A Middleware Support for Agent-Based Application Mobility in Pervasive Environments

Yu Zhou[1,2], Jiannong Cao[1], Vaskar Raychoudhury[1], Joanna Siebert[1], Jian Lu[2]
{csyuzhou, csjcao, csvray, csjsiebert}@comp.polyu.edu.hk, lj@nju.edu.cn
[1]Internet and Mobile Computing Lab
Hong Kong Polytechnic University, Hong Kong
[2]State Key Laboratory for Novel Software Technology
Nanjing University, Nanjing, China

Abstract—Application mobility is an efficient way to mask uneven conditioning and reduce users' distractions in pervasive environments. However, since mobility brings more dynamism and uncertainty, it also raises new research issues in developing pervasive applications, including underlying application models, adaptive resource rebinding mechanisms, synchronization and fault tolerance techniques, etc. In this paper, we approach these problems from the middleware perspective. Inspired by software agent's inherent capability of autonomy and mobility, we investigate its potential use in application mobility and propose an agent-based architecture called MDAgent. Three salient features are emphasized: 1) Reduced mobility overhead. Flexible bindings of application components avoid migrating whole application. 2) Simplified mobility management. Mobile agent takes over the responsibility of mobility and synchronization, so user intervention is reduced. 3) Enhanced customizability and adaptability. Context information can be updated dynamically, and ontology-based reasoning ability embedded in autonomous agents can direct the application to adapt to the changes accordingly. On top of MDAgent, we have developed several applications, and evaluated the performance.

Keywords: Pervasive computing; application mobility;software agent

1. INTRODUCTION

Recent years have witnessed the daunting progress in the integration of cyber space and its physical counterpart since Mark Weiser envisioned the computer for the twenty-first century [1]. Computers with higher processing ability are diversified into common consumer electronics connected by various kinds of networks. In this computation-pervasive environment, how to coordinate various kinds of smart devices and make them serve people in a more natural and less annoying manner becomes one of the main research concerns in both the academia and the industry community. Users have specific operation habits and preferences, and when they move from one place to another, it may cause some inconvenience in the new environment. For example, if one person is left-handed, he will certainly feel uneasy to work in right-handed application environments where he moves to. If the application can migrate with the user or be customized according to his preferences, and adapt to new environments proactively, it will become personalized and thus naturally reduce users' distraction [2].

However, making the application mobile, personalized and adaptable faces several challenges. The most fundamental two problems are when and how to migrate and adapt the application. Different devices usually have different properties, such as screen size, resolution ratio, and computation capability. Thus one application running well on one device can not be taken for granted that it would work well without any adaptation on another device. Meanwhile, in some cases, not only the cut-paste kind of application mobility, but also the copy-paste kind of mobility is needed. By cut-paste like application mobility, we mean that, applications (or parts of applications) save the states and migrate to the destination. By copy-paste like application mobility, we mean the application clone first and migrate. We use a metaphor to express this, as it is very like the everyday text editing operation. In the latter case, some synchronization channels need to be established between or among the involved applications.

An executing application generally consists of user interfaces, logic, computation states, and resource bindings, etc. We need to investigate the management of mobility, application architecture, and resource matching mechanisms. Besides, to capture users' movement and intention also requires the attention on context modeling and reasoning capability. The issues stretch from the application layer to the context layer, while current system software offers limited support for mobility and context management. Putting all these concerns in the application layer would be too much for application developers. The above observations motivate us to approach it from the middleware perspective, offering a middleware-level support for application mobility.

Inspired by the coincidence of software agent's inherent features and pervasive environments' requirements, we investigate and exploit agents' potential use in application mobility to support the vision of pervasive computing. Software agents generally contain two complementary semantics. The first is on mobility, and the second is on autonomy. Mobile agents (MAs) are programs that can migrate in a network at times and to places of their own

27th International Conference on Distributed Computing Systems Workshops (ICDCSW'07)
0-7695-2838-4/07 $20.00 © 2007 IEEE

choosing [3]; while autonomous agent is a system situated within and a part of an environment that senses and acts on it [4]. Various kinds of agent-based solutions have been proposed and proved to be feasible and efficient in a considerable amount of applications, ranging from software engineering to knowledge engineering [5].

Based on our preliminary work on agent-enabled application mobility [6, 7], we extend the research into the development of the underlying application model, synchronized mechanism and adaptation techniques. Our previous work proves the feasibility of agent enabled application mobility; however it doesn't further investigate the component-level migration, the clone-dispatch application mobility, and resource description and reasoning mechanisms. Compared with other works on application mobility in pervasive environments [8-10], our approach highlights the following characteristics:

i) Agent-enabled loosely coupled application architecture and flexible resource binding mechanisms support light-weight transmission.
ii) Agent-based coordination mechanism supports not only follow-me kind of mobility, but also clone-dispatch kind.
iii) Embedded logic-based reasoning utilities in autonomous agents support adaptive migration behaviors.

