(1) AOE—A Mobile Computing Platform

Nayeem Islam, Dong Zhou,
Shahid Shoaib and Masaji Katagiri

This article introduces the AOE proposed by DoCoMo USA Labs, which is a computing platform that maintains and enhances the runtime performance and end-user's experience by adapting to changing runtime conditions in mobile computing environments.

1. Introduction

Over the last few years, there has been a large deployment of wireless data services by carriers all over the world. A vast majority of the deployed systems assume a relatively simple client that is browser-based and non-programmable. Two dominant examples of systems that have proliferated this model are i-mode [1] and Wireless Access Protocol (WAP) [2]. These mobile terminals suffer from a variety of different problems including poor performance, inability to allow the user to work in unconnected network, poor User Interface (UI) and do not enable simple Peer to Peer (P2P) applications, leading to overall poor user experience.

At DoCoMo USA Labs, we aim to realize a web-based application platform that can utilize current web-based services and applications with better runtime performance by making effective use of client-side resources. Our proposed Agile Operating Environment (AOE) provides the following three features:

1) On–device Adaptive Replication of Server-side Program Code

Mobile users may perceive poorer experience because of longer response latency, lower throughput, wider variation in responses, and more likelihood of disconnection. AOE allows the caching of server-side code units on the client device to alleviate such problems. Such on-device caching and execution is adaptable in that it can dynamically make caching decisions at runtime depending on environment changes, and that once the code is cached, it can dynamically select either to invoke the client cached copy or the code residing on the server.

2) On-device Dynamic UI Binding

A problem baffling Web applications targeted to mobile devices today is the heterogeneity in the interaction capabilities of these devices. Ideally, an application should be written only once, independent of any specific device. Yet, when the application is used on different devices, its presentation should be automatically adjusted to a form optimal to the device. AOE allows the binding between the abstract description of the presentation and its implementation to occur on the client device whenever there is enough resource on the device to do so. In addition, such binding is dynamic so that implementation of the presentation can change in accordance with the change in resource availability.

3) On-device Support for Adaptive Fault Tolerance[*1]

One of problems with current existing approaches for fault tolerance is that they are inflexible and do not adapt to changing system conditions or application requirements. But, mobile environment are characterized by change and hence, no one fault tolerance mechanism will work for all instances or all the time. We believe that dynamically adapting fault tolerance addresses this issue. Dynamic changes to the fault-tolerant mechanism can be made by separating the processes performed during fault-free operation from the processes performed when a fault has occurred and by encapsulating the processing routines with interfaces. This also enables part of the processing to be allocated to the client-side processor.

An important feature of this platform is that these facilities can be customized for devices of different capabilities, reconfigured by user or application to adjust to runtime resource availability, and self-adapt to changes in resource availability, to enhance user experience in mobile environments.

2. AOE Architecture

AOE supports web-based applications where users request services through a browser from their user client device. It specifically targets applications that involve dynamic, personalized content.

A typical AOE application consists of a cluster of Mervlets [3] that can dynamically generate Web pages. A Mervlet is sim-

*1 Fault tolerance: The ability to continue functioning and complete a process without conflicts by performing a recovery process when an error has occurred. Different measures are taken for different types of fault. Fault tolerance is often used between servers that perform critical processes (e.g., systems that perform commercial transactions).

ilar to a Servlet [4] except that it can be replicated and executed on client devices as described above, its UI can be dynamically attached, and it can recover from faults in the client terminal, network or server.

The AOE runtime is an environment for the execution of Mervlet applications. In this platform, the client and server each run an instance of the runtime environment (**Figure 1**). HyperText Transfer Protocol (HTTP) requests issued from the terminal's browser are intercepted by the AOE runtime environment (client AOE) on the terminal. The client AOE then either:

- passes the request to the server-side AOE runtime environment (Server AOE) using any form of transport (When the server and client support this transport, it is possible to use the Reconfigurable Messaging System (RMS) proposed by DoCoMo USA Labs [3]), or
- serves the request locally when the requested page is locally available or when the client AOE decides to execute the requested Mervlet locally.

