Mobile Information Systems 8 (2012) 231–254
DOI 10.3233/MIS-2012-0141
IOS Press

# Context-aware HCI service selection

Yao Shen[a], Minjie Wang[a], Xiaoxin Tang[a,*], Yi Luo[b] and Minyi Guo[a]
[a]*Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai, China*
[b]*China Xinhua Network Co. Ltd, Beijing, China*

**Abstract.** Context-awareness has become a key issue in Human-Computer Interaction(HCI) to provide better user experience under multi-device and multi-modal environment. With this intuition, we have proposed a web service based framework, which associates interactions with services, and provided service selection mechanism using context knowledge to achieve smart interaction migration [1]. A fundamental problem of such a service-oriented framework for interaction migration is to design an effective while scalable algorithm for service selection. In this paper, we propose a service selection algorithm considering not only context information and user preferences but also inter-service relations such as relative location. Our algorithm detects interaction hot spots within user active scope and presents the best service combination based on evaluation of interaction effectiveness. We also conduct simulation and the results illustrate that our algorithm is effective and scalable for interaction service selection.

Keywords: Human-computer interaction, interaction migration, multi-modal interaction, pervasive computing

## 1. Introduction

Computer systems play a prime role in support of information delivery. Generally, the effectiveness of information systems largely relies on two factors: information quality and presentation. On the other hand, expansion of the Internet and other sources of digital media have provided people with access to a wealth of information. This trend (or *"information overload"*) raises new requirements for both high information quality and smart presentation. Moreover, versatile sources of digital media greatly extend computer interfaces and thereby promote Human-Computer Interaction(HCI) technologies to a new generation.

Traditional HCI is restricted to one-one mode, that is one user only interacts with one device like personal computer through a few communication channels such as mouse and keyboard. Research works on HCI have been promoting new technologies and improving user experience. However, in past decades, the rise of mobile devices (e.g. PDA, Smart phone, tablet computer, etc.) hits the monotonous interaction pattern between human and personal computers, and the development in computer vision and sound processing enriches communication channels to digital media (e.g. speech, gesture, touch, etc.). With such trend of ubiquitous computing [2], traditional HCI technology gradually appears to be insufficient and inconvenient, while collaborative and distributed HCI technologies using multiple devices and interaction modalities are gaining more and more attentions. One solution in such HCI area

[*]Corresponding author: Xiaoxin Tang, Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai 200240, China. E-mail: tang.xiaoxin@sjtu.edu.cn.

is called *interaction migration*, that is to migrate interaction process among multiple devices and even different modalities in a graceful manner.

Interaction migration can be achieved at different levels. *Process level migration* is a straight-forward intuition but its bottleneck in transferring large data and reconstructing process on various embedded platforms hampers its widespread use. *Task level migration*, on the other hand, extracts logic tasks and their relationships from application using model-based method [3] and then migration can be achieved by distributing tasks to multiple devices. However, although such migration techniques are good at modeling logic and temporal relationships among tasks, to construct ontology and categorization of tasks is non-trivial, thus placing obstacles for migration among interaction modalities. In recent years, a more flexible and nature concept – *service-oriented* concept – has been proposed for HCI modeling. Since a majority of commercial information systems support and provide services, the nature and progression of the service encounter should be a key concern of human computer interface designers [4]. This *service-oriented* HCI concept transforms interaction migration problem to *service selection* problem and attempts to find solutions from web service selection techniques. We adopt the service-oriented concept and proposed a web service based HCI migration framework in our previous work [5] and discussed methods for service selection in [1]. The framework has its pros and cons. Three main pros are mentioned in the paper. Our framework models interaction logic of application as interaction service and through such modeling and ontology of services, it is possible to migrate interaction to multiple devices and modalities. Besides, user preference and context-awareness are concerned in the framework by enclosing them into descriptions of interaction interfaces. Moreover, the framework supports dynamically emerging and disappearing devices in ubiquitous environment. However, it does not consider correlations between services, for example relative locations which may lead to a possibly bad service combination result but each individual services are optimal in our concerns (we will show this in Section 5).