Besides the above features, employing agents can also leverage the existing methodology and architecture, thus getting the advantage of simpler persistence and mobility management as well as stronger resilience capability [3].

The rest of this paper is organized as follows. In Section 2, we review the related work to our research. In Section 3, we discuss architectural requirements. Based on these requirements and previous analysis, we present the design of our architectural framework in Section 4. Implementation of the applications in our proposed framework and their performance evaluations are described in Section 5. Finally, Section 6 gives the concluding remarks.

2.RELATED WORK

The idea of application mobility results from the requirements of personalized and adaptive services in pervasive environments. Several other research projects have worked on this aspect. For example, there are Gaia, Aura, BEACH, one.world, to name a few. Comprehensive surveys of research in application mobility are impractical in this paper, thus we select and highlight some representative examples in the literature.

Gaia [8, 11] models the pervasive environment as an active space where the application framework uses reflection to explicitly separate the application base-level from meta-level. Applications are decomposed into five parts, namely, model, presentation, adapter, controller and coordinator. Computational reflection manages the complexity in the development of applications, allowing developers to concentrate on the base-level and providing mechanisms to automate meta-level configuration dynamically. The coordinator manages the component registration, application's life cycle and mobility. However placing all these responsibilities into a single static coordinator module will unavoidably increase its complexity and cause the problem of single point of failure. Besides, Gaia lacks a unified resource definition framework.

Project Aura [12] aims to offer a framework for user mobility in pervasive environments. Applications are organized into services. User tasks become first class entities represented as coalitions of abstract services. The task manager will coordinate these services and transmit them accordingly through a file transfer system after they sense users' mobility. However, Aura didn't address much about adaptation issues after the migration. Besides, inter-space application transmission as well as multi-application synchronization issues have not been further investigated.

BEACH [13] is a software infrastructure providing functionality for synchronous cooperation and interaction with room-ware components. It uses an event dispatching mechanism to support multiple persons using the same devices concurrently. The synchronization is realized through shared objects. When the state of these shared objects changes, the updates are triggered automatically. This is somehow similar to the update mechanism used in our system. But the emphasis of BEACH is to support synchronized multiple devices collaboration, while our research mainly addresses application mobility.

Previously, we also designed an agent-enabled platform supporting application-level mobility [7]. We further the investigation from the aspects of the underlying application model, the mobility management and the separation of concerns in agents. The original framework uses a static binding between agents and applications while the current one adopts an adaptive binding mechanism, in which only parts of application need to be wrapped to migrate. It can help reduce the migration cost significantly which can be shown in the performance study. Our original framework supports only follow-me kind of mobility, we extend this and also supports clone-dispatch mobility. Also, the reasoning functionalities are separated and incorporated into specific autonomous agents; while these functionalities were formerly mixed together in mobile agents. This separation of concerns also facilitates the agents design because different agents just need to concentrate on their specific roles.

3.ARCHITECTURAL REQUIREMENTS

In this section, we identify some key requirements that should be addressed in the architecture design for application level mobility in pervasive environments.

3.1 Application Model

Component level migration is more desirable compared with migrating the whole application. This requires the applications to support flexible component binding and composition. Pervasive environments offer various kinds of network connections. By leveraging this infrastructure, applications can be designed as a collection of reusable

distributed objects. The requirements of the application model can be summarized as follows:

i) Applications should be decomposed into separate parts, such as logics, presentations, resources, data, etc.;
ii) To coordinate these components, Synchronization mechanisms need to be provided;
iii) Before and after migration, application states should be consistent and continual, so a state manager component should be provided;
iv) Various kinds of devices and network conditions exist, so the adaptation mechanisms are also required.

Application models that take the above requirements into consideration can significantly ease this process to achieve environment-adapted and user-customized application level mobility.

3.2 Mobility Management

What distinguishes application mobility from other kinds, such as data mobility, etc. is that application is a proactive and executable entity. After migrating to the destination, it can continue its execution in the new environment. Basically, there are three aspects to consider. First, which components should be migrated? Second, where is the destination, in the same virtual space or across the space? Third, what kind of migration is needed, cut-paste like or copy-paste like?