The response, which is usually the presentation of the application described in languages such as HyperText Markup Language (HTML), is then optionally re-bound with the user interface library deployed on the client device, and finally returned to the browser in HTTP format.

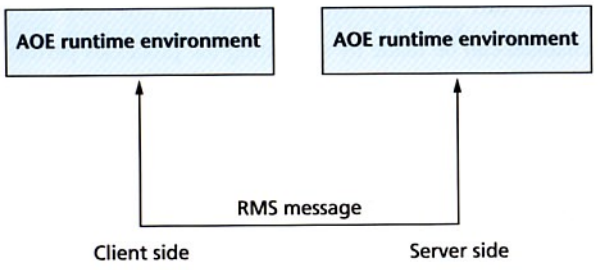

Figure 1 Symmetric AOE model

The replication manager is responsible for facilities that decide where to serve the requests, and the UI composer is responsible for facilities that dynamically re-bind the UIs of applications (**Figure 2**). When required, the client AOE and server AOE cooperatively assure that once client AOE receives an HTTP request from the browser, it will serve according to a reliability guarantee such as processing the request once and only once. Such reliability assurances are provided by the Adaptive Reliability Manager (ARM). A key feature of our system is that the basic system facilities are adaptable. The three key facilities—UI Composer, Replication Manager (RM), and RMS—are adaptively controlled by three adaptation managers (UI Adapter, Replication Adapter and ARM, respectively) that make adaptive decisions based on input from the Preference Manager and the Capability Profiler. An adaptation coordinator coordinates the individual adaptation managers of each facility. Detailed descriptions of each component can be found in Reference [3].

3. On-Device Adaptive Replication

Existing technologies, including web caching and prefetching*2 techniques, can be applied to mobile web applications and to some dynamic applications that produce static web pages.

*2 Prefetching: A technique whereby data (web content) likely to be needed by a user is automatically read in before it is requested. This often involves deciding which web pages to read in based on clues derived from the structure of web content or the user's previous activity.

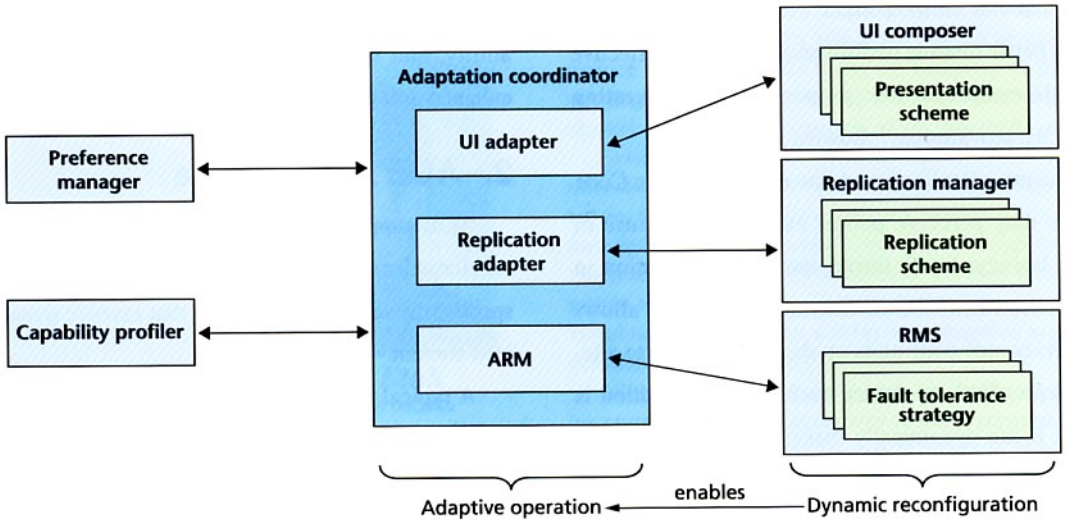

Figure 2 Layered components of the AOE runtime environment

However, these technologies are not necessarily effective for dynamic personalized web content generated by server-side code units (such as servlets) that are widely used in mobile Web applications. AOE handles this sort of dynamic personalized web content by replicating server-side code units on the client terminal.