With these concerns, we present in this paper a new context-aware HCI service selection process which keeps the flexibility of service-oriented concepts and replaces the original selection algorithm with a new scalable algorithm utilizing both user and inter-service contexts. The main contributions of this paper are:

– *Context-awareness* method for service selection. Take both *inter-service* contexts and user contexts into consideration during migration request submission and service matching algorithm, making matching decision more flexible and intelligent.
– Service selection algorithm with good *scalability*. Introduce *interaction hot spot* to evaluate service combination results.
– Formulate device matching during migration process as service combination discovery process. Use service concept to describe interaction functionalities of application. This *service-oriented* concept generalizes interaction types and makes it possible to migrate interaction to multiple devices and modalities.

## 2. Related work

The proliferation of mobile devices and networks leads to an increasing demand for multi-device and multi-modal interactions. The idea of *interaction migration* has been proposed for decades to meet this demand, in which HCI can be migrated across different platforms and modalities to provide better user experiences. Interaction migration systems like [3,6,7] take advantage of model-based method to separate interaction tasks and generate corresponding interfaces on target platforms. The authors of [5,8], however, use another mechanism based on web service structure to provide migration under pervasive computing environment [2]. In the work of [1,5], we propose a web service based framework for interaction migration and corresponding service selection method.

Our framework follows *Service-oriented Architecture*(SOA), which models HCI by interaction services, and then formulates devices' semantics matching as a service selection process. To treat the issue of service selection, or service combination discovery, researchers have developed language descriptions for web services such as WSDL [9], BPEL [10] and DAML [11]. These languages successfully unify the structure and provide a formal description of web service functionality. The shortage of these languages are their ignorance of semantics. Therefore, ontology languages, such as OWL-S [12], are deployed to specify the web service semantics.

Within these ontology-based service matching methods [13–17], OWLS-MX [13] is the first hybrid OWL-S service matchmaker. It exploits means of both logic based and information retrieval (IR) based approximate matching. It uses IR to handle semantics, thus improving its performance in web service matching [14] uses ranking method to decide the most suitable services. It also presents a device ontology that provides a general framework for device description. When the accuracy and scalability of semantic matching process are considered, ranking surpasses other methods for its robustness. Other researchers take a look into a finer decomposition of applications. For example, ScudWare [15] successfully proposes a middleware platform for smart vehicle space. Different from other frameworks, it further divides service into many interdependent components. These components support migration and replication, thus making them able to be distributed onto different devices. The behaviors of components are crucial during the process of application migration and thereby is non-trivial to perform such partition.

Besides service modeling, another important factor needed to be concerned in service selection is contexts. Defined by [18], a *context* is *"any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and the application themselves"*. Combining contexts into service selection makes selection framework more flexible and more intelligent to meet environment changes and user behaviors automatically [19] presents a context-based matching for web service combination. The paper adopts an ontology-based categorization, two-level mechanism for modeling and a peer-to-peer matching architecture. Another implementation of context-awareness is published in [20], where the authors add context attributes to the famous mobile service discovery system Jini. In [21], the authors highlight the context-awareness in mobile network environment. They propose an algorithm for context-aware network selection.

Our approach also consider contexts in semantic matching. Our framework is similar to [20], because we integrate context attributes into interaction device description and use them to aid service matching. Besides context information, we also consider user preferences of each kind of services by recording interaction history. This kind of information helps match user-expecting devices for certain services, thus increasing matching accuracy.

In order to achieve better accuracy, we have made two assumptions: one is that services are well categorized so that different services are distinguished from each other; the other is that a powerful middleware is used so that services are described in a standard and unified format, which greatly reduces the difficulty for service matching. We believe that the two assumptions are reasonable. For the first one, services in pervasive computing environment can be categorized according to their corresponding hardware features, due to the reason that they are device-oriented and their service boundaries are definite. For the second one, powerful middlewares are being well studied and some of them are even OS level platforms. These progresses in researches and industry practices have made it possible to do semantic matching on the middleware level.