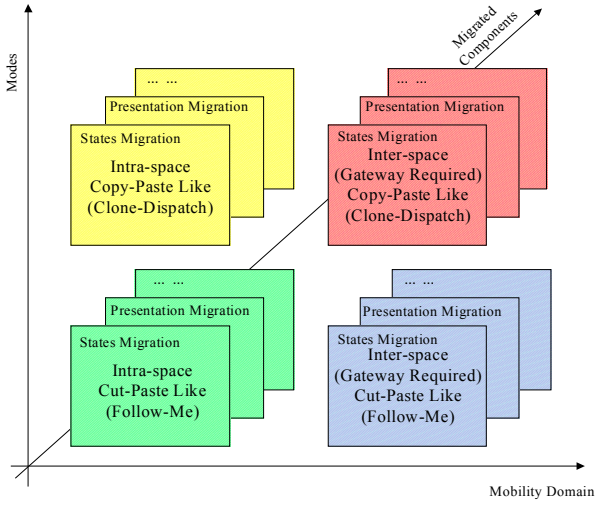

Fig. 1 Mobility classification

Figure 1 gives a mobility classification illustration. In the dimension of mobility modes, there are two categories, one is following the user's location and the other is cloning itself and dispatching to the destination. Application-level clone and dispatch modes are intuitively similar to copy-paste and cut-paste operations respectively, but involve many more concerns. In some cases, we need cut-paste like mobility, for example, when a person is listening to a piece of music, but he has to go to other places for some reason and he doesn't want the music to be interrupted. Now, the best way is, the music player can stop when he moves out (*cut*) and continue when he enters the new place (*paste*). While in other cases, we need copy-paste like mobility. For example, in conference scenarios, we often face the following embarrassments, one or several of the members cannot come due to various kinds of reasons and in this way, the meeting applications might clone themselves (*copy*), and move the copy to the destination (*paste*). The application would start automatically and synchronize with the source application at the destination. Along with the voice transmission, a remote meeting would be possible. In the dimension of mobility domain in Fig.1, due to the current technology limits of coverage, generally one smart space only covers a specific area. Migration across the space boundary requires additional gateway support. Also, applications should be aware of which parts of the components can be migrated, data, presentation, logics or other components. The mobility management design should take these into consideration.

3.3 Resource Binding & Service Customization

As discussed in the Section 1, after the migration of applications' components, for various reasons, the original resource bindings may be lost. For example, if the network is busy and destination machine has the required resources, then the local resource can be used without the need to transfer resources from the remote source host. This requires a resource rebinding mechanism. As different hosts often have the same resources but with different names, simple syntax-based matching puts much strict unnecessary constraints, and semantics-based resource matching is much preferred.

Service customization has two categories. The first is for different devices, while the second is for different users. This requires explicit specifications for these two cases, and an introspection ability of applications to adapt to different scenarios.

3.4.Context Awareness

To capture users' mobility or intention, the application should be aware of users' current context which involves any information that can be used to characterize the situation of an entity relevant to the interaction between a user and an application [14]. Since application's mobility and customizability are strongly connected with users' locations and personal preferences, in system design, this kind of context should be specifically paid attention to.

Different context information often has different properties. For example, users' location information usually changes frequently as people often move from one place to another, while users' preferences or operational habits are generally more stable. Modeling different context information also requires taking their temporal characteristics into consideration.

Usually, the underlying sensors can only collect raw data such as distance, badge (listener) identity, etc. To map these data to useful information such as location, user identity, etc. requires context fusion mechanisms. Besides, some context reasoning and prediction functionalities should also be provided to improve the performance.

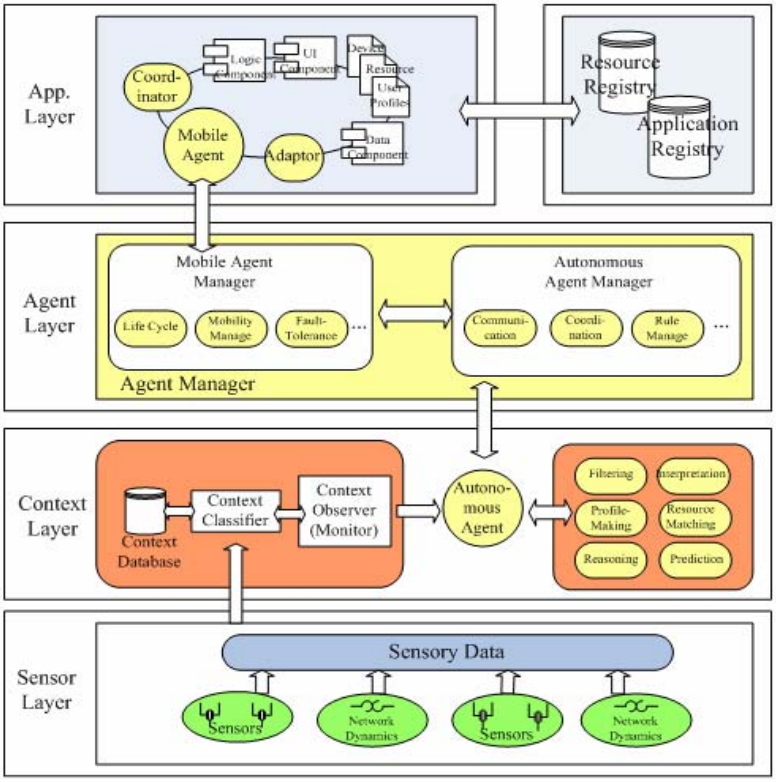

Fig. 2 Framework Overview.