Mervlets are replicable active objects. A Mervlet is composed of classes that define the Mervlet, read-only data used by the Mervlet, and mutable data/application state of the Mervlet. The mutable states of an application shared by a number of users can typically be divided into two disjoint sets: those that are private to a user or session, and those that are shared by all the users or sessions. In our system, applications are maintained at the required level of consistency by subjecting this data to the bare minimum of synchronization.

The selection and populating of a replica is customizable and adaptable in that devices, servers and applications can define their own triggers for selecting a device as a replication site and populate the site, and the dynamic capabilities provided by the AOE runtime are used to automatically evaluate the predicates for the trigger. Adaptation in replica invocation is supported by per-request replica selection for invocation. That is, for each request received by the client, the client- and server-side AOE runtime will collaboratively decide which replica to use for this particular request.

4. On-Device Dynamic UI Binding

One of the challenges in developing application for a ubiquitous environment is the user interface adaptation for different types of devices. Traditionally the problem is solved using a server proxy that adapts content on its way to a device. The presentation parts of these applications can be written in using eXtensible Markup Language (XML) [5] and eXtensible Stylesheet Language Transformations (XSLT) stylesheets [6]. As different types of devices, these types of solutions are cumbersome to update and maintain. Also, these solutions do not utilize the available resources in the new smart devices where the application may be fully device resident.

In AOE, we require applications to write their presentation in tags [7] to translate XML for rendering. However, the tags are not statically bound to the application but are constructed as dynamically attachable libraries that may be chained together at runtime in a specific order.

To enable dynamic presentation binding we choose a model similar to Java Server Pages (JSP) [4], which is called Mervlet Server Pages (MSP). In this model—unlike JSP libraries where UIs are embedded in the applications at development time—a UI proxy is added to the program. At runtime the UI Adapter attaches the appropriate library to enable the presentation of the application.

5. On-Device Support for Adaptive Fault Tolerance

Support for adaptive fault-tolerance is realized by the RMS and recoverable Mervlets.

The RMS provides configurable message delivery functionality in the Mervlet environment. This functionality is encapsulated under an interface called the Reconfigurable Messaging System–Failure Free Strategy Interface (RMS-FFI). An application only calls methods via this interface. The methods that are actually executed are set by the ARM based on criteria given by the user or application. For example, the RMS can be set up to use a point-to-point messaging service or to use a centralized messaging server (such as Java Message Service (JMS)).

At the application level, recoverable Mervlets are used to provide fault tolerance. Recoverable Mervlets allow the same application to use different fault tolerance mechanisms according to circumstances. For example, a web mail application may be configured to be more reliable for corporate email than personal email.

Dynamic reconfigurability support in fault-tolerance is achieved by allowing the two main components, the RMS and the Recoverable Mervlet, to have different failure free and recovery strategies, which can be set dynamically by the adaptive reliability manager as shown in **Figure 3**. The separation between failure free and recovery strategies makes it easier to develop multiple fault recovery strategies corresponding to a failure free strategy. For example, in case of RMS, one recovery strategy may prioritize the order in which messages are recovered while another fault recovery strategy may not.

Using the adaptability of this fault-tolerance support, we implemented a facility that allows the server-side logging to be turned on or off according to the server load. When the server is heavily loaded, the ARM can reconfigure the RMS to suspend server-side logging. Under certain circumstances this can significantly improve the response time.

