Pervasive and Mobile Computing 9 (2013) 2–17

Contents lists available at SciVerse ScienceDirect

Pervasive and Mobile Computing

journal homepage: www.elsevier.com/locate/pmc

Review

Application mobility in pervasive computing: A survey

Ping Yu [a,*], Xiaoxing Ma [a], Jiannong Cao [b], Jian Lu [a]

[a] State Key Laboratory for Novel Software Technology, Department of Computer Science and Technology, Nanjing University, Nanjing, China
[b] Internet and Mobile Computing Laboratory, Department of Computing, Hong Kong Polytechnic University, Hung Hom, Kowloon, Hong Kong

ARTICLE INFO

Article history:
Received 6 January 2011
Received in revised form 30 June 2012
Accepted 25 July 2012
Available online 31 July 2012

Keywords:
Application mobility
Pervasive computing
Seamlessness
Context-awareness

ABSTRACT

Pervasive computing applications often need to maintain uninterrupted computing experiences when users move across devices. This advanced feature, recognized as application mobility, brings many challenges to the pervasive computing community. For a better understanding of the challenges and existing approaches to application mobility, this paper surveys related work with a classification and comparison framework established along four dimensions of design concerns in application migration: temporal, spatial, entity and other concerns. Through this survey this paper attempts to provide a systematic reference for developers to leverage off among different migration strategies for seamless application mobility. Moreover, it sheds some light on future work directions.

Contents

* Correspondence to: Nanjing University, Department of Computer Science and Technology, Hankou Road 22, Nanjing, Jiangsu, China. Tel.: +86 89680921; fax: +86 83593283.
E-mail addresses: yuping.nju@gmail.com, yuping@nju.edu.cn (P. Yu), xxm@nju.edu.cn (X. Ma), csjcao@comp.polyu.edu.hk (J. Cao), lj@nju.edu.cn (J. Lu).

1574-1192/$ – see front matter © 2012 Elsevier B.V. All rights reserved.
doi:10.1016/j.pmcj.2012.07.009

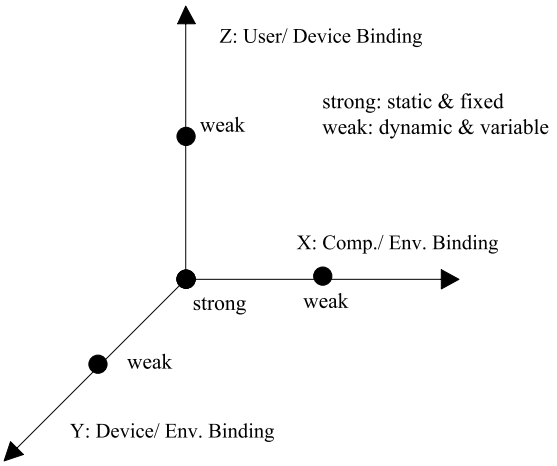

Fig. 1. User/device/environment binding framework.

1. Introduction

Mobility is not a novel feature initiated by the pervasive computing community. In the history of computing science, mobility was taken into consideration at the very beginning of timesharing systems in the early 1960s. At that stage, a user's mobility was supported by dumb terminals, which enabled users to log on the mainframe by any authorized stateless client-end device. In the late 1970s distributed computing introduced computation mobility (typically process migration) into operating systems, in order to distribute the work load or enable fault resilience. From the 1990s, thanks to the wide spread of mobile devices, such as laptops, PDAs and mobile phones, and wireless networking technologies including 2G/2.5G/3G/LTE and WiFi, etc., mobile computing has been penetrating into people's daily life, making in-motion communication, entertainment and Internet surfing feasible. Both academia and industry are making greater efforts into hardware and software research and development for perfecting mobile computing experiences.

When the era of pervasive or ubiquitous computing comes close, people look forward to a smart world where computation is invisibly weaved into daily life as envisioned by Weiser in 1991 [1]. Portable and smart devices are expected to realize the dream of "all time everywhere computing" [2]. The expectation of distraction-free computing also grows in view of the limited resources of human attention [3–5]. All these requirements stimulate a new pursuit of "seamless" application mobility, which refers to a *continuous or uninterrupted computing experience when the user moves across devices*. A seamless transition involves a potentially disruptive state change and may keep the user from distraction [6]. With high seamlessness, the user's experience would be smooth and unobtrusive when moving from one site to another. Seamless application mobility also addresses the integration with today's well-entrenched basis of personal computing applications and personal servers, which provide a variety of different, dynamic modes of operation and interaction depending on the environment and user's requirements [7]. Some existing and ongoing projects from mobile and pervasive computing groups have started to support application mobility in their systems. In this paper we analyze the critical challenges in supporting seamless application mobility and survey existing proposals with a classification and comparison framework.

The rest of this paper is organized as follows. Section 2 classifies different kinds of mobility in different computing scenarios. Section 3 elaborates the design concerns on how to support application mobility in pervasive computing environment, which results in a classification and comparison framework for related proposals. Section 4 reviews existing projects and compares their strategies in detail. Section 5 discusses future directions on application mobility, and the last section concludes this paper.

2. Mobility evolution

Mobility is the ability and willingness to move or change. In this paper, we classify mobility into three principal categories based on the relative association of a computation with a user, a computing device or a computing environment. The association is measured by a scale ranging from strong to weak. Hereby, "strong" stands for strong associations or static and fixed bindings, and "weak" represents weak associations or dynamic and variable bindings. The scenarios are illustrated in Fig. 1.

In Fig. 1, the *X* axis indicates *Computation to Environment* binding. The *Y* axis denotes *Device to Environment* binding and the *Z* axis *User to Device* binding. To support mobility, we should loosen the binding between each entity. The mobility pattern is evolving with the computing paradigm as shown in Table 1.

Table 1
Mobility pattern classification.

X	*Y*	*Z*	Computing paradigm	Mobility pattern
Weak	–	–	Distributed computing	Process, object, code
–	Weak	–	Mobile computing	IP, session
Weak	Weak	Weak	Pervasive computing	Seamless application mobility

First, as distributed computing developed new features such as process, object and code mobility, the association between a computation and its environment became broken. Client/Server architecture, Remote Evaluation (REV), Code on Demand (COD) facilities and world-wide Internet successfully provide people access to services or resources from remote servers. Code mobility helped to change the binding between code fragments and the location where they are executed [8]. Herewith, people can download or upload executable code or entities such as mobile agents from or to heterogeneous hosts, thus greatly loosening the "*Computation to Environment*" binding.

Mobility is an intrinsic property of mobile computing. It refers to the capability of a mobile terminal to change its point of attachment to the network and retain the original network identity, as well as communication sessions. It also implies the potential of mobile users to access services through different types of terminals and networks while in a state of motion. A series of cross-layer protocols are proposed to manage user's mobility in the wireless Internet infrastructure. Specifically, the network layer protocol "Mobile IP" [9] and the application layer protocol "SIP" (Session Initiation Protocol) [10] together aim at achieving transparent mobility to applications and higher level protocols. In this way, mobile computing allows users to carry their portable computing devices across different locations by relaxing the strong association between the device and the computing environment.

Mobile computing was closely followed by pervasive computing. In fact, the latter is the superset of the former, and it tries to establish a proactive support for computations in an "all the time everywhere" approach [2]. Pervasive computing emphasizes the idea of seamless integration across different platforms and masking all the heterogeneity of the underlying systems. Ideally, no computational or communicational gap should exist when users move in a pervasive computing world. Minimized user distraction and maximized resource utilization are expected in this new era paradigm. A typical scenario (Scenario I) is described as follows: Bob is watching a fantastic movie on his desktop computer at home. Soon it is time to leave for the office, but he wants to continue the video on his smart phone when he is in the subway. Seamless application mobility allows him to continue watching the movie from exactly where it was suspended. The video will be adapted to a suitable display size and tuned to the appropriate quality. This scenario implies that a user can transfer his computation task to any available device to continue from where the task is suspended, so as to make good use of various portable hardware resources.

Now, the story is going to Scenario II. When Bob is doing some coding in his office, his boss asks him to present the code design in the meeting room. Bob considers that it would be better if he can show the code and do some online modification during discussion. Now Bob leaves his office and steps into the meeting room. Amazingly, seamless application mobility enables his code editing workspace to be immediately recovered on a workstation at the meeting room and projected to a large screen. The migration process happens autonomously without Bob's intervention, benefiting from the smart space facilities such as locating sensors, camera and service discovery. This scenario implies that the application can as well autonomously migrate to the preferred device to continue computation, following its user. This kind of application was also recognized as a "*follow-me*" application [11,12]. Any traditional unmovable application such as a word processor or a media player can become a "*follow-me*" application if mobility is regarded as a first-class property and supported by the infrastructure.

Seamless application mobility in pervasive computing is promising and attractive, regarded as an important capability for mobile users. However, to bring it into play is more complex than the aforementioned mobility in distributed computing or mobile computing because it often needs to relax all the three bindings in the framework depicted in Fig. 1. Challenging issues arise with this fully loosely bound paradigm. Table 2 lists a set of properties, as a superset of those listed in [6]. These properties should be taken into account when proposing the solution to application mobility. A score of "high" is best for the attributes of seamlessness, ubiquity, smartness, heterogeneity, network resilience and solution generality. A score of "low" is ideal for the attributes of network load and implementation complexity.

3. Design concerns

Mobility in pervasive computing breaks down all fixed and static bindings between the user, device, application and environment. Though lots of proposals have provided some mobility supports or migration services, application mobility in pervasive computing is still a fresh and hot issue which needs further investigation from a software engineering perspective. Fig. 2 presents a conceptual framework structured from four dimensions: temporal (when), spatial (where), entity (what) axes and an open axis for other design concerns.

3.1. When

A general migration rule goes like this: when a user is leaving, his/her current session is suspended; after the user switches to another device, the previous session is resumed in the new execution environment. Even in this simple case, there are at

Table 2
Properties of seamless application mobility.

Property	Description	Best score
Seamlessness	User feels smooth and unobtrusive when application moves across devices.	High
Ubiquity	The solution can be easily deployed in ubiquitous computing devices.	High
Heterogeneity	The solution should hide platform heterogeneity as much as possible from users for the sake of low distraction, but also expose as many changes as possible to the application for agile adaption.	High
Smartness	Context-awareness and adaptation bring forth smartness to pervasive computing applications. It implies intelligence of computing, with the expectation of accurately comprehending user's intention, monitoring physical world status and taking appropriate reactive actions.	High
Network resilience	The latency of migration should be reduced even if the network quality is poor. The migration process should not depend much on continuous, stable connection or broadband connection speeds.	High
Network load	The data transferred should not be too large. The migration process should not consume too much transient bandwidth.	Low
Generality	It will be ideal if the solution is general to all kinds of existing applications without modification or recompilation.	High
Complexity	The solution should be easily implemented.	Low

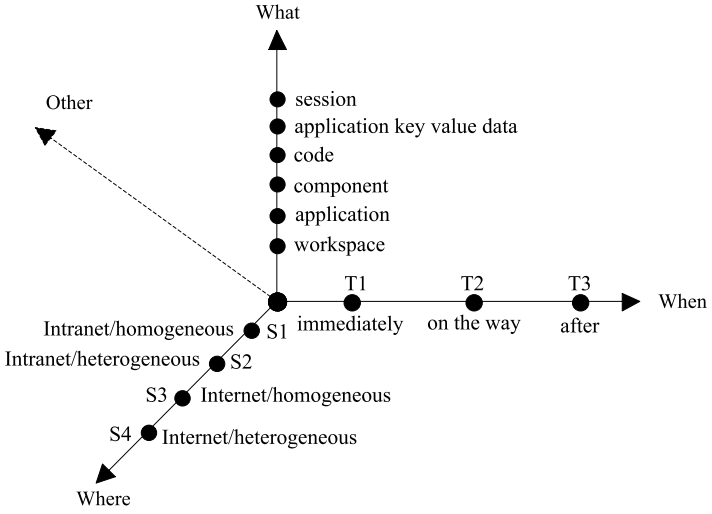

Fig. 2. Design concerns of application mobility.

least three choices for when to migrate: (1) migrate after the user switches to the new environment; (2) migrate when the user is on the way; (3) migrate immediately when the user is leaving. Since the migration process will take some time, the user would have to wait before all the data were ready if the first choice were used. The delay could be unacceptably long when the network bandwidth is limited. The other two choices can reduce the delay by moving in advance. In an ideal case, the user can continue his/her work as soon as he/she arrives at the new site.

According to the three migration choices, two implementation strategies are available: *reactive mode* and *proactive mode*. Both the underlying system and upper layer services for pervasive computing should be intrinsically context-aware in view of the ever changing environment and user activities. In other words, context changes drive systems and services to adapt in order to serve users better. Most proposals for context-aware computing adopt the reactive mode in which adaptation will take place when context changes. Specifically, reactive mobility can be triggered by a rule-based inference engine that decides whether or not to migrate, with the help of a locating system for tracking users. Although in most reactive systems adaptations are triggered immediately with the well-known event/condition/action (ECA) pattern, migration is not always timely enough because of the unavoidable network latency. In contrast, the proactive mode shows its value by acting before the situation changes. However, to achieve proactivity is more complex than to enable reactivity, on account of the complexities from user behavior prediction and user intention perception. Unless carefully designed, proactivity may disturb users and effectively defeat the goal of invisibility [3]. Therefore, a reactive system with some carefully controlled proactivity is preferred for pervasive computing.

3.2. Where

The migration destination will be assigned an Internet address in Scenario I or an Intranet portal in Scenario II in the above section. The application will resume execution on a heterogeneous device (from a desktop computer to a smart phone) in Scenario I or a homogeneous device (from a desktop computer to a workstation) in Scenario II. Some problems arise here: (1) addressing; (2) resource rebinding; (3) portability.

Firstly, the migration destination depends on the user's mobility. When synchronizing physical mobility with logical mobility under pervasive computing circumstance, a *discovery service* for looking up involved physical devices and logical services is indispensable. It should be able to discover, retrieve and bind service components dynamically and autonomously in a foreign situation without user intervention. Though some service discovery protocols such as Bluetooth [13], SLP [14], Salutation [15], and Surrogate [16] are available, how to enable services of different protocols to work together is a problem worthy of study.

Another issue accompanying computation migration is resource rebinding, which refers to dynamically reconnecting with resources at the source site and binding to available resources at the destination site. As a basic rule, the more the data transferred and localized to the new site, the less the computation's dependence on the previous site. Fuggetta et al. classified resources as "freely transferable", "fixed transferable", and "fixed not transferable" according to their properties [8]. They propose three data management strategies of "*by move*", "*by copy*" and "*rebinding*" for transferring resources of different properties. The first two strategies get resources ready in the new site before the computation continues. The connection can be intermittent after successfully moving or copying all local transferable resources required. However, to have more than one copy of data introduces another problem of data consistency. The last strategy requires a new binding by network reference. Some resources can be left in the previous site. After migration, the computation will continue with remote references to resources on demand. By late binding, it can flexibly refer to neighboring or local services and resources rather than connecting to the original ones. No matter which strategy is adopted, the network must be reliable during the period of transferring resources or when remotely referring to resources. However, wireless network is more popular in a pervasive computing environment, where connection quality cannot be guaranteed as in wired network. Hence, it is also necessary to allow disconnected operation for high network resilience. Moreover, the conventional method of mixing a resource binding strategy with service logic is not suitable to pervasive computing applications because a resource binding strategy might be dynamically changed or adapted according to various execution conditions.

Heterogeneity is the biggest obstacle when making application migrate seamlessly. *Platform independence* is partly achieved by running applications on top of the same system or process virtual machine (VM). However, in pervasive computing environments, platforms span from heavyweight operating systems such as UNIX, Linux, Mac OS and Windows, to lightweight embedded, mobile systems such as Palm, Symbian, iOS, Windows Phone and Google Android. There is no system virtual machine as yet which can hide such kind of heterogeneities. A process virtual machine makes up for this limitation in virtue of a specific *portable language*, such as Java. However, there are different JVM versions for devices of diverse capabilities. To be specific, a Java application is not always portable among those Java enabled devices, so mobile logical entities should be carefully planned in view of heterogeneity of VM once again.

3.3. What

Logical mobility melts down the binding between the computation and execution environment. Previous researches in distributed systems have demonstrated some benefits from this loose binding, such as enabling movement toward a desired resource, the use of computer resources while moving, and improved flexibility [17]. In particular, *process migration* supported by a few operating systems aims at moving an executing process from one machine to another, for load balancing and fault resilience. *Object migration* makes it possible to move objects among different address spaces. It implements a more fine-grained mobility than process level migration. However, process migration and object migration were originally designed for small-scale and homogeneous local area networks. With the development of large scale networking facilities such as the Internet, *mobile code* offers more benefits of service customization, autonomy, data management flexibility, protocol encapsulation and so on. Code mobility is assumed to operate in large-scale and heterogeneous networks, which are managed by different authorities and connected by links with different bandwidth [8]. On the whole, code can be downloaded from remote server to the local host, providing services locally. Customized code can be uploaded to a remote server, executing tasks or accessing services located there and getting result back. Various design paradigms, such as COD, REV and MA (mobile agent), can be used to enable code mobility.

Practically, the mobility supports from distributed computing are helpful in providing underlying migration services, but inadequate for the seamless application mobility. Considering seamless application mobility in a pervasive computing environment, movable logical entities will vary from a web or a desktop session, application key value data, some code segments and a component or an application, to a complete workspace such as the guest operating system, as marked in Fig. 2. Fig. 3 illustrates the typical size of various movable entities. The smaller the movable entity size is, the finer the mobility granularity is. A file is excluded from Figs. 2 and 3 due to frequent file transfers in most cases, while the file's size can be very small (e.g. several kilobytes accompanying code segment mobility) or very large (e.g. tens of gigabytes accompanying VM migration). Generally speaking, monotonous mobility granularity is not suitable for various pervasive computing scenarios, due to the heterogeneity of physical environments. For example, moving a complete Microsoft Windows "desktop" from a laptop to a PDA would not be efficient or affordable, even if it were possible. Basically, fine-grained mobility involves specific code segments, object and component with runtime data space and execution session, in a most economical way. On the contrary, more attention is paid to coarse-grained mobility by moving the entire working environment and addressing the maximum user workspace migration. Ideally, mobility for pervasive computing should be achieved by minimizing resource consumption and perfecting a seamless user experience as well. A trade-off between fine-grained and coarse-grained mobility should be delicately considered, in terms of environment unevenness.

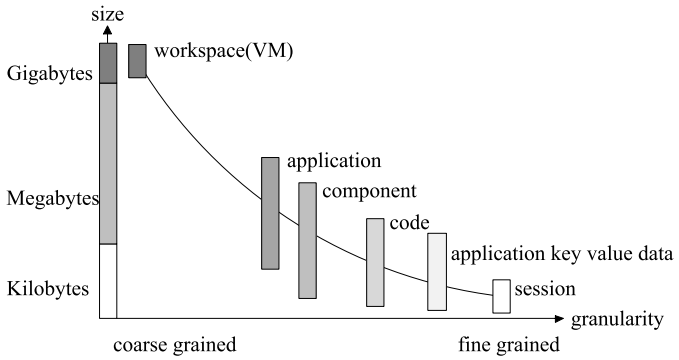

Fig. 3. Size of different movable entities.

3.4. Other concerns

Besides the problems of when, where and what to migrate in a pervasive computing environment, some concerns are also significant to design infrastructure for application mobility. In particular, we discuss the architecture style of the client side and the server side. Thin clients and thick clients are two opposite patterns for the client side design, while middleware is a popular solution for the application server.

- Thin client vs. Thick client.

Arising from client/server architecture networks long time ago, a *thin client* usually has little or no application logic and relies strongly on the remote server for carrying out tasks. A thin client simplifies seamless mobility for pervasive computing, where devices can be regarded as portals into application/data space rather than repositories for custom software [18]. Since both the application logic and user's personal states/preferences are stored in the back-end server, a mere reconnection is required to continue computation after the user switches to another device. That is, by minimizing the terminal dependence the user can access the same services in a similar way from anywhere. Supposing service states are all held by the remote server, seamless workspace migration is easily achieved with the least effort. For example, users' Gmail opened in a web browser always looks the same to them in their favorite manner regardless of the time and place. In fact, most of the web-based applications fall into this category. However, while reducing client side capabilities, network connectivity and bandwidth become the critical bottleneck. Traditional thin client tools such as "remote desktop" require a continuous connection to the remote server, but latency or fragile connectivity often disappoints users.

In another extreme, a *thick or fat client* prefers to have as many services as possible in a single device, which can be regarded as a personal server. Supposing that a mobile user carries a portable device acting as a thick client, the work in progress such as editing, calculating and web browsing can be continued when the user arrives at a different place. In this case, limited battery capacity is the greatest obstacle. Moreover, excessive dependence on a specific device will result in tremendous loss if the device is broken or stolen. The goal of the user being distraction-free is hardly achieved in the thick client scenario because the user has to carry the mobile device with him/her in the state of motion.

From our point of view, the thin client pattern does not loosen the binding between the computation and the device, whilst the thick client pattern still relies on the fixed association between the user and the device. In other words, the advantages of pervasive computing where computation resources are abundant and ubiquitously available could not show effectiveness. A trade-off between these two extreme patterns must be investigated.

- Middleware solution vs. Other tools.

Most current proposals for physical and logical mobility resort to the technique of *middleware* which resides between the operating system and the application layer. Middleware is a class of software technologies designed to support application development by enhancing the level of abstractions associated with the programming effort [19]. In general, middleware provides basic services for upper layer applications by providing programming interfaces and taking charge of management functions such as caching, fault tolerance and security. With an eye to middleware for pervasive computing, especially for application mobility, *mobility support* is the most important part of the middleware. The efforts include introducing atomic actions for distribution and migration in the APIs opened to applications, providing tools for administrating migration, such as code and state persistence, data management, and consistency checking.

In addition to middleware solutions, there are still some other methods for application mobility. Tools such as "remote desktop", "system virtual machine" belong to this category. These tools are dependent on specific operating systems and protocols, and pose little or no disturbance on existing applications. However, it is difficult to tailor and customize them according to a broad scope of requirements and situations. We will discuss these different approaches respectively in the next section.

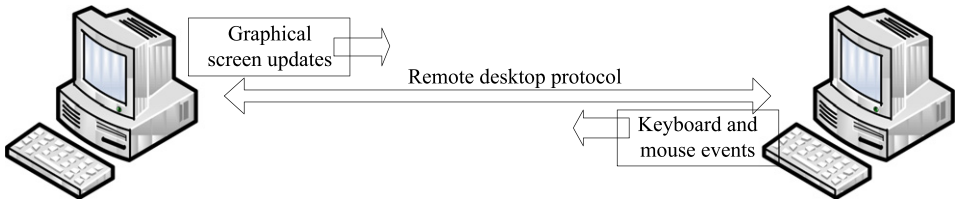

Fig. 4. Input/output mobility paradigm.

4. State of the art

In general, lots of approaches have been proposed to enable seamless application mobility in pervasive computing environments. In this section, we will survey several representative proposals according to the design concerns and strategies mentioned above. In Section 4.1, we briefly review some traditional approaches to user's "desktop" mobility. Section 4.2 discusses some system virtual machine techniques targeting at maintaining a continuous computing workspace for a user everywhere. In Section 4.3, we focus on the pervasive computing middleware by a couple of topics such as the migration paradigm, the application model and state consistency. Section 4.4 refers to approaches under the novel computing metaphor of cloud computing. Section 4.5 gives a comparison framework for all mentioned proposals and derives a guideline for choosing a most appropriate mobility strategy.

4.1. Remote desktop

Though the "Remote Desktop Protocol" (RDP) for Windows [20], "Virtual Network Computing" (VNC) [21] and "OSXvnc" for Mac [22] were not initiated from a pervasive computing society, they actually bring the seamless experience of accessing the whole workspace of one computer from another. The applications, files and network resources, are shared by the users wherever they are. Through a remote desktop connection, a user can continue his/her work left in the office when he/she goes back home. Taking the example of "Microsoft Remote Desktop Connection", it allows for separate virtual channels for carrying presentation data, serial device communication and license information from the server side, as well as encrypted client mouse and keyboard data [20]. As Fig. 4 shows, only the keyboard, mouse and display data are transferred. In other words, it is a type of *input/output mobility*. The client side acts as a controller to manage applications and files on the remote server; and the execution state and data are still kept in the server side over time, with no application adaptation requirements for running applications. However, the resources (e.g. file system, network printer, audio, and communication port) should be transparently redirected to the client side environment. For example, the music player is started in the remote server, while the sound is sent out from the local speaker.

In general, "remote desktop" adopts a client/server architecture where the client side is similar to a graphical user interface controller of a server session. It requires users to specify the accurate server access portal of IP address and port number after users launch the client side remote desktop service. With a successful connection established, a complete set of screen data is transferred. No finer grained migration can be done. Although most remote desktop tools claim that a 56K modem connection suffices, they could not work well under the condition of unstable and fragile mobile network connections, since a 5–6 screen refresh per second is always required during the connection.

4.2. System virtual machine

In a broad sense, a remote desktop service such as RDP and VNC is also a type of virtualization technique, i.e., desktop virtualization. Now we will focus on another big virtualization family – system virtual machine or virtual workstation.

Internet Suspend/Resume (ISR) is such an approach to mobile and pervasive computing in which a user's computing environment follows the user through the Internet he/she travels [6,23]. Learning from the suspend/resume mode in laptops, ISR aims at the same effect of resuming working in the same environment that was active at the time of suspension, without requiring the user to transport the physical device. The hands-free property of ISR makes it suitable for pervasive computing environments where computing devices may be widely deployed for transient use. ISR is implemented with the combination of two essential techniques: virtual machine and distributed file systems. Here, the virtual machine is used to capture the user's computing environment state at the time of suspension. This environment consists of a guest operating system (OS), guest applications and all data files, customizations and the current execution state. As Fig. 5 shows, the state can be stored into a distributed file system (i.e. Coda) or even exported into a portable storage device. By leveraging virtual machines, this state is re-instantiated or restored directly on the host platform at the new site.

Unlike the remote desktop service, no connection to the original site is required after resumption, leading to the higher network resilience of ISR. All applications will continue execution by consuming the current, available computing resources (e.g. CPU, memory, hard disc) at the new place, thereby implying that the client side is not so "thin". ISR prefers manual migration rather than context-driven movement. Proactivity is embodied in the sense of caching the data ahead of time or on the way. Adaptation is rarely required thanks to the virtual machine which hides heterogeneity of platforms. However,

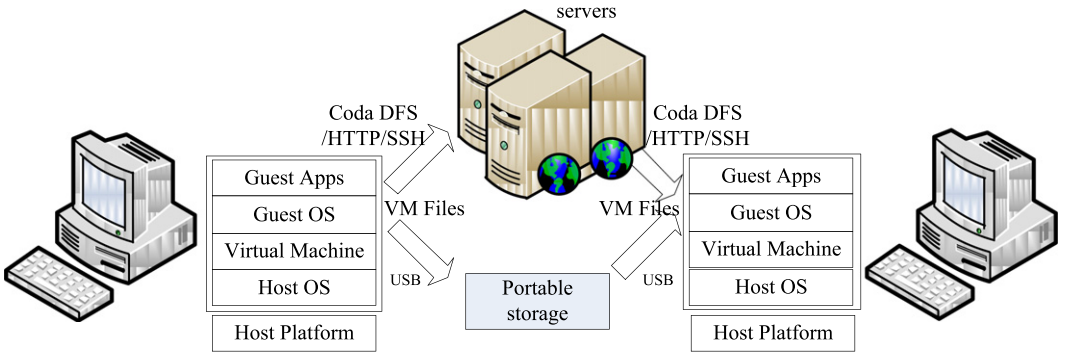

Fig. 5. System virtual machine migration architecture for workspace mobility.

Fig. 6. Middleware for application mobility.

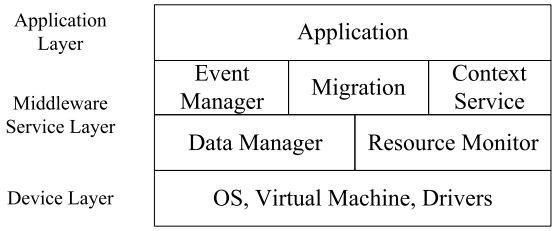

Fig. 7. General middleware structure.

this approach is challenged by transferring a complete set of environment states which can be very large, often amounting to tens of gigabytes. Moving such a bulk of data through the network is hardly an efficient job, especially in a wide area network or a wireless network. Recently the ISR group developed a Pocket ISR client, which can be installed on a USB flash drive and booted on any PC to provide a complete ISR client quickly without any modification to the PC [24]. A trusted personal assistant called Horatio for smart phone users is also developed to serve as a self-cleaning portable cache of the VM state [25]. SoulPads provides another approach by carrying an auto-configuring operating system along with a suspended virtual machine on a small, portable device [26]. It does not require any pre-installed software as ISR does, but the resume time increases due to Host OS auto-configuration.

Some other virtual machine/workstation projects such as VMWare's VMotion technology [27], Stanford University's "Computer Capsule" [28] and the Collective system [29] satisfy the pervasive computing requirement of user computation and experience continuity in the state of motion as well. The gap between different execution environments is smoothed by hardware virtual machines and the operating system virtualization layer respectively. The three bindings shown in Fig. 1 are loosened or partially loosened, as expected by the pervasive computing society. However, these approaches still involve a lot of user intervention due to lack of smartness. Moreover, most of these approaches adopt a monotonous granularity of movable entities, incapable of tailoring and adaptation according to various user requirements and environment conditions. These reasons lead to the appearance of numerous middleware, which endeavor to build smart spaces and enable finer grained application mobility.

4.3. Middleware

A lot of middleware systems for programming smart space appeared in the last decade. Fig. 6 describes that a middleware runs on top of process virtual machine and establishes a channel for applications to move across devices. By studying some research projects focusing on application mobility and seamless integration of various resources, we summarize some fundamental services and draw a general structure as Fig. 7 shows.

The bottom is the *Device Layer*, which includes the operating system, all device drivers and some language based process virtual machine (e.g. Java's JVM and Microsoft's Common Language Runtime). The top layer is the *Application Layer* where applications are implemented in a programming framework as specified by the middleware. The *Middleware Service Layer* consists of some fundamental services for enabling application mobility. The *Data manager* is critical for transferring different kinds of resources across devices. The *Resource monitor* is responsible for device or service discovery and application-aware adaptation according to resource changes. Service and resource discovery facilities such as Jini [30] and UPnP [31] bring "anywhere" computing into practice by dynamically detecting new, available services and binding them to the original application. *Migration* is the pivotal service which targets moving computation entities between different devices while keeping state or session consistency after migration. *Context service* includes sensing, processing and reasoning context from users and computing environments. *Event manager* is in charge of broadcasting context change events to other services and enabling context-awareness. A rule inference engine is often embedded in the event or context manager for filtering events and triggering context-aware adaptation. The following paragraph will discuss these aspects by a review of some middleware projects.

- Aura.

As a pioneer of projects in pervasive computing area, CMU's Aura aims at minimizing user distraction by creating a smart environment that adapts to user's context and needs [4,32]. Aura introduced a task-oriented concept to application programming framework. A high level system abstraction layer – *Task layer* is proposed to represent user intent explicitly. The task layer is available to the rest of the system as a powerful basis on which to adapt or anticipate user needs. Aura is established based on Coda and Odyssey. Coda permits nomadic, disconnectable and bandwidth-adaptive file access, and Odyssey supports resource monitoring and application-aware adaptation. Seamless application mobility is achieved by using two application abstractions: *Suppliers* and *Connectors*. Suppliers provide the data abstraction (like text, video, etc.), and connectors provide the application interfaces to the data, such as Microsoft Word (for word processing), Winamp (for media playing), etc. Actually, only the user's task of what he/she is doing is required to move. When a user moves to a different location, the *Context Observer* reports the event immediately to the *Task manager* – Prism. Then, Prism requests the *Environment manager* for the allocation of a new application supplier based on the current task and context information. By the allocation of a new supplier, the user can carry on with his task in the new environment.

With the middleware support from task/context/environment managers, proactivity is conceivable, and user distraction is lowered down in need. Moreover, it no longer needs to connect with the previous site after task migration. Platform independence is partially achieved by installing the middleware to hide heterogeneity of operating systems. Aura avoids the difficulty of the application process migration and adaptation by choosing a most suitable application supplier after context changes. Thus, it can ensure a user's task to be continued after migration. Since the application supplier will change, most of the task's initial execution states would be discarded.

- Gaia.

Gaia focuses on the problem of migrating applications across different ubiquitous environments, which are characterized as *Active Space* [5,33,34]. Gaia's active space programming framework enables an application to adapt its structure by decomposing an application into a design pattern of smaller components: model, presentation, controller, input sensor and coordinator, which is compatible with the MVC [35] pattern. During the process of application migration, a snapshot of the current application is stored in the *Application Customized Description* (ACD) file which contains not only the application high level state but also the application structure description. After arriving at another location, the user can resume the previous tasks by manually selecting the ACD to load or the system can automatically restore the application from the ACD through the *Context File System*. In Gaia, application adaptation includes dynamically changing component type, cardinality and device mapping according to the new environment. After migration, controller and presentation components can be replaced by semantically similar components if the same components are not available. Device mapping is also dynamically configured based on system rules and user defined rules in a context-aware manner. Computational reflection technology is exploited during reconfiguration progress.

Similar to task migration in Aura, ACD is required to migrate for recovering a user's tasks. However, due to the decomposable application component model, Gaia addresses finer grained migration and adaptation than Aura. Gaia allows two kinds of *component mobility*, i.e., intra-space mobility and inter-space mobility. Intra-space mobility allows moving the interactive components of an application (i.e., presentations and input sensors) to different devices present in an active space. Inter-space mobility allows all components to move to different active spaces. For intra-space mobility, the moved presentation component synchronizes itself with the application by contacting the original model component where the application states are stored. For inter-space mobility, not only the application base-level state (managed by the model) but also the application meta-level state (managed by the coordinator) are made persistent in the *Context File System* for application reconfiguration and recovery.

- One.world.

As an integrated and comprehensive framework for building pervasive applications, One.world supports application mobility depending on a combination of process migration and language-based mobility [36,37]. It uses a process virtual machine environment to provide uniformity across heterogeneous platforms, and it exploits object serialization to convert

between virtual machine objects and byte streams. One.world introduces the concept of "nested environment" for structuring and composing applications. It organizes applications in nested environments, which contain all the data of an application, both runtime state and persistent storage. A "tuple" based common data model is defined for applications to share data. Application migration consists of capturing (or checkpointing) current application environment and storing the captured state as a tuple. The tuple and the application's environment tree are then converted into a byte-stream and are transmitted to the new host. The application state is de-serialized there from the byte-stream of the checkpoint tuple. After its state is restored in the new host, the application is notified about that, and it then tries to adapt to the changed execution context. One.world provides a set of services that help developers make applications movable and adaptable: migration APIs, remote event passing, discovery and checkpointing services.

Different from Aura and Gaia, One.world follows the *object mobility* paradigm. It simplifies application migration by the standard process of object marshaling and unmarshalling, in virtue of language portability. It provides a unified I/O interface to storage and networking, i.e., reading and writing tuples across the network. Consequently, applications can freely exchange tuples and events between each other and migrate from one environment to another.

- MDAgent.

MDAgent is another middleware to support application mobility in pervasive environments. It adopts a mobile agent based approach to enable application and components to move with a user across devices [11,38]. Mobile agent is known as a paradigm for *code mobility* with the advantages of reducing network load by accessing resources locally, executing tasks asynchronously and autonomously, and being hardware and transport-layer independent [39]. In MDAgent, a traditional unmovable application is wrapped by a mobile agent that acts as a controller of the application, i.e., suspending, migrating and resuming the application. Aided by the underlying middleware services – *context manager*, *rule engine* and *event mechanism*, mobile agent-enabled applications can be made sensitive to the user's context, including location, activity, preference, and the profile of the device being used. Each mobile agent is attached with a rule set for runtime adaptation when encountering a changed context. A reflective application model is proposed for runtime resource rebinding and component reconfiguration. In this model, application-dependent resources and components are required to be explicitly described in RDF format as application meta-data. Since some resources can move with the application, such as the document currently under editing or the music file being played, while other resources like the database cannot be moved, different rebinding strategies like "by copy" or "by reference" should be provided for these resources. The agent can manage resources with more accurate mobility control by analyzing the meta-data. Moreover, it is possible to proactively move an application to the destination because an agent could turn into an autonomous entity with some intelligence.

- Roam.

Roam is an application framework for developers to build multi-platform applications and move a running application seamlessly to heterogeneous devices [40]. In Roam, an application is modeled as a *Roamlet* which can migrate between any two connected devices that have the Roam middleware running. The architecture contains three main components: *Roam Agent*, *Roamlet* and *HTTP Server*. The *Roam Agent* on the source device first negotiates with the *Roam Agent* on the target device to exchange necessary information for migration. The *Roam Agent* on the target device downloads *Roamlet* classes from the *HTTP Server* for all application components that will be instantiated on the target device. The *Roamlet* on the source device serializes its execution state and sends it to the target Roam Agent, who instantiates the downloaded *Roamlet* and recovers the application. When encountering a heterogeneous target device (e.g., from PC to PDA), the Roam Agent may transform execution state with a SGUI toolkit. The Roamlet is similar to the Java applet which follows a *code on demand mobility* paradigm.

- FollowMe.

Li et al., designed and implemented an *OSGi* based pluggable context-aware middleware which provides infrastructural support for "*follow-me*" applications [12,41]. OSGi is a standardized, component oriented computing environment for networked services [42]. It is also accepted by many research groups from academia. SOCAM [43] and "Gator Tech Smart House" [44] are such kinds of pervasive computing middleware built on the OSGi framework. In addition, the OSGi framework is essentially a module system and service platform where applications can be dynamically installed, started, stopped, updated and uninstalled. It provides a good testbed for practicing application mobility. Under FollowMe infrastructure, a user's task is abstracted into a workflow, which is a declarative model used to describe a sequence of operations. The "follow-me" mobility is implemented with a workflow engine service which can interpret and manage workflow across devices. The same operation may be implemented in different ways on different hardware platforms, possibly using entirely different service bundles. What is preserved and migrated is the notion of workflow progress, much like the *task mobility* in Aura.

- Other proposals.

Besides the proposals mentioned above, a lot more work is done or being done on application mobility in pervasive computing environments. For example, Francis et al., proposed another agent based mobile application framework which provides the opportunity for applications to better adapt their interface to the new environment [45]. Different from MDAgent which advanced a novel application model, they focused on the classic MVC pattern to enable GUI decoupling

Table 3
Summary of middleware proposals.

Proposal	Mobility paradigm	Application model	State consistency	Complexity of adaptation
Aura	Task mobility	Task oriented + Supplier/Connector abstraction	Task level (weak)	Low (change of the supplier)
Gaia	Component mobility (partial and full)	Active space + extensive MVC design pattern	Application snapshot level (medium)	Medium (component reconfiguration)
One.world	Object mobility	Tuple-based data + nested environment structure	Process level (strong)	High (environment reconfiguration)
MDAgent	Code mobility (mobile agent)	Mobile agent based reflective model	Process level (strong)	High (resource rebinding)
Roam	Code mobility (code on demand)	Roamlet + Roam Agent	Application snapshot level (medium)	Medium (component and state transformation)
FollowMe	Task mobility	OSGi based + Workflow	Task level (weak)	Low (change of service bundle)
Francis' framework	Code mobility (mobile agent)	Mobile agent + MVC design pattern	Process level (strong)	High (resource rebinding)
A2M	Code mobility (code on demand)	*Not mentioned*	Application snapshot level (medium)	Medium (GUI adaptation)
ScudOSGi	Task mobility	OSGi based + Whiteboard model	Task level (weak)	Low (change of service bundle)

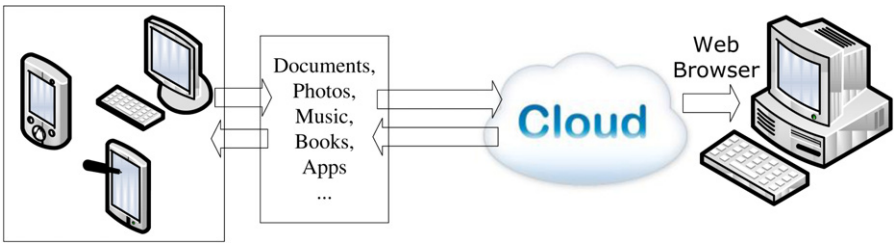

Fig. 8. Cloud computing paradigm for user content mobility.

and rebuilding. Ahlund et al., also proposed a context-aware and decentralized architecture named "A2M" for application mobility [46]. Based on "A2M" architecture, a "Mobile YouTube Player" capable of moving between heterogeneous devices was implemented to provide a seamless video playing experience. Xu et al., proposed a task migration mechanism in the OSGi Framework named ScudOSGi which focuses on context-driven migration and local facilities rebinding [47].

In Table 3, these proposals are summarized from four significant aspects: mobility paradigm, application model, state consistency and complexity of adaptation. Mobility paradigms are classified as task, component, object and code mobility, according to the entity to be moved. Each paradigm is related to a specific application model as listed below. From the last two columns of Table 3, we conclude that the lower the complexity of adaptation is, the weaker the consistency of execution states would be. As far as task mobility is concerned, only the user's task and the task's high level state are moved to the new environment to continue by selecting a new application supplier, while the detailed task execution states on the source site are often ignored. As for object and mobile agent mobility, it is possible to reserve and recover all execution states by marshaling and unmarshalling. However, the requirements of environment or resource rebinding would result in the high complexity of adaptation. Component and code-on-demand mobility involves component/code migration and reconfiguration, as well as ensures that an application snapshot should be transferred to the new environment for application resumption.

4.4. Cloud services

With the novel computing metaphor of *cloud computing*, software systems are becoming increasingly mobile and ubiquitous. As Fig. 8 shows, cloud computing allows service clients to upload or download shared resources, software and information on demand with a secure connection to the cloud. Typical cloud computing providers deliver applications or services online that are accessed from another Web service or software like a Web browser, while the software and data are stored on servers. The philosophy of cloud computing also accords with the goal of application mobility that is to make best use of more powerful and easily accessible computing resources. As an Internet-based computing paradigm, cloud computing allows clients to have a location and machine-independent view of applications. It is possible to save the session of a user's application in the cloud, restore it after the user logs in again no matter where he/she is and adapt its facade automatically according to the device he/she is using, and consequently makes the application look like it is "following" the user.

Rooted in Apple's ecosystem, the *iCloud* [48] service automatically synchronizes a user's contents, such as documents, photos, music, email, calendar and applications, among all of the user's Apple devices: Mac, iPad, iPod and iPhone. If no Apple device is available, the user can fetch some of his/her contents directly from iCloud.com with a Web browser and

Table 4
Comparison of existing proposals (I).

Proposal	Design concerns								Strategies			
	When			Where				What	Reactivity	Connectivity	Client thickness	Independence
	T1	T2	T3	S1	S2	S3	S4					
RDP			●	●	○	○	○	Input/output	Manual	Connected	Thin	OS dependent
VNC			●	●	●	○	○	Input/output	Manual	Connected	Thin	OS independent
ISR	○	○	●	●	○	●	○	Workspace	Manual	Disconnected	Thick	VM dependent
VMotion	○	○	●	●	○	●	○	Workspace	Manual	Disconnected	Thick	VM dependent
SoulPads			●	●	●	●	●	Workspace	Manual	Disconnected	Thin	VM dependent
Aura	●	●	●	●	●	●	●	Task	Context-aware	Disconnected	Medium	Middleware dependent
Gaia			●	●	●	●	●	Component	Context-aware	Connected or disconnected	Medium	Middleware dependent
One.world			●	●	○	●	○	Object	Context-aware	Disconnected	Medium	Middleware dependent
MDAgent	○	○	●	●	●	●	●	Agent	Context-aware	Disconnected	Medium	Middleware dependent
Roam			●	●	●	●	●	Code	Context-aware	Disconnected	Medium	Middleware dependent
FollowMe			●	●	●	●	●	Task	Context-aware	Disconnected	Medium	Middleware dependent
Francis' framework			●	●	●	●	●	Agent	Context-aware	Disconnected	Medium	Middleware dependent
A2M			●	●	●	●	●	Code	Context-aware	Disconnected	Medium	Middleware dependent
ScudOSGi			●	●	●	●	●	Task	Context-aware	Disconnected	Medium	Middleware dependent
iCloud	●	●	●	●	●	●	●	Content	Automatic	Disconnected	Thin	Apple's ecosystem dependent

his/her iCloud account. Moreover, iCloud storage APIs are open to developers who can store documents and application key value data in the cloud [49]. It competes with other cloud storage services, such as *Dropbox, Microsoft SkyDrive* and *Google Drive*, that all support file storage in the cloud and file synchronization across devices. Compared to the application mobility discussed above, these cloud services loosen the binding between the user and the devices via powerful cloud providers who automatically collect, synchronize and distribute the user's contents. An application does not truly move, but the user's settings/preferences and the application pivotal state information are kept up to date in different iCloud enabled devices when they are connected to the cloud. It is a way for the user's application to share exceedingly small amounts of data (tens of kilobytes) with other instances of itself.

4.5. Comparison framework

Based on the design concerns and strategies discussed in Section 3, a comparison framework of the typical proposals above is given in Table 4 (T1,T2,T3,S1,S2,S3,S4 are marked in Fig. 2). The option that a proposal chooses is marked as "●". The option that a proposal is probably supported by some extension or under specific conditions is marked as "○". The first two approaches prefer a thin client strategy, relying much on user interface remote controlling. The "ISR" and "VMotion" approaches adopt a thick client strategy, in virtue of system virtual machine techniques. "SoulPads" slims the client because it moves the heavy virtual machine to a portable storage. They all address the monolithic workspace migration. They usually have little support on migration smartness and still involve a lot of user intervention due to the lack of context-awareness. The other approaches except "iCloud" all target at establishing "smart" middleware to hide device and underlying platform heterogeneity. The pre-requirement of middleware installation balances client side overload between the thinnest and the thickest. Context services embedded in these middleware provide some smartness for application migration. The "iCloud" service can automatically push content to user's iCloud enabled devices. It depends on Apple's ecosystem because an iCloud account can only be created from an iPhone, iPad or iPod touch with iOS 5, or a Mac with OS X Lion v10.7.2 or later. All of these solutions, except "RDP" and "VNC", do not require a continuous connection with the previous site after the user moves to a new site or switches to a new device, trying to make better use of currently available resources.

Another critical comparison framework of existing proposals is described in Table 5, based on the knowledge from the above analysis. We evaluate them along with the properties listed in Table 2. Unfortunately, no proposal has arrived at the ideal status where each attribute gets the best score. The first five proposals achieve high seamlessness because the source and target environments are indistinguishable except for possible minor differences in the display and keyboard/mouse. They also score high on ubiquity and generality since they are easy to be deployed and require no modification to the upper layer applications. They are weak in smartness due to lack of context-awareness modules. Network resilience depends much on the requirement of connection between source site and target site after migration. The first two proposals score low because a continuous connection should be always maintained. The "ISR" and "VMotion" proposals suffer from high network load because they have to move a large volume of data between machines, whilst they earn a top score on network resilience because the target site no longer depends on the source site after migration. "SoulPads" provides an offline solution and does not consume any network bandwidth. Heterogeneity refers to the difference between computing environments. All proposals support running on heterogeneous operating systems or devices because of the existence of a virtualization layer or middleware. "RDP" gets a low score because the heterogeneity it supports is limited. All middleware proposals score low in generality because applications running on them must follow a particular programming model. They all score medium on the ubiquity attribute because they rely on deploying the middleware in the environment where some preconditions

Table 5
Comparison of existing proposals (II).

Proposal	Seamlessness	Ubiquity	Smartness	Heterogeneity	Network resilience	Network load	Solution generality	Implementation complexity
RDP	High	High	Low	Low	Low	Low	High	Low
VNC	High	High	Low	High	Low	Low	High	Low
ISR	High	High	Low	Medium	High	High	High	High
VMotion	High	High	Low	Medium	High	High	High	High
SoulPads	High	High	Low	High	High	None	High	High
Aura	Low	Medium	Medium	High	High	Low	Low	Medium
Gaia	Medium	Medium	Medium	High	Medium	Medium	Low	Medium
One.world	High	Medium	Low	Medium	High	Medium	Low	Medium
MDAgent	High	Medium	Medium	High	High	Medium	Low	Medium
Roam	Medium	Medium	Low	High	High	Low	Low	Medium
FollowMe	Low	Medium	Medium	High	High	Low	Low	Medium
Francis' framework	High	Medium	Low	High	High	Medium	Low	Medium
A2M	Medium	Medium	Low	High	High	Low	Low	Medium
ScudOSGi	Low	Medium	Medium	High	High	Low	Low	Medium
iCloud	High	High	Medium	Medium	High	Low	Medium	Medium

should be satisfied. "Aura", "Gaia", "Roam", "FollowMe", "A2M" and "ScudOSGi" allow for the possibility of a different application or modality of interaction when the user moves from an old site to a new one. They score low or medium on the seamless attribute. "One.world", "MDAgent", and "Francis' framework" get high seamlessness because they adopt the migration strategy with little or no state loss. "Aura", "Gaia", "MDAgent", "FollowMe", and the "Francis' framework" receive a medium score for smartness because they all provide some context services for perceiving a user's intention and situation. Benefiting from Apple's ecosystem and the cloud computing paradigm, "iCloud" gets good scores under the proposed evaluation scheme. Apple has provided an iCloud control panel for Windows users. However, it currently does not support Linux and any other mobile platform except iOS. The openness and interoperability of iCloud storage API is also limited, since all iCloud usage implies calls to the iOS/OS *X* SDK and is not available in any other way outside the Xcode environment.

From the above discussion, we can find some clues that are beneficial to help select the appropriate application mobility paradigm according to different requirements and computing conditions. As discussed in Section 2, critical migration requirements include *seamlessness*, *ubiquity*, *generality*, *smartness* and *complexity*. Computing conditions often focus on three aspects: *network stability*, *network bandwidth* and *device heterogeneity*. In Fig. 9, each factor is valued as low, medium and high. According to different factor values, four curves are sketched out to provide a reference for application mobility paradigm selection. For example, if a user addresses migration seamlessness and solution generality and the network is remarkably reliable, input/output mobility of a remote desktop service would be the preferred mobility strategy. If network bandwidth is always good enough, and the user does not mind doing some manual operation, workspace migration in virtualization technology would work pretty well. Someone would argue that it is a thick client solution and may not be appropriate for resource constrained devices. Fortunately, some virtualization platform for mobile device is emerging, such as Pocket ISR and VMWare MVP [50]. MobiDesk provides a utility computing infrastructure for desktop migration by virtualizing display, OS and network [51]. SnowFlock provides an approach to speed up the VM cloning process based on VM fork abstraction and the cloud computing environment [52]. The techniques of lazy state replication, avoidance heuristics and multicast distribution are all useful in shortening VM recovery time and improving scalability. Middleware enables lightweight, context-aware and scalable application mobility in a moderate way. Each mobility strategy (i.e. task/component/code/object mobility) has its own pros and cons as Table 3 concludes. Cloud service such as "iCloud" would be more competitive if the user trusts in cloud providers and does not care about dependence on one specific ecosystem. Some good experience from middleware approaches is possible to be absorbed and transformed into cloud services.

5. Future directions

Since there is not an all-around satisfactory solution till now, researchers are still working hard on perfecting each proposal, especially those middleware services. Fig. 7 gave a minimal middleware service set. However, some issues that are not covered in this set should be studied in the future.

Smartness is a weak point in most proposals. As is discussed above, little existing work provides enough smartness in the desired user distraction-free manner. One of the difficulties comes from user mobility and context prediction which rely much on artificial intelligence technologies including planning, reasoning and machine learning, etc. A lot of reasoning technologies, such as first order logic, fuzzy logic, Bayesian network and neutral network, are exploited in context (e.g. location) reasoning and prediction services [5,43,53–56]. In practice, an intelligent and mobile agent has played a crucial role in some pervasive computing middleware [57–60], acting as an autonomous and intelligent delegate for the user. With intelligence weaved into the agent, it is possible to make the user's application more smart.

Security and privacy can never be ignored in software system design and implementation. To enjoy a distraction-free experience, users have to sacrifice some of their privacy. Specific to application mobility, a user's location and activity

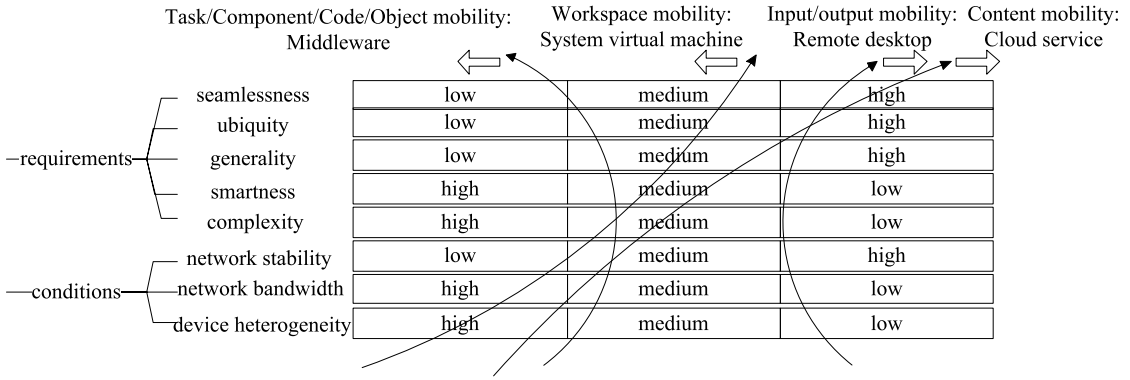

Fig. 9. Reference for application mobility strategy selection.

should be perceived by the context-aware system. In addition, the system should be aware of the user's preferences and intention in order to autonomously configure services in the user's favorite manner. Without careful protection, it will annoy users if their private information is stolen by others. On the contrary, when users switch to a new environment with the expectation of exploiting resources there, they should access these resources in a safe and legal way. Consequently, to enable seamless application migration, the system should not only protect the user's information from unauthorized access, but also protect resources from untrustworthy and malicious access. Lu et al., proposed a contextual rule based access control policy mechanism enriched with methods of generating blurred context and guaranteeing information anonymous [61]. However, sometimes a secure migration request will still be blocked or denied by some protection software. For example, to avoid service denial, a secure and attack-resilient desktop computing hosting infrastructure is proposed by extending MobiDesk [51] with a stateless and secure communication protocol, a single-hop indirection-based network and a remote display architecture [62].

The evolution of a mobility pattern challenges the traditional architecture of the software system. When mobility is considered as a first class property, existing *software architectural principles* need to be adapted. Some models have been proposed for logic mobility [63], but they did not rise to the high level of software architecture. In fact, mobility profoundly challenges traditional connectors, which is the core concept of software architecture. Connectors should not only coordinate interaction between components, but also deal with component migration. FarGo provides an elegant solution to capturing commonly occurring movement coordination patterns [64]. CommUnity allows connectors to coordinate both the behavior and the movement of the interconnected components and make each one dependent on the other, by introducing a specific data type that captures the required physical and/or logical mobility space [65,66]. Ali et al., proposed Ambient-PRISMA for modeling and developing distributed and mobile applications [67]. They borrowed the concept of ambient calculus [68] and introduced "ambient" as a new kind of architectural elements, which defines a bounded place where other architectural elements reside and is coordinated with the exterior of the boundary. Component migration is modeled by the in/out operation as in the ambient calculus. Based on the successful experience in architecture-based dynamic adaptation, Malek et al. tried an architecture-driven approach to modeling and analyzing mobility, as well as reasoning and managing the runtime adaptation of software systems [69].

Mobile cloud computing (MCC) leads a new trend in the future since it combines the advantages of both mobile computing and cloud computing [70]. MCC applications move the computing power and data storage away from mobile phones and into the cloud. As one of the main features of MCC, *mobile code offloading* helps mobile devices overcome obstacles related to performance, the environment and security. Essentially, mobile code offloading is a form of application mobility, specifically code mobility. Its critical issues include determining whether to offload, where to offload and what code to be offloaded. Algorithms of program partition are explored in [71–74] for computation offloading in a static environment. A few approaches take dynamic environment changes into consideration and provide more efficient code migration strategies [75–77]. On the cloud side, Microsoft and Google both offer cloud computing platforms and cloud services to enable seamless migration of computation across devices. Microsoft's project *Hawaii* helps Windows Phone application developers to access Windows Azure and a set of cloud services (e.g. relay service and speech-to-text service) for computation and data storage [78]. *Google App Engine* helps developing and hosting scalable applications in Google managed data centers [79]. They both give chances to mobile application developers to migrate some of the applications' functionalities to the cloud. *Monterey* middleware developed by Cloudsoft enables highly efficient, fine-grained application mobility. It ensures that applications have the flexibility at runtime to optimize code placement in accordance with cost, demand or compliance by using policy-based mobility [80].

6. Conclusion

Application mobility is an attractive and promising property of mobile and pervasive computing. It brings a continuous computing experience with no or very low user distraction even in the state of motion and/or under other changing context.

It also makes better use of ubiquitous computing resources. However, to support seamless application mobility, one must face great challenges in breaking the bindings between users, devices and the execution environments. Tremendous research efforts in pervasive computing area have been put on the design and implementation of application mobility middleware systems, but up to now very limited success has been achieved in terms of practical acceptance.

In this paper, we investigated some existing work and proposed a design concern framework which is structured along four dimensions: temporal, spatial, entity axes and an extra axis for other design concerns. These concerns together determine the strategy of making application seamlessly move from one environment to another. We reviewed a lot of approaches to application mobility and classified existing proposals into four groups: remote desktop, system virtual machines, middleware and cloud services. We also analyzed the strength and weakness of each proposal. A reference for application mobility strategy selection is put forward based on comprehensive comparisons.

Certainly, this paper could not include all projects on application mobility. However, the conceptual framework presented in this paper is believed to be a general one that can be used to evaluate new approaches. We hope this survey can provide a better understanding of the issues in supporting application mobility and serve as a guideline for further application mobility research.

Acknowledgments

This research was supported by the National Key Basic Research and Development Program of China (973) under Grant No. 2009CB320702, and the National Natural Science Foundation of China under Grant No. 61100037, 61021062, 60973044.

References

[1] M. Weiser, The computer for the twenty-first century, Scientific American 265 (1991) 94–101.
[2] D. Aaha, A. Mukherjee, Pervasive computing: a paradigm of the 21st century, Computer 36 (3) (2003) 25–31.
[3] M. Satyanarayanan, Pervasive computing: vision and challenges, IEEE Personal Communications 8 (4) (2001) 10–17.
[4] D. Garlan, D. Siewiorek, A. Smailagic, P. Steenkiste, Project aura: toward distraction-free pervasive computing, IEEE Pervasive Computing 1 (2) (2002) 22–31.
[5] M. Roman, C.K. Hess, R. Cerqueira, A. Ranganathan, R.H. Campbell, K. Nahrstedt, A middleware infrastructure to enable active spaces, IEEE Pervasive Computing 1 (4) (2002) 74–83.
[6] M. Satyanarayanan, M.A. Kozuch, C.J. Helfrich, D.R. O'Hallaron, Towards seamless mobility on pervasive hardware, Pervasive and Mobile Computing 1 (2) (2005) 157–189.
[7] M. Bylund, Z. Segall, Towards seamless mobility with personal servers, Info 6 (3) (2004) 172–179.
[8] A. Fuggetta, G.P. Picco, G. Vigna, Understanding code mobility, IEEE Transactions on Software Engineering 24 (5) (1998) 342–361.
[9] C.E. Perkins, Mobile IP, IEEE Communications Magazine 35 (5) (1997) 66–82.
[10] SIP: Session Initiation Protocol. http://www.ietf.org/rfc/rfc3261.txt.
[11] P. Yu, J.N. Cao, W.D. Wen, J. Lu, Mobile agent enabled application mobility for pervasive computing, in: Proc. UIC'06, Springer, Wuhan, 2006, pp. 648–657.
[12] J. Li, Y.Y. Bu, S.X. Chen, X.P. Tao, J. Lu, Followme: on research of pluggable infrastructure for context-awareness, in: Proc. AINA'06, Vol. 01, IEEE, Vienna, 2006, pp. 199–204.
[13] http://www.bluetooth.com.
[14] SLP. RFC2608.
[15] http://www.salutation.org.
[16] http://surrogate.jini.org.
[17] D. Milojicic, F. Douglis, R. Wheeler, Mobility: Processes, Computers, and Agents, ACM Press, 1999.
[18] G. Banavar, J. Beck, E. Gluzberg, et al., Challenges: an application model for pervasive computing, in: Proc. MobiCom, Boston, 2000, pp. 266–274.
[19] G.C. Roman, G.P. Picco, A.L. Murphy, Software engineering for mobility: a road map, in: Proc. ICSE, Limerick Ireland, 2000, pp. 241–258.
[20] http://msdn.microsoft.com/library/default.asp?url=/library/en-us/termserv/termserv/remote_desktop_protocol.asp.
[21] T. Richardson, Q. Stafford-Fraser, K.R. Wood, A. Hopper, Virtual network computing, IEEE Internet Computing 2 (1) (1998) 33–38.
[22] OSXvnc homepage: http://sourceforge.net/projects/osxvnc/.
[23] M. Satyanarayanan, B. Gilbert, M. Toups, N. Tolia, A. Surie, D.R. O'Hallaron, A. Wolbach, J. Harkes, A. Perrig, D.J. Farber, M.A. Kozuch, C.J. Helfrich, P. Nath, H.A. Lagar-Cavilla, Pervasive personal computing in an Internet suspend/resume system, IEEE Internet Computing 11 (2) (2007) 16–25.
[24] B. Gilbert, A. Goode, M. Satyanarayanan, Pocket ISR: virtual machines anywhere, Carnegie Mellon University School of Computer Science, CMU-CS-10-112, March, 2010.
[25] S. Smaldone, B. Gilbert, N. Bila, L. Iftode, E. de Lara, M. Satyanarayanan, Leveraging smart phones to reduce mobility footprints, in: Proc. MobiSys'09, ACM, Kraków, Poland, 2009, pp. 109–122.
[26] R. Cáceres, C. Carter, C. Narayanaswami, M. Radhunath, Reincarnating PCs with portable soulpads, in: Proc. MobiSys'05, ACM, New York, 2005, pp. 65–78.
[27] VCE coalition: enhanced business continuity with application mobility across data centers, Technical Report, 2010.
[28] B.K. Schmidt, Supporting ubiquitous computing with stateless consoles and computation caches, Ph.D. Thesis, Computer Science Department, Stanford University, August, 2000.
[29] R. Chandra, N. Zeldovich, C. Sapuntzakis, M.S. Lam, The collective: a cache-based system management architecture, in: Proc. the Second Symposium on Networked Systems Design and Implementation 2005, Boston, 2005, pp. 259–272.
[30] http://www.jini.org.
[31] http://www.upnp.org/.
[32] J.P. Sousa, D. Garlan, Aura: an architectural framework for user mobility in ubiquitous computing environments, in: Proc. the 3rd Working IEEE/IFIP Conference on Software Architecture, Kluwer Academic Publishers, 2002, pp. 29–43.
[33] M. Roman, H. Ho, R. Campbell, Application mobility in active spaces, in: Proc. the 1st International Conference on Mobile and Ubiquitous Multimedia, Oulu, Finland, 2002.
[34] A. Ranganathan, C. Shankar, Roy H. Campbell, Application polymorphism for autonomic ubiquitous computing, Multiagent and Grid Systems 1 (2) (2005) 109–129.
[35] G.E. Krasner, S.T. Pope, A description of the model-view-controller user interface paradigm in the smalltalk-80 system, Journal of Object-Oriented Programming 1 (3) (1988) 26–49.
[36] R. Grimm, System support for pervasive applications, ACM Transactions on Computer Systems 22 (4) (2004) 421–486.

[37] R. Grimm, One. world: experiences with a pervasive computing architecture, IEEE Pervasive Computing 3 (3) (2004) 22–30.
[38] Y. Zhou, J.N. Cao, V. Raychoudhury, J. Siebert, J. Lu, A middleware support for agent-based application mobility in pervasive environments, in: Proc. ICDCSW'07, IEEE, Toronto, 2007.
[39] D.B. Lange, M. Oshima, Seven good reasons for mobile agents, Communications of the ACM 42 (3) (1999) 88–89.
[40] H. Chu, H. Song, C. Wong, S. Kurakake, M. Katagiri, Roam, a seamless application framework, Journal of Systems and Software 69 (3) (2004) 209–226.
[41] S.X. Chen, Y.Y. Bu, J. Li, X.P. Tao, J. Lu, Toward context-awareness: a workflow embedded middleware, in: Proc. UIC 2006, Springer, Wuhan, 2006, pp. 766–775.
[42] http://www.osgi.org/.
[43] T. Gu, H.K. Pung, D.Q. Zhang, A service-oriented middleware for building context-aware services, Journal of Network and Computer Applications 28 (1) (2005) 1–18.
[44] S. Helal, W. Mann, et al., The gator tech smart house: a programmable pervasive space, Computer 38 (3) (2005) 50–60.
[45] F.M. David, B. Donkervoet, J.C. Carlyle, E.M. Chan, R.H. Campbell, Supporting adaptive application mobility, in: Proc. OTM 2007 Ws, Part II, in: LNCS, vol. 4806, Vilamoura, 2007, pp. 896–905.
[46] A. Ahlund, K. Mitra, D. Johansson, C. Ahlund, A. Zaslavsky, Context-aware application mobility support in pervasive computing environments, in: Proc. Mobility 2009, ACM, Nice, 2009.
[47] Y.Q. Xu, S.J. Li, G. Pan, ScudOSGi: enabling facility-involved task migration in OSGiFramework, in: Proc. FCST'09, IEEE, Shanghai, 2009, pp. 125–131.
[48] iCloud: http://www.apple.com/icloud/.
[49] iCloud for developers: https://developer.apple.com/icloud/.
[50] http://www.vmware.com/products/mobile/.
[51] R.A. Baratto, S. Potter, G. Su, Jason Nieh, MobiDesk: mobile virtual desktop computing, in: Proc. MobiCom'04, ACM, Philadelphia, 2004, pp. 1–15.
[52] H. Andrés Lagar-Cavilla, J.A. Whitney, R. Bryant, et al., SnowFlock: virtual machine cloning as a first-class cloud primitive, ACM Transactions on Computer Systems (TOCS) 29 (1) (2011) Article 2, 1–45, ACM, New York.
[53] A. Ranganathan, J. Al-Muhtadi, R.H. Campbell, Reasoning about uncertain contexts in pervasive computing environments, IEEE Pervasive Computing 3 (2) (2004) 10–18.
[54] R. Mayrhofer, H. Radi, A. Ferscha, Recognizing and predicting context by learning from user behavior, in: Proc. MoMM'03, Austrian, 2003, pp. 25–35.
[55] P. Nurmi, M. Martin, J.A. Flanagan, Enabling proactiveness through context prediction, in: Proc. CAPS 2005, Helsinki University Press, Helsinki, 2005.
[56] R. Cheung, J.N. Cao, G. Yao, A. Chan, A fuzzy based service adaptation middleware for context-aware computing, in: Proc. EUC'06, Springer, Seoul, 2006, pp. 580–590.
[57] N. Hanssens, A. Kulkarni, R. Tuchida, T. Horton, Building agent-based intelligent workspaces, in: Proc. IC'02, CSPEA Press, Las Vegas, 2002, pp. 675–681.
[58] M. Kumar, et al., PICO: a middleware framework for pervasive computing, IEEE Pervasive Computing 2 (3) (2003) 72–79.
[59] H. Chen, F. Perich, et al.,Intelligent agents meet semantic web in a smart meeting room, in: Proc. AAMAS'04, New York, 2004, pp. 69–79.
[60] J. Lu, Y. Zhu, B. Du, L. Ren, The mobile agent based service migration mechanism in wide area pervasive computing system, in: Proc. WASE'09, Taiyuan, 2009, pp. 375–378.
[61] W.T. Lu, J. Li, X.P. Tao, X.X. Ma, J. Lu, Shadow: a middleware in pervasive computing environment for user controllable privacy protection, in: Proc. EuroSSC'06, Enschede, 2006, pp. 143–158.
[62] A. Stavrou, R. Baratto, A. Keromytis, J. Nieh, A2M: access-assured mobile desktop computing, in: Proc. ISC'09, Pisa, 2009, pp. 186–201.
[63] N. Medvidovic, G. Edwards, Software architecture and mobility: a roadmap, The Journal of Systems and Software 83 (2010) 885–898.
[64] O. Holder, I. Ben-Shaul, H. Gazit, Dynamic layout of distributed applications in FarGo, in: Proc. ICSE'99, Los Angeles, 1999, pp. 163–173.
[65] J.L. Fiadeiro, A. Lopes, CommUnity on the move: architectures for distribution and mobility, in: Proc. the International Symposia on Formal Methods for Components and Objects 2003, Leiden, 2003, pp. 177–196.
[66] C. Oliveira, M. Wermelinger, J.L. Fiadeiro, A. Lopes, Modelling the GSM handover protocol in CommUnity, Electronic Notes in Theoretical Computer Science 141 (3) (2005) 3–25.
[67] N. Ali, I. Ramos, C. Solis, Ambient-PRISMA: ambients in mobile aspect-oriented software architecture, The Journal of Systems and Software 83 (2010) 937–958.
[68] L. Cardelli, Abstractions for mobile computation, in: Secure Internet Programming: Security Issues for Distributed and Mobile Objects, in: LNCS, vol. 1378, Springer, 1998, pp. 140–155.
[69] S. Malek, G. Edwards, Y. Brun, H. Tajalli, J. Garcia, I. Krka, N. Medvidovic, M. Mikic-Rakic, G.S. Sukhatme, An architecture-driven software mobility framework, The Journal of Systems and Software 83 (2010) 972–989.
[70] H.T. Dinh, C. Lee, D. Niyato, P. Wang, A survey of mobile cloud computing: architecture, applications, and approaches, in: Wireless Communications and Mobile Computing, John Wiley & Sons, Inc, 2011.
[71] G. Chen, B.T. Kang, M. Kandermir, N. Vijaykrishnan, M.J. Irwin, R. Chandranouli, Studying energy trade offs in offloading computation/compilation in Java-enabled mobile devices, IEEE Transactions on Parallel and Distributed Systems 15 (9) (2004) 795–802.
[72] K. Kumar, Y. Lu, Cloud computing for mobile users: can offloading computation save energy, Computer 43 (4) (2010) 51–56.
[73] Z. Li, C. Wang, R. Xu, Computation offloading to save energy on handheld devices: a partition scheme, in: Proc. the 2001 International Conference on Compilers, Architecture, and Synthesis for Embedded Systems, CASES, 2001, pp. 238–246.
[74] C. Wang, Z. Li, A computation offloading scheme on handheld devices, J. Parallel Distrib. Comput. 64 (2004) 740–746.
[75] S. Ou, K. Yang, A. Liotta, L. Hu, Performance analysis of offloading systems in mobile wireless environments, in: Proc. the IEEE International Conference on Communications, ICC, Glasgow, 2007, pp. 1821–1826.
[76] B.-G. Chun, P. Maniatis, Dynamically partitioning applications between weak devices and clouds, in: Proc. the 1st ACM Workshop on Mobile Cloud Computing & Services: Social Networks and Beyond, MCS, San Francisco, 2010, p. 7.
[77] E. Cuervo, A. Balasubramanian, Dae-Ki Cho, A. Wolman, S. Saroiu, R. Chandra, P. Bahl, MAUI: making smartphones last longer with code offload, in: Proc. the 8th International Conference on Mobile Systems, Applications, and Services, MobiSys, San Francisco, 2010, pp. 49–62.
[78] Project Hawaii: http://research.microsoft.com/en-us/projects/hawaii/.
[79] Google App Engine: https://appengine.google.com.
[80] Monterey whitepapers: http://www.cloudsoftcorp.com.