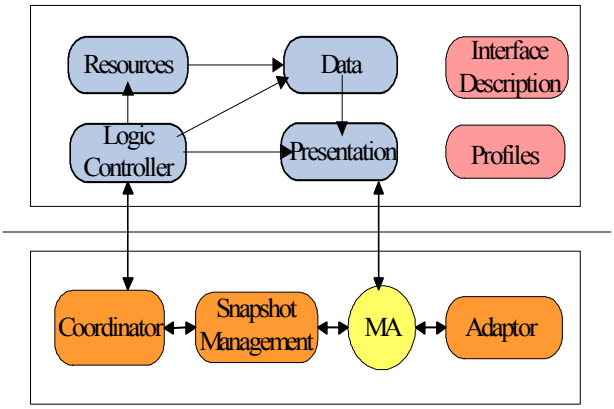

Fig. 3. Application Model

4. ARCHITECTURAL FRAMEWORK

In this section, we introduce our architectural framework addressing the issues discussed above.

4.1 Framework Overview

Figure 2 gives the general view of our architecture design. The architecture is comprised of four layers, i.e., Sensor Layer, Context Layer, Agent Layer, and Application Layer.

Sensor layer will collect data from these physically or logically deployed sensors detecting users' mobility, network connectivity, latency, etc. Due to the variety and frequent inaccuracy of these data sources, they cannot be used directly in the upper level.

In context layer, first, a classifier component will store the data into different databases according to their temporal characteristics. A context monitor will observe this process. If some predefined conditions occur, the autonomous agents will be triggered and these agents will continue the following process.

Agent layer is the key to connect the context layer and the application layer. It consists of two kinds of agent managers, one is the mobile agent manager, and the other is autonomous agent manager. Autonomous agent (AA) is responsible for reasoning and decision-making according to the data received from context layer. Mobile agent (MA) is responsible for the wrap of application components. They communicate through message passing. When autonomous agent finds user's movement or user's indication to move an application to a remote host (cut-paste kind or copy paste kind), it first notifies the MA to prepare to migrate, and record the application state. After getting the destination, MA retrieves complied resource and application information (maybe owl-enabled as can match in a semantic way) from the registry center. Then according to the result and the application-specific rules, AA decides whether to transfer the states only or the interface only or other possible component combinations in application layer. Mobile agents will take over the next transmission and synchronization work according to the application-specific requirements.

4.2 Application Management

To support highly customizable and adaptable applications, we proposed a loosely-coupled application model based on the Observer Design Pattern [15].

4.2.1 Application architecture

Our application architecture has two levels as shown in Fig. 3. Upper level mainly consists of some application components, such as logics, presentations, resources, etc., together with some description files, such as user profiles, device profiles, resource profiles and interface descriptions. Logic controller handles the processing of data and resources and controls the presentation components. As this level directly interacts with users, it is visible to them.

In base level, the main modules are coordinator, snapshot manager, mobile agent, and adaptor. The coordinator establishes the synchronization link between different presentations and interacts with snapshot management and mobile agent. Basically, different presentations register themselves to the coordinator. When the states change, these