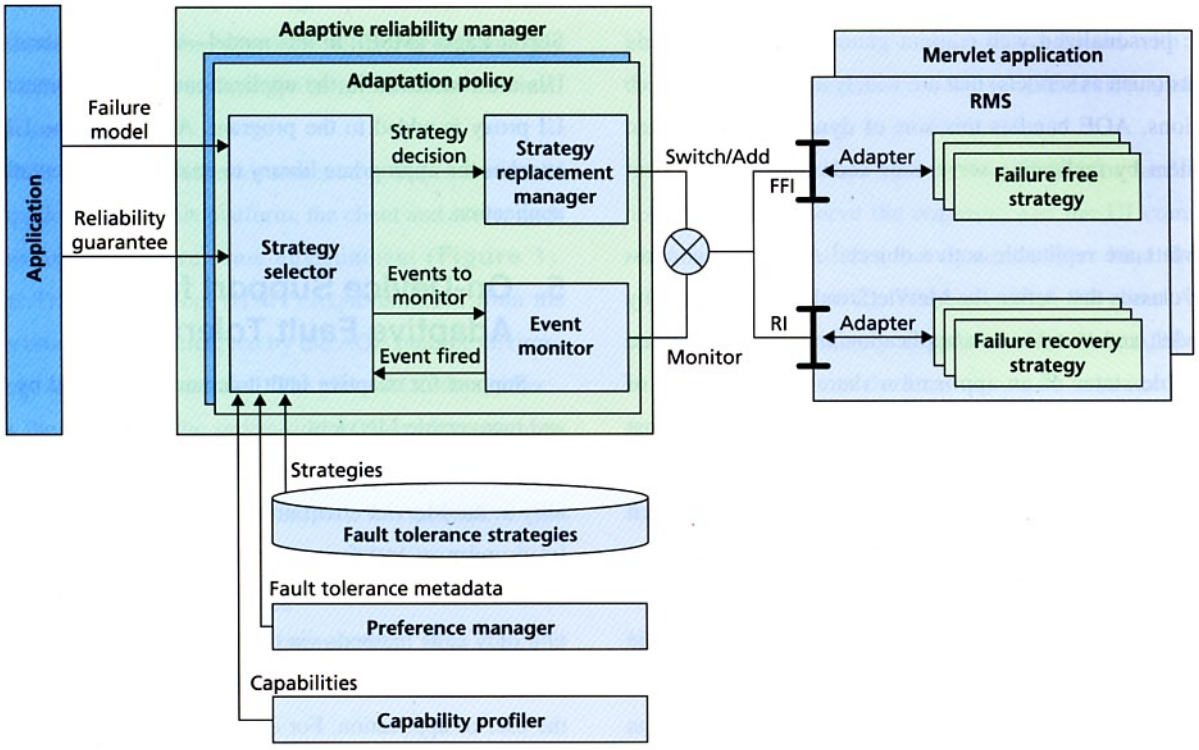

Figure 3 Reliability support in AOE

6. Implementation and Evaluation

We have implemented a prototype AOE runtime environment and a number of sample applications. Here, we introduce a number of results of evaluation tests performed using this prototype.

6.1 Benefits of Adaptive Replication

Figure 4 shows user perceived response time for the WebChess application under the following three scenarios:

1) Server load: Low; Adaptive replication: Prohibited
2) Server load: High; Adaptive replication: Prohibited
3) Server load: High; Adaptive replication: Enabled

In this test example, replication was performed at step #4 due to the long response time at step #3. (Here, each step corresponds to a pre-defined step in a chess application.) This graph shows that the combined cost of replica populating and local execution is even slightly (about 6.6%) lower than when the steps continue to be executed on a heavily-loaded server. (The transfer file size needed for replication in WebChess is about 37 kB.) After replication, the response time decreased significantly, and was only slightly longer than when the server was run with no load despite that the laptop is computationally less capable than the server desktop.