The authors of [22] consider the problem of bottlenecks in centralized dynamic query optimization methods due to messages exchange on a bad network. They present a decentralize method to do the optimization processes. Authors of [23] have done an interesting research on user measurement of the adoption of mobile services. They list some of the constructs that may influence the user acceptance of services. "Context", "personal initiatives and characteristics", "trust", "perceived ease of use", "perceived usefulness" and "intention to use" are mentioned as important constructs of their instrument.

Service selection is a classical problem in web service related researches. There are several projects studying the problem of QoS-empowered service selection. In [24], authors present a QoS-aware middleware supporting quality-driven Web service combination. They propose two service selection approaches for constructing composite services: local optimization and global planning [25] gives a similar approach in service selection with QoS constraints in global view. Both methods are based on linear programming and best suitable for small-sized problems as their complexity increases exponentially when problem size increases [26] presents an autonomic service provisioning framework to establish QoS-assured end-to-end communication paths across independent domains. They model the domain composition and adaptation problem as classical k-multi-constrained optimal path (MCOP) problems. These works are all internet-wide service selection, thus do not consider the service location during selection. In HCI, service location is an important factor in service selection. We use a local matching procedure to get the matching degree for each service, and a global selection procedure to find the best service combination. Service combination is selected when they can effectively cover some areas, defined as interaction hot spots, where user can effectively achieve HCI requirements.

The authors of [27] introduce a context-based collaborative selection of SOA services. While it takes service locations into account, the distance-based location is not sufficient for HCI service selection since most devices/services have not only the requirement of interaction range but also the best interaction angle. In [28,29], the authors give an introduction to HCI migration. They mentioned the problem of device location, but had not defined any location constraints, so actually did not integrate location into their model.

## 3. Interaction service selection framework

### *3.1. Selection process overview*

The basic idea of our design is that when encountering interaction migration, users are usually inexpert in deciding the target platform because they do not hold the insight into device capability and compatibility. Therefore, our selection method aims at providing an automatic selection mechanism rather than user-initiative selection with the help of context information and user preference. We also separate interaction tasks from application logic and model them as *interaction services*. With the help of service description technology, we design a framework for *interaction service selection* to achieve multi-modal and multi-platform migration.

Figure 1 shows an overall description of selection process in our framework. Three main components consists of the interaction environment:*User Device*, which contains user identification and personal information; *Context Manager*, which maintains contexts catched by various sensors and *Interaction Device*, which performs service matching and provides interaction interfaces. Following steps are involved in the whole process.

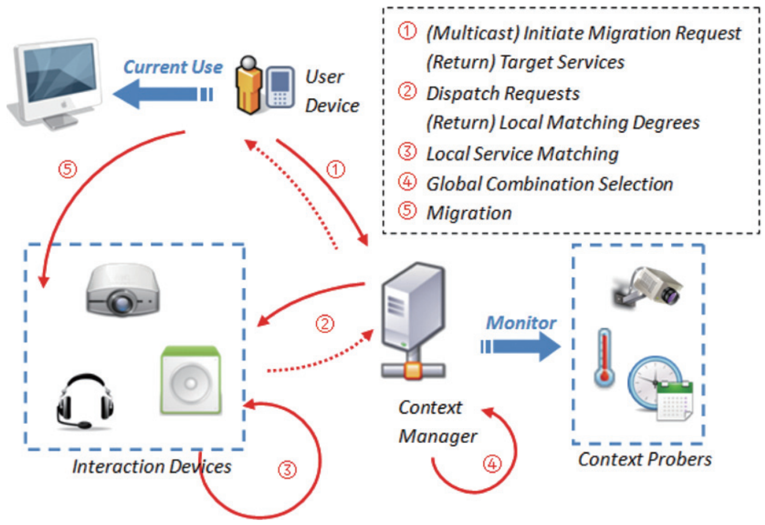

Fig. 1. Selection Process Overview. Three main components: *User Device*, which contains user identification and personal information; *Context Manager*, which maintains contexts catched by various sensors; *Interaction Device*, which performs service matching and provides interaction interfaces.

1. *Initiate migration request*: Migration requests can either be driven by context or announced by users. In both situations, those requests are sent to *context manager*. In our design, *context manager* takes charge of all sensor networks in the environment while *user device* holds user identification. Context manager is able to trace the context changing of certain user by monitoring parameters such as user position, facial expressions and gestures. A key idea is that it is often natural for a user to migrate interaction when context switch occurs. For instance, a user may express the needs for interaction migration by performing certain gesture or simply approaching target devices. Techniques such as human behavior modeling [30] are potential solutions for such context detection. Therefore, for user-driven migration request, user device directly initiate migration request to context manager. Notably, although initiated by user, migration requests does not directly specify target devices, thus different from user-initiative selection method. On the other hand, when context switch is detected (context-driven situation), context manager will announce the user device to send migration request back. This apparent redundant behavior is necessary because context manager does not hold any user information for privacy concerns.
2. *Dispatch requests*: The whole service selection process in our framework is divided into two parts: *Local Service Matching* and *Global Combination Selection*. *Local Service Matching* is performed on device terminals in a distributed manner. Therefore, after receiving migration requests from user device, context manager dispatches requests to devices near user for local service matching. The matching degrees will send back to context manager after calculation. Future improvement such as device filtering and requests reforming can be implemented within this step for accuracy and performance concerns.
3. *Local service matching*: Local service matching is performed on device terminals and return matching degree back to context manager. Service functionality requirements, user preferences and contexts are taken into account in the algorithm. For those interaction devices who cannot provide these services, they will not respond to the request, thus alleviate burdens on network transmission under device-rich environment. Context information can be retrieved from context manager if required in local selection matching.

4. *Global combination selection*: After receiving matching degrees calculated by device terminals, context manager selects the best service combination regarding user's requests. User request, inter-service contexts and local matching degrees are integrated here to discover a good service combination. Due to the complexity of combination problem, we provide an approximate algorithm for finding optimal service combination (Section 3.5). The final selected service combination results are returned back to user device.
5. *Migration*: After user device obtains target interaction devices, it multicasts connection request to those targets. Selected devices then send back web service descriptions so that connections between user device and targets can be established. As we focus on service selection, subsequent interaction processes are not included in this paper.

Sections below will discuss details of main components. Section 3.2 focuses on user devices and describe how we model interaction as services and how we make usage of user history and user preference. Section 3.3 talks about context manager and the information it monitors. Section 3.4 describes our local service matching algorithm and how we integrate user history and context information into it. Section 3.5 illustrates our algorithm for discovering service combination.

## 3.2. User devices

### 3.2.1. Service-oriented middleware support

Although having been proposed for decades, the concept of interaction migration does not achieve a general implementation in practical applications. The major obstacle is that it is hard for application developers to cover tedious details of interaction process and to modify existing applications to support such migration. To solve this problem, we put forward an *HCI Migration Support Environment*(MSE) in our previous work [5], which lay between application level and OS level, acting as a middleware, hiding platform-dependent details from users and providing interaction migration APIs for upper applications.

In this paper, we extend our previous MSE middleware structure, in which HCI processes can be fulfilled though calling *interaction services*. In the system, local and remote interaction services provided by interaction devices can be called with a unified interface. Therefore, when encountering interaction logic, the upper level applications just call APIs provided by our middleware and let the middleware select target interaction services. Hence, interaction migration, in our framework, can be treated as seeking proper remote service combination that can fulfil requesting interaction requirements. Our service-oriented middleware then needs to take charge of extracting service descriptions based on application's interaction behavior. Two questions should be answered in the descriptions: *"What services are needed?"* and *"What is the basic properties of these services?"*. Our system can solve these questions by tracing the APIs used by upper level applications. We categorize APIs into several interaction service types (such as "Video Display Service" and "Keyboard Input Service") and record key parameters as service properties. Therefore, once an application calls some API, our middleware can specify corresponding types and properties of services called by the application (shown in Fig. 2).

### 3.2.2. User history and preference

We assume that different users possess different preferences for interaction devices and such preferences can be acquired from users' interaction history. For example, a user who prefers large screens may have a higher average screen size of his past used devices. Therefore, catching the parameters of user-expecting device from history is crucial to select appropriate interaction devices. In our method, we adopt service-oriented concept and associate parameters of device hardware with interaction services.

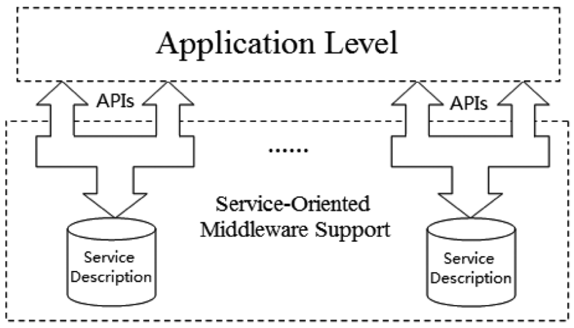

Fig. 2. Service-oriented middleware support.

Besides, the interaction services are categorized into independent groups such as *Video Output* and *Keyboard Input* to facilitate service matching.

Assume that there are $k$ service categories $C = \{c_1, c_2, ..., c_k\}$. Let a device $d$ with hardware feature set $F_d = \{f_1, f_2, ..., f_n\}$. We assume that it can provide services $S_d = \{s_1, s_2, ..., s_m\}$. The properties $P_i$ of service $s_i$ is a subset of the hardware feature set, i.e., $P_i \subseteq F_d$. Since each service can be assigned to a service category, we then extract *general properties* from service properties set. General properties actually belong to service category because they are used by all services in the same category. For instance, a desktop computer commonly has two service categories: *Video Output* and *Audio Output*. If it has two displays, then there are two interaction services on the computer that both belong to *Video Output* category. Among the properties of these two services, *screen size* and *resolution* are general properties.

In user device, we maintain service property set $U^c$ to represent *user preferences* and describe user's expected properties for service category $c \in C$. Notably, we ONLY record *general properties* for a service category since only those common properties can be used in comparison among the services in a category. User preferences are formed from device-using records in the past history. After a user selects a service $s$ from category $c$, user device will record general properties of the service and update the $i^{th}$ properties of corresponding user preferences by

$$u^c_{i,k} = (1 - \alpha)u^c_{i,k-1} + \alpha c_{i,k},$$

where $u^c_{i,k}$ denotes the $i^{th}$ property in user preferences for category $c$ after using a service from $c$ for the $k^{th}$ time, $\alpha$ denotes the update rate for the newly-coming device descriptions and $c_{i,k}$ denotes the $i^{th}$ general property value of the $k^{th}$ time selected service. Suppose the initial user preference is $u^c_{i,0}$, then a candidate value for $\alpha$ is $\frac{1}{k}$, which means the user preferences are calculated by averaging past device parameters. When requesting migration to some services, user device will envelope user preferences for these services and send them to nearby devices for service matching.

## 3.3. Context manager

Our goal is to propose a context-aware method to select proper interaction services which makes it necessary to deploy a *context manager* to monitor environment and users, record and analyze contexts and provide global access for interaction devices. Another important responsibility for context manager is to integrate multimodal context information such as speech, gesture, writings, facial expressions or combinations of them. Context manager runs as a software system connected with user devices and sensors. We do not put much emphasis on how context manager connects with sensors while we provide our insight into what contexts should be provided and how we use these information. We assume that in our framework, context manager is able to monitor data from sensors and detect context switch along with user behaviors. Two major parts of context information should be maintained by context manager:

- *Environment Context*: Including environment temperature, moisture, brightness, current time and etc., environment context is crucial to those interaction devices that have some running constraints under environment condition. For example, high temperature and moisture may greatly influence some sensitive devices, decreasing their process ability, thus harming user experiences. Another interesting usage of environment context is related to time records. For instance, a device may record when its services are being used, compare the current time with the records and judge whether itself is appropriate to provide such services.
- *User Context*: Researchers have made great progress on HCI technologies [31] such as face detection [32], expression analysis [33], gesture and large-scale body movement recognition [34], and even eye tracking [35]. For a smart pervasive environment, it is always common to deploy these technologies, corresponding sensors and algorithms, which are called *Context Probers* in our framework. Our context manager monitors these context probers and fetches context information from them. A simple example is user position which can be used to match nearest interaction devices. We can also apply user face orientation information based on detection technology to match proper screens in user's vision.

In our method, context information mainly aids interaction migration