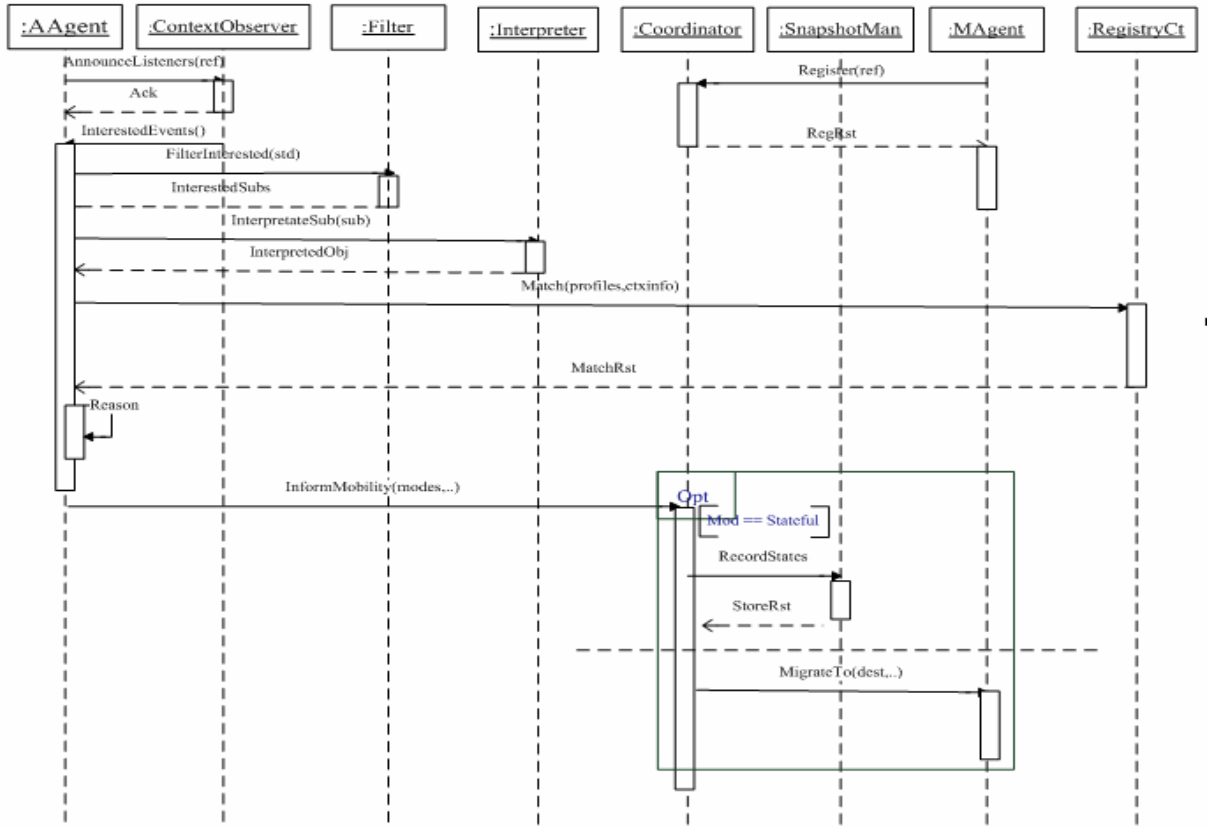

Fig. 4 Interaction Diagram

presentations can get notified automatically. In this way, not only we get a loosely-coupled architectural model but also simplified consistence control and higher component reusability. The snapshot management is responsible for persistence process control of running applications, while mobile agent is for application wrapping and migration to the destination. Due to the high dynamism and variety in the environments, the adaptor comes to bridge the mismatch. As this layer mainly deals with underlying supporting work, it is transient to end users.

4.2.2 Dynamic interaction

Applications first register themselves to the application and resource registry centers with their interface descriptions and other parameters such as specific device requirements, user preferences, etc, in a WSDL-like format. When the mobile agent gets the message of migration, it firstly parses the scripts. If the migration is follow-me alike, it contacts the registry centers first to find whether the destination has the corresponding components and resources. Then it suspends the current execution of application, collects and wraps the snapshots together with corresponding components, migrates to destination hosts and resumes application execution there. If the migration is clone-dispatch alike, it also looks up in the registry center first to find whether the destination host has required resources and components.

After migration, the application needs to be adapted in the new environments, the mobile agent will contact adaptor to conduct necessary adaptations according to some customizable parameters to adjust some sizes, resolutions, etc.

4.3 Agent Management

Figure 2 shows some key components of agent management. In this section, we will elaborate on them and other agent-related components.

In our model, the agents function like a thread weaving the applications and the context management. Specifically, we distinguish two kinds of agents according to their different roles. They are autonomous agent and mobile agent respectively. These agents collaborate together and interact closely with both the application layer and the context layer.

The Autonomous agent manager mainly has the communication and coordination utilities and serves as a rule manager for autonomous agents. These agents will logically exist in the context layer and listen to the context events. The context observer continually monitors and broadcasts the context information. Not all of this information is useful. Some are duplicates and some are irrelevant. Agents will filter and find their interested subjects and interpret them accordingly. For example, when the context observer finds user's location being changed and announces this event, autonomous agents will capture this information and interpret it as the user will leave the room and inform the coordinator. The coordinator will subsequently call for snapshot manager to record the current application states if necessary, and then suspended the application. When the user's new location is announced, autonomous agents will firstly check application related profiles including resources, preferences, and device properties. Then the autonomous agents will contact the

registry center about destination environment information, such as, whether the devices are compatible, if the application components exist there, whether the network situation allows the local data to be copied. Based on the above considerations and user defined rules, the autonomous agent will decide whether and what parts of application will be shipped to the new environments through a message to the mobile agent manager.

Mobile agent will wrap the corresponding components, check out from the current site, check in at the destination, inform the coordinator to establish the synchronization link if necessary and resume the execution. The interaction is pictorially described as a sequence diagram in Fig. 4. In this way, mobile agent is not bounded to a specific component of applications; instead it can wrap any serializable part and migrate to the destination.

4.4 Resource Description & Agent Reasoning Mechanism

In pervasive environments, various kinds of resources with different properties exist. Some are transferable, others are not; some can be easily substituted, others can not. For example, a printer is not transferable but can be substituted while database is neither transferable nor easily substituted, and a PDA is transferable but not easily to be substituted as users' profiles and preferred software are installed. In order to share and utilize these resources, a representation framework is in need. We use ontology to model the resources and their inter-relations, as it not only supports resource matching semantically, but also facilitates the reasoning process.

In the domain of knowledge-based systems, ontology means a specification of a representational vocabulary for a shared domain of discourse -- definitions of classes, relations, functions and other objects, as in software literatures, what "exists" is exactly what can be represented [16].

To support ontology, one description mechanism must be selected. We choose Web Ontology Language (OWL) for its generality. OWL [17] is a semantic markup language for publishing and sharing ontology proposed by W3C's Web Ontology Working Group. It is developed as a vocabulary extension of Resource Description Framework (RDF). OWL follows the XML syntax and has the advantage of platform-independence. For example, we can define a specific printer in this way:

```
<owl:Class rdf:ID = "hpLaserJet">
<rdfs:comment>hp color printer</rfds:comment>
   <rdfs:subClassOf                                    rdf:resource=
        "#Printer;Substitutable;UnTransferable"/>
        <owl:ObjectProperty rdf:ID="locatedIn">
            <rdfs:range  rdf:resource="#Office821"/>
            <rdfs:type rdf:resource="TransitiveProperty"/>
        </owl:ObjectProperty>
      ...
</owl:Class>
```

Fig. 5 Owl Description Illustration

By abstracting and specifying some key properties in OWL format, we can check the resource compatibility semantically and customize the application accord to the checking results and other context information. First, an autonomous agent will retrieve the resources available in the destination host from the registry center in the standard OWL Query Language (OWL-QL) and then carry out the compatibility checking using predefined rules which can be encoded in a RDF format as the following script shows. The example script in Fig.8 means predicate '*locatedIn*' is a transitive property; if the resources in the source and destination are both the 'printer' types, then they are compatible; and if the resources in sources and destinations are compatible and network condition is good (response time is less than 1000 ms), then the autonomous agent will issue a move command which will be transformed to a concrete action.

```
[Rule1: (?p imcl:locatedIn ?q), (?q imcl:locatedIn ?t)  ->
 (?p imcl:locatedIn ?t)]
[Rule2: (?ptr imcl:printerObj 'printer'), (?srcRsc rdf:type ?ptr), (?destRsc
imcl:printerObj ?ptr)                                  ->
(?srcRsc imcl:compatible ?destRsc)]
[Rule3: (?addr1 imcl:address ?value1), (?addr2 imcl:address ?value2), (?srcRsc
imcl:compatible  ?destRsc),  (?n  imcl:responseTime  ?t),lessThan(?t,
'1000'^^xsd:double)  ->
(?action  imcl:actName  "move"),(?action  imcl:srcAddress  ?add1),(?action
imcl:destAddress ?add2)]
……
```

Fig. 6 RDF Rule Illustration

When MA gets to the destination and resumes the application there, it will also check with the coordinator and make some adjustments according to the environment configurations.

5.IMPLEMENTATION & PERFORMANCE EVALUATIONS

In this section, we describe the implementation of a prototype of the proposed architecture for application mobility and some sample applications built upon the framework. The prototype is written in Java 1.4, and the agent server is JADE 3.4 [18]. We use several open source packages (in Jar file). Dozens of Cricket Sensors are deployed to collect user's location and identity data. The prototype consists of a running kernel of context management, MA manager, AA manager, and abstract application interfaces. Context kernel employs a publish/subscribe design pattern. When the subscribed events occur, the information will be multicast to the registered listeners. Both autonomous agents and mobile agents are implemented as specific agents inheriting JADE's Agent class. Jena [19] is used as the reasoning engine embedded in autonomous agents.

We built six demo applications based on this infrastructure, namely smart media player, follow-me editor, ubiquitous slide show, handheld editor, handheld music player, and follow-me instant messenger. Among these applications, we will introduce two of them as they demonstrate different kinds of application mobility. The first is a follow-me kind of music player. It can stop music when listener is out of the room and continue playing when the listener enters the room within the same space. In this demo, application is divided into several functional components, codec logic, interface, and data files. When the context

manager senses the change of user's location, it notifies autonomous agents, autonomous agents think the user is going to leave the room and issue a command to the coordinator suspending the current music, as this is a stateful application, coordinator will call snapshot to record the current states. When user enters a new place, context manager notifies autonomous agents, which first contact the destination hosts and check whether the required resource or application exists or not. In this case, the resource is the music files in the playlist. If these files don't exist in the destination, they will be played remotely through URL in the original host. We use Juddi and MySQL as the backend application and resource registry center. Autonomous agent first check whether the application exists or not in the destination. If it exists, mobile agent just wraps the state and migrates. Otherwise, it will also carry the logics and user interface as well as the states.

To evaluate the platform performance and without the loss of generality, we assume the destination host contains the application user interface but no music data nor application logic. We calculate the time consumption in three phases: suspension, migration takes and resumption. Time consumptions of suspension and resumption are easy to calculate, as they occur in the same place. But migration involves two places whose clocks are not synchronized. In this case, we calculate the round trip time cost. According to stable physical properties of crystal frequency, the difference of time values of clocks at the same time is nearly a constant value. In this way, adding up the round trip migration time cost can just eliminate the error introduced by asynchronization in different hosts, i.e.,

$$T_2@H2 - T_1@H1 + T_4@H1 - T_3@H2 =$$
$$T_2@H2 - T_1@H2 + T_4@H1 - T_3@H1$$

Note: Ti@Hj means time value at the moment of 'i', in Host (Place) j

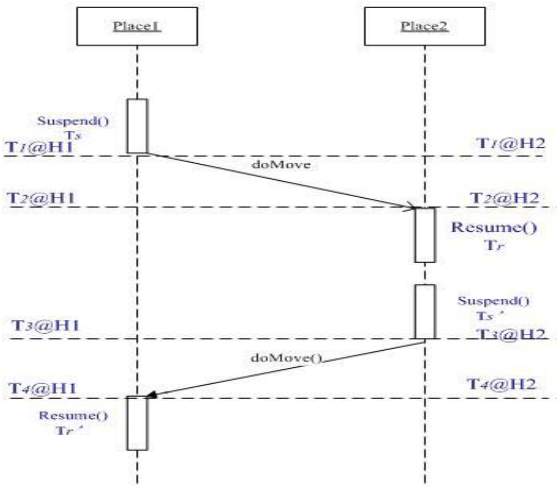

Fig. 7 Time Cost Illustration

In our previous work, for specific applications, we use a static binding between mobile agents and applications. In this way, application components including the data, logic, and user interfaces all migrate with users. It will decrease the performance when the applications' size grows up

In the experiment conducted, we use different sizes of music files. The evaluation result is shown in Fig.8. The experiment is done on 2 computers with P4 1.7GHz, 256M memory and PM1.6GHz, 512M memory respectively connected by 10Mbps Ethernet. The evaluation results hint that as the file size increases, only resumption takes more time, suspension and migration are not affected much. Although resumption takes more time, the total increased scale is acceptable, about less than 200 milliseconds when the file size increases from 2.0MB to 7.5MB.

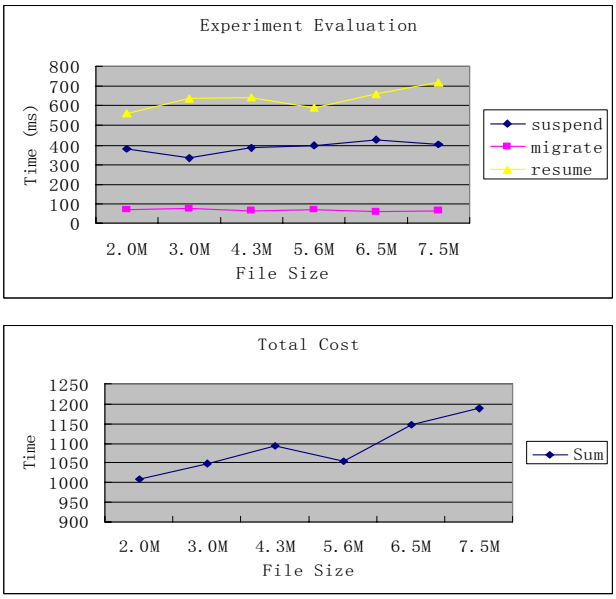

Fig. 8 Performance with adaptive component binding

In order to give a comparative view of the efficiency of the adaptive component migration, we also measured the time consumption in the original design [7]. The corresponding performance evaluation and comparison are given in Fig.9 and Fig.10.

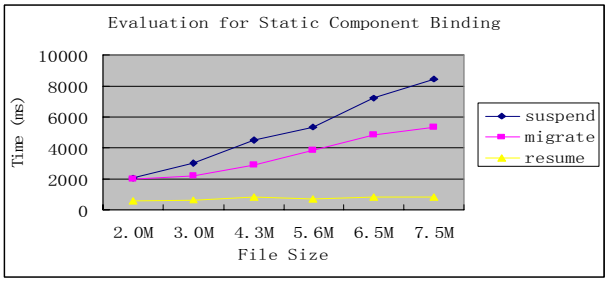

Fig. 9 Performance with static component binding

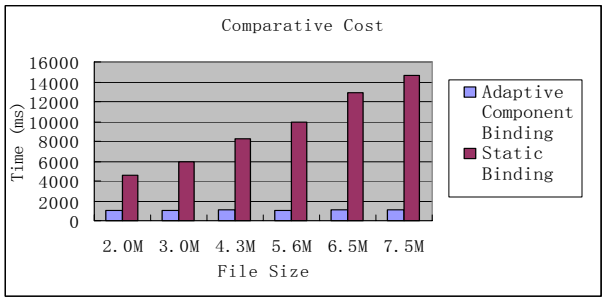

Fig. 10. Comparative time cost

The second application is to demonstrate clone-dispatch kind of migration. It needs to cross different spaces in our

case. One lecture is going to be given, but so many listeners that one room is not big enough to sit all of them. Parts of attendees are arranged in other meeting rooms. Traditionally, besides the audio transmitted to these rooms, separate assistants are needed to open the slides and synchronize manually with the main room (where the speaker is). Our demo simplifies this process and lets agent clone the application and migrate to the separate rooms and establish the synchronization links with the main room automatically. AAs get the context information from user indication and get the list of destinations, after resource retrieving and matching, it will notify MAs to migrate the components to the destination. In this case, each meeting room is equipped with a presentation application, a projector, what lacks is the slides. So MAs just need to carry to slides to the destination, collaborate with the MA manager and synchronize the slides with the speaker's presentation controls. Meantime, separate channels broadcast the speaker's voice. In this way, attendees can listen to the same lecture in different rooms. In our scenario, different rooms belong to different cyber domains, gateways are provided to connect them. In implementation, we import part of *Open Office Impress* as the slide show presenter and *Open Office SDKs* to get the controller handle. We refactored the program according to the structure model introduced previously and added the coordination components to synchronize the different presentations.

6. CONCLUSIONS

In this paper, we mainly exploit the potential use of software agents to support application-level mobility in pervasive environments.

We investigate the problem of application mobility from the aspects of the underlying application model, mobility management, collaboration of different kinds of software agents, resource matching and service customization mechanisms. By application migration, users can interact with environments in a more natural and comfortable way, and our experiments and experience have indicated that software agent technology is a promising approach to support application mobility.

MDAgent has some unique features which distinguish from other frameworks. It supports flexible, multiple kinds of application mobility. Semantics-based resource matching and reasoning mechanisms enable richer information process. The collaboration of autonomous agents and mobile agents achieves a higher level of migration capability and lower level of migration costs which are demonstrated by the experiments.

ACKNOWLEDGMENTS

This work is partially supported by the University Grant Council of Hong Kong under the CERG PolyU 5183/04E, China National 973 Program with Grant Number: 2002CB312002, NSF of China with Grant Number: 60403014, 863 Program of China with Grant Number: 2006AA01Z159.

References

[1] Weiser, M., *The Computer for the Twenty-First Century*, in *Scientific American*. 1991. p. 94-101.

[2] Satyanarayanan, M., *Pervasive Computing: Vision and Challenges*. IEEE Personal Communications, 2001: p. 10-17.

[3] Lange, D.B. and Oshima, M., *Seven Good Reasons for Mobile Agents*, in *Communications of the ACM*. 1999. p. 88-89.

[4] Franklin, S. and Graesser, A. *Is it an Agent, or just a Program?: A Taxonomy for Autonomous Agents*. in *Proceedings of the Third International Workshop on Agent Theories, Architectures, and Languages*. 1996.

[5] Iglesias, C.A., Garijo, M., and Gonzalez, J.C. *A Survey of Agent-Oriented Methodologies*. in *Procedings of the Fifth Workshop on Intelligent Agents V. Agent Theories, Architectures, and Languages*. 2000.

[6] Cao, J.N., Tse, C.K., and Chan, T.S. *PDAgent: a Platform for Developing and Deploying Mobile Agent Enabled Applications for Wireless Devices*. in *Proceedings of the 2004 Internationsl Conference on Parallel Processing*. 2004. Montreal, Canada.

[7] Yu, P., Cao, J.N., Wen, W.D., et al. *Mobile Agent Enabled Application Mobility for Pervasive Computing*. in *UIC*. 2006.

[8] Ranganathan, A., Chetan, S., and Campbell, R. *Mobile Polymorphic Applications in Ubiquitous Computing Environments*. in *Proceedings of the First Annual International Conference on Mobile and Ubiquitous Systems: Networking and Services*. 2004.

[9] Grimm, R., *System Support for Pervasive Applications*. December, 2002, University of Washington.

[10] Ponnekanti, S.R., Lee, B., Fox, A., et al. *ICrafter: A Service Framework for Ubiquitous Computing Environments*. in *Proceedings of Ubicomp 2001: Ubiquitous Computing: Third Internaltional Confernece*. 2001.

[11] Roman, M., Hess, C., Cerqueira, R., et al., *A Middleware Infrastructure for Active Spaces*, in *Pervasive Computing, IEEE*. Oct-Dec, 2002. p. 74-83.

[12] Garlan, D., Siewiorek, D., Smailagic, A., et al., *Project Aura: Toward Distraction-Free Pervasive Computing*, in *IEEE Pervasive Computing*. April-June 2002. p. 22-31.

[13] Tandler, P., *The BEACH Application Model and Software Framework for Synchronous Collaboration in Ubiquitous Computing Environments*. Journal of Systems and Software, 2004. 69(3): p. 267-296.

[14] Dey, A.K. and Abowd, G.D., *Towards a better understanding of context and context-awareness*. June 1999, Georgia Institute of Technology.

[15] Gamma, E., Helm, R., Johnson, R., et al., *Design Patterns: Elements of Reusable Object-Oriented Software*. 1995, Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc.

[16] Gruber, T.R., *A translation approach to portable ontology specifications*, in *Knowledge Acquisition*. 1993. p. 199-220.

[17] Antoniou, G. and Harmelen, F., *Web Ontology Language: OWL*, in *International Handbooks on Information Systems*. 2004, Springer Verlag. p. 67-**92.**

[18] Bellifemine, F., Caire, G., TTrcco, T., et al., *JADE Programmer's Guide*. 2006.

[19] Reynolds, D., *Jena 2 Inference Support*. August, 2003.