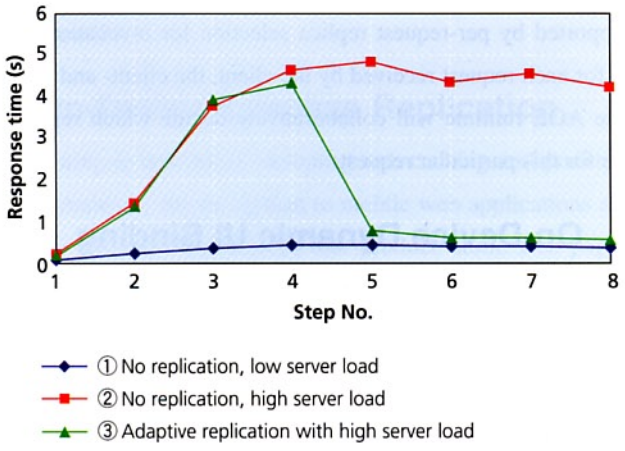

Figure 4 Changes in the response times of WebChess (replication at step #4)

6.2 Evaluating the Overall Effects of Adaptation

Figure 5 shows an example of how the response time is improved by the combined adaptive behavior of the system for the Web Calendar application.

In this test, we studied the adaptive behavior of the system by loading the server with incoming client requests from three computers which were simulating 100 clients per machine. As a result of the increased server I/O caused by server-side logging, the server load increased and the response time went up (> 20 seconds). At this time (step #3) the system stopped logging

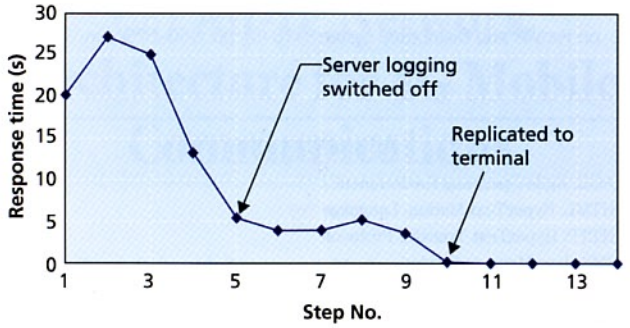

Figure 5 Effectiveness of combined adaptations

messages on the server side by adaptively modifying the message logging process. As a result, the response time fell to around five seconds (after step #5). The system then judged that the reduction of response time was still insufficient, and migrated the Web Calendar application itself to the client terminal (step #10). As a result, the response time was once again substantially reduced (< 10 ms). These results confirm the effects on the runtime performance of introducing adaptive replication and adaptive fault tolerance.

7. Related Works

Several systems have been developed with similar goals to our own, including Rover [8] and Odyssey [9]. However, an important aspect of our system that distinguishes it from these other systems is that we have focused our attention on performance and adaptation.

Server-managed replication is mainly used to enhance the availability and reliability of services and for server-side load balancing (load distribution) [10]–[12]. The study we have described here is different in that the client terminals have some degree of control over each stage of replication. Although Rover and Coda use client-controlled replication to support offline operations [8][13], they do not address the dynamic creation of replicas or adaptability in service invocation or data access.

With regard to fault tolerance, our approach differs from that of existing work such as Rover [14] which provides tools for producing reliable mobile applications, and instead we adopt a new approach in which the application response time is significantly improved by adaptively switching logging from the server to the client. Furthermore, we have proposed a unique framework that allows multiple failure recovery strategies to be provided for a specific failure free strategy. This feature is not supported by existing adaptive fault tolerance systems such as [15] through [18].

8. Conclusion

Mobile environments change continuously, and this affects the runtime performance and the end user's experience. In this article, we have presented a system called AOE that adapts itself to the prevailing conditions in order to mitigate the degradation of runtime performance and user experience. In particular, we have discussed three key facilities—service replication, reconfigurable messaging, and dynamic UI binding—that are essential for realizing a comfortable user experience. An important feature of our system is that these three facilities are able to adapt at runtime to the system conditions and the characteristics of the client terminal. We have also demonstrated how multiple runtime adaptations in different parts of the system can work together to improve system performance. Although we have not addressed security issues in this article, we are currently treating the security aspects of our system as a matter of top priority.

REFERENCES

[1] Enoki et al: "Special Issue on i-mode Service," NTT DoCoMo Technical Journal, Vol. 1, No. 1, pp. 4–30, Oct. 1999.
[2] M. Van der Heijden and M. Taylor: "Understanding WAP: Wireless Applications, Devices and Services," Artech House, 2002.
[3] N. Islam, D. Zhou, S. Shoaib, A. Ismael and S. Kizhakkiniyil: "AOE: A Mobile Operating Environment for Web-based Applications," To appear in Proc. Of IEEE Symposium on Applications and the Internet (SAINT) 2004.
[4] M. Hall: "Core Servlets and JavaServer Pages (JSP)," Prentice Hall PTR, 1st edition, 2000.
[5] E. R. Harold and W.S. Means: "XML in a Nutshell, 2nd Edition," O'Reilly & Associates.
[6] M. Kay: "XSLT: Programmer's Reference, 2nd edition," Wrox, 2001.
[7] B. Shannon, et al: "Java 2 Platform, Enterprise Edition: Platform and Component Specifications," Addison-Wesley, 2000.
[8] A. D. Joseph, A. F. deLespinasse, J. A. Tauber, D. K. Gifford and M. F. Kaashoek: "Rover: a toolkit for mobile information access," In Proc. Of ACM SOSP-15, 1995.
[9] B. Noble, M. Satyanarayanan, D. Narayanan, J. E. Tilton, J. Flinn and K. R. Walker: "Agile application aware adaptation for mobility," In Proc. Of ACM SOSP-16, 1997.
[10] C. Pu and A. Leff: "Replica Control in Distributed Systems: An Asynchronous Approach," In Proc. of 1991 SIGMOD, May 1991.
[11] B. Liskov, A. Adya, M. Castro, M. Day, S. Ghemawat, R. Gruber, U. Maheshwari, A. Myers and L. Shrira: "Safe and Efficient Sharing of Persistent Objects in Thor," In Proc. of 1996 SIGMOD, Jun 1996.
[12] P. Felber, R. Guerraoui and A. Schiper: "Replicating Objects Using the CORBA Event Service?," In Proc. of FTDCS '97, 1997.

[13] J.J. Kistler and M. Satyanarayanan: "Disconnected Operation in the Coda File System," ACM Transactions on Computer Systems, 10 (1), 3-25, Feb. 1992.

[14] A. D. Joseph and M. F. Kaashoek: "Building Reliable Mobile-aware Applications Using the Rover Toolkit," In Proc. of MOBICOM '96, Nov. 1996.

[15] I. Chang, M. A. Hiltunen, and R. D. Schlichting: "Affordable Fault Tolerance through Adaptation," Parallel and Distributed Processing, LNCS 1338, pp. 585–603. Apr. 1998.

[16] C. Sabnis, M. Cukier, J. Ren, P. Rubel, W. H. Sanders, D. E. Bakken, and D. Karr: "Proteus: A Flexible Infrastructure to Implement Adaptive Fault Tolerance in AQuA," In Proc. of 7th IFIP Working Conference on Dependable Computing for Critical Applications, pp.137–156, 1999.

[17] N. Venkatasubramanian, M. Deshpande, S. Mohapatra, S. Gutierrez-Nolasco, and J. Wickramasuriya: "Design and Implementation of a Composable Reflective Middleware Framework," In Proc. of ICDCS 2001.

[18] Z. T. Kalbarczyk, R. K. Iyer, S. Bagchi, and K. Whisnant: "Chameleon: A Software Infrastructure for Adaptive Fault Tolerance," IEEE Transactions on Parallel and Distributed Systems, 10 (6), pp. 560–579, Jun. 1999.

ABBREVIATIONS

AOE: Agile Operating Environment
HTML: HyperText Markup Language
HTTP: HyperText Transfer Protocol
JMS: Java Message Service
JSP: Java Server Pages
MSP: Mervlet Server Pages
P2P: Peer to Peer
RMS: Reconfigurable Messaging System
RMS-FFI: Reconfigurable Messaging System-Failure Free strategy Interface
UI: User Interface
WAP: Wireless Application Protocol
XML: eXtensible Markup Language
XSLT: eXtensible Stylesheet Language Transformations