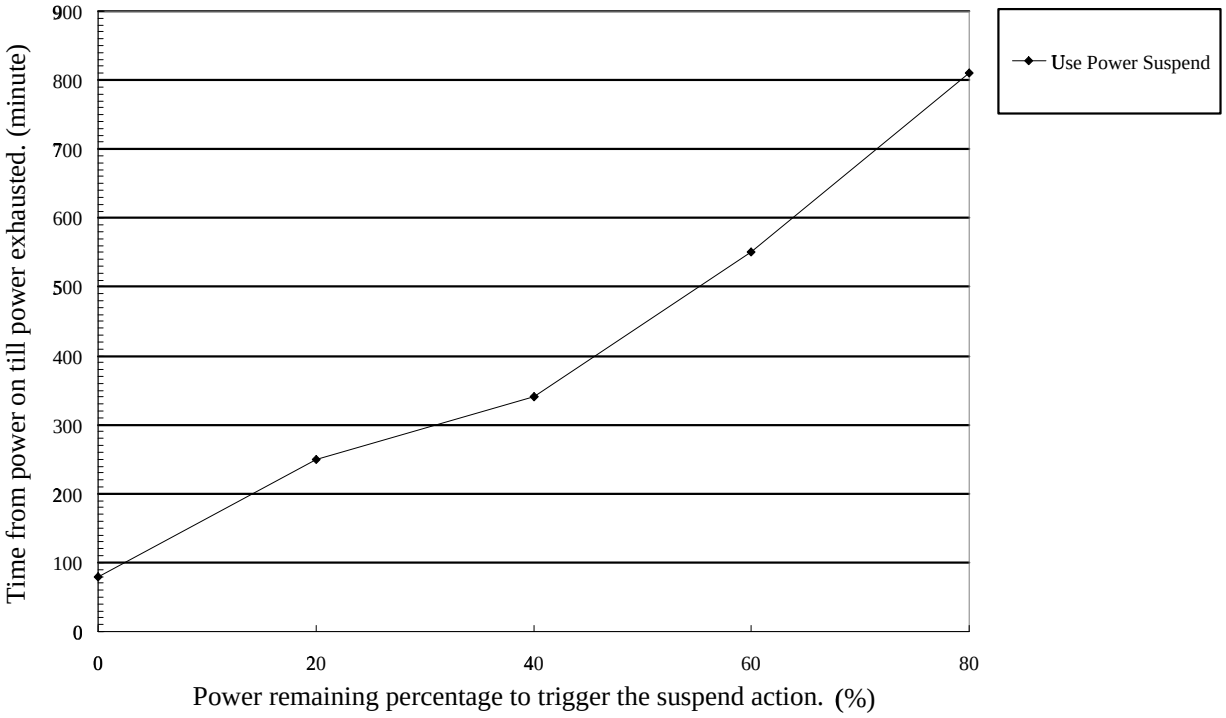

Fig. 5. Power Adaptation.

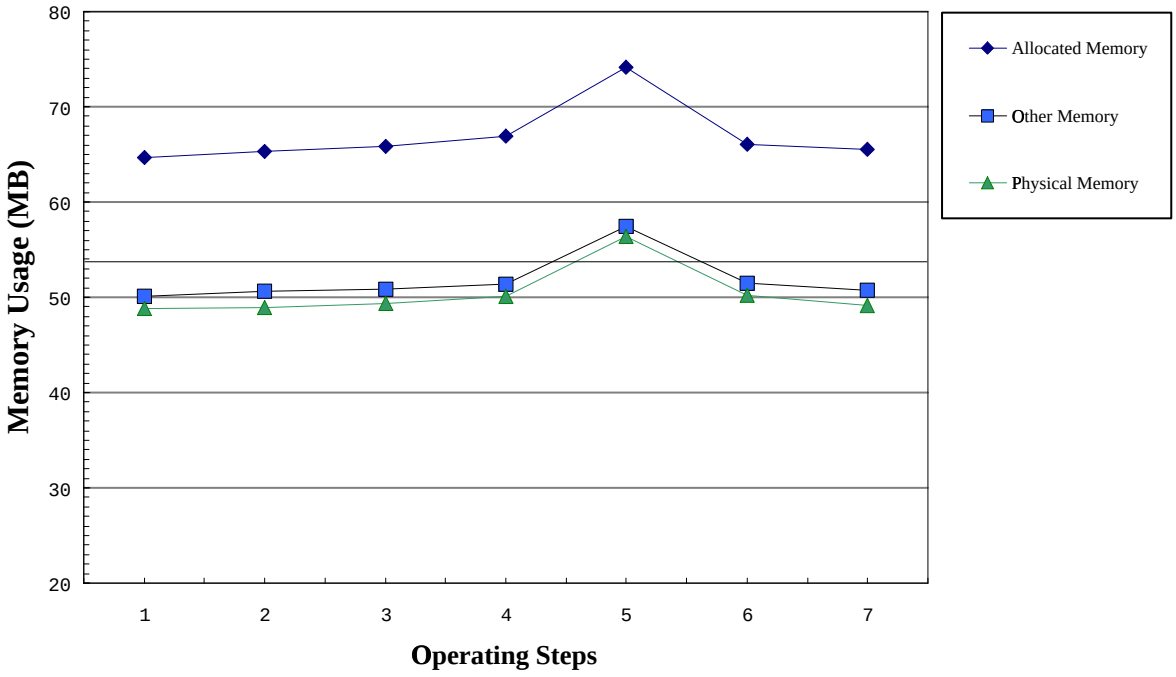

Fig. 6. Memory Adaptation.

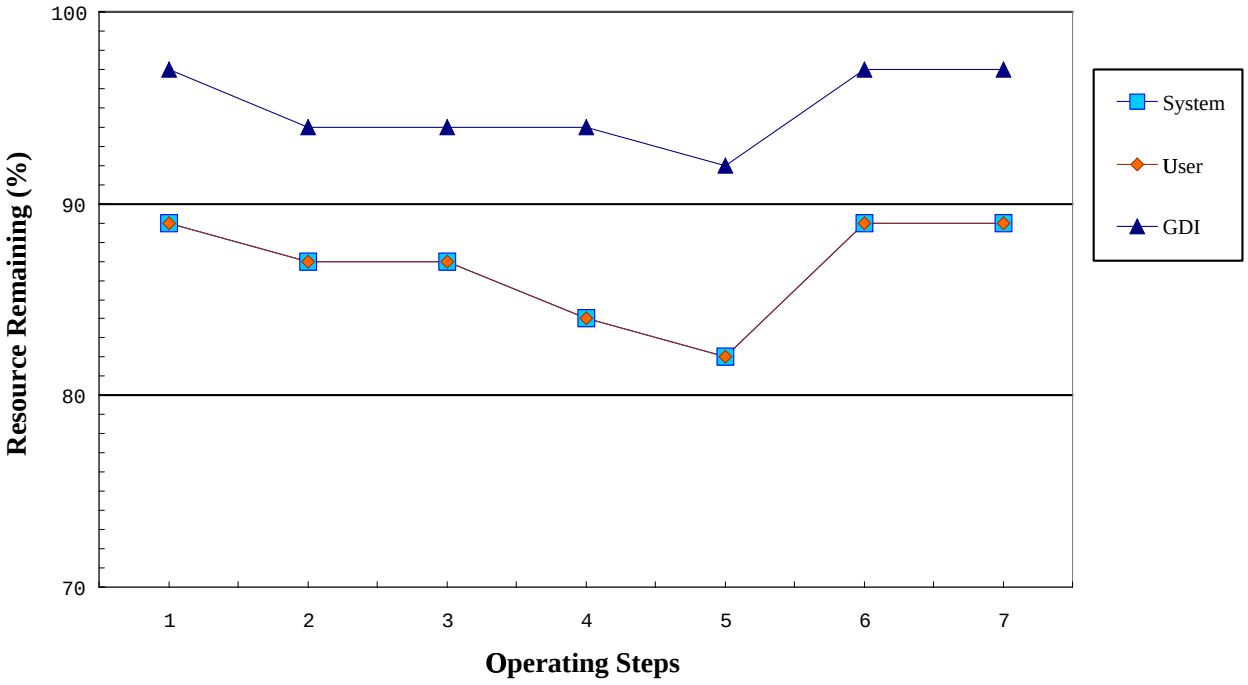

Fig. 7. Memory Adaptation.

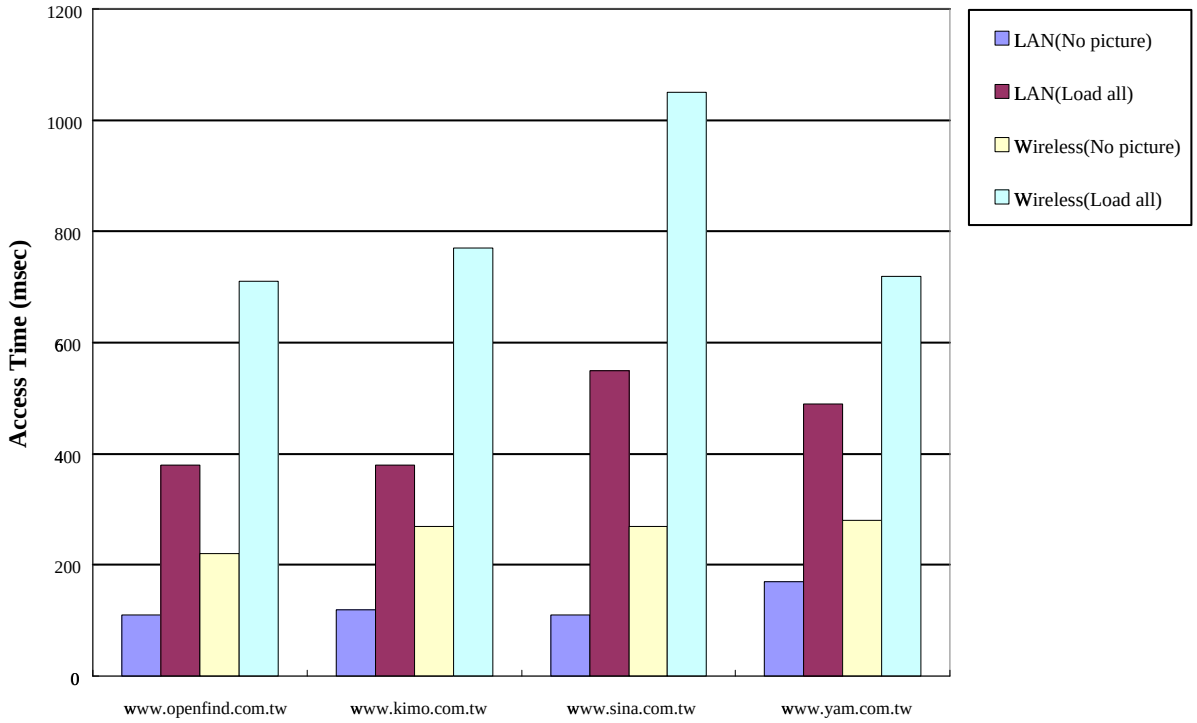

Fig. 8. Bandwidth Adaptation.

adaptation action of switching from full retrieval mode to text only mode (i.e. without loading the images). We measure the response time of both modes from four major ISPs in Taiwan under fixed network as well as wireless network. It is quite clear from Figure 8 that changing the retrieval mode on a Web browser is an effective way of responding to bandwidth drop in mobile environment.

As a summary, our preliminary implementation of the mobile event engine framework can effectively detect the change in resource availability and trigger registered actions for adaptation. The set of built-in actions we provided is also a valuable tool for responding to different resource challenging situations.

6. Conclusions and Future Work

We have proposed and implemented a mobile event engine to detect resource and environment status for mobile application programs and to trigger responsive actions when application specified events occur. Evaluation results demonstrate that our approach is quite effective as a mechanism for building adaptive applications. The work reported in this paper is part of a modular framework for adaptive mobile information management [19, 20]. The framework integrates an event engine and a rule system to facilitate intelligent adaptation in mobile environment. We are now in the stage of system integration and testing. We plan to apply our technology to turn a Web browser an adaptive application such that it can automatically adjust its browsing strategy in the constantly changing mobile environment.

References

1. Fred Douglis, P. Krishnan, and Brian Bershad. Adaptive disk spin-down policies for mobile computers. *Computing Systems*, 8(4):381–413, Fall 1995.
2. D. Duchamp. Issues in wireless mobile computing. In *3rd IEEE Workshop on Workstation Operating Systems*, pages 2–10, Key Biscayne, Florida, U.S., 1992. IEEE Computer Society Press.
3. G. Forman and J. Zahorjan. The challenges of mobile computing. *IEEE Computer*, 27(6):38–47, April 1994.
4. Stella Gatziu, K. R. Dittrich, and O. Shmueli. Composite event specification in active database : Model and implementation. Technical report, AT&T Bell Lab., Murray Hill, New Jersey, 1992.
5. N. H. Gehani and H. V. Jagadish. Event specification in an active object-oriented database. Technical report, University Zurich, Switzerland, 1993.
6. Andreas Geppert and Dimitrios Tombros. Event-based distributed workflow execution with EvE. In *Middleware'98*, pages 427–442, 1998.
7. David P. Helmbold, Darrell D. E. Long, Tracey L. Sconyers, and Bruce Sherrod. Adaptive disk spin-down for mobile computers. *Mobile Networks and Applications (MONET) Journal*, to appear.
8. T. Imielinski and S. Vishwanathan. Adaptive wireless information systems. Technical report, Department of Computer Science, Rutgers University, 1994.
9. R. H. Katz. Adaptation and mobility in wireless information systems. *IEEE Personal Communications*, 1(1):6–17, 1994.
10. Norman W. Paton and Oscar Díaz. Active database systems. *ACM Computing Surveys*, 31(1):63–103, March 1999.
11. E. Pitoura and B. Bhargava. Building information systems for mobile environments. In *Third International Conference on Information and Knowledge Management*, pages 371–378, November 1994.
12. M. Satyanarayanan. Mobile information access. *IEEE Personal Communications*, 3(1), February 1996.
13. M. Satyanarayanan, B. Noble, P. Kumar, and M. Price. Application-aware adaptation for mobile computing. *Operating System Review*, 29, January 1995.
14. B. Schilit, N. Adams, and R. Want. Context-aware mobile applications. In *IEEE Workshop on Mobile Computing Systems and Applications*, Santa Cruz, CA, U.S., December 1994.
15. James Vera, Louis Perrochon, and David C. Luckham. Event-based execution architectures for dynamic software systems. In *Proceedings of the First Working IFIP Conf. on Software Architectur*, 1999.
16. T. Watson. Application design for wireless computing. In *IEEE Workshop on Mobile Computing Systems and Applications*, Santa Cruz, CA, U.S., December 1994.
17. G. Welling and B. R. Badrinath. Mobjects: Programming support for environment directed application policies in mobile computing. In *ECOOP'95 Workshop on Mobility and Replication*, August 1995.
18. J. Widom and S. Ceri. *Active Database Systems: Triggers and Rules for Advanced Database Processing.* Morgan Kaufmann, San Francisco, California, 1996.
19. Shiow-yang Wu and Chun-Shun Chang. An active database framework for adaptive mobile data access. In *Workshop on Mobile Data Access, 17th International Conference on Conceptual Modeling, Singapore*, 1998.

20. Shiow-yang Wu and Shih-Hsun Ho. Active rule system for adaptive mobile data access. In *MDA'99: 1st International Conference on Mobile Data Access, Hong Kong*, 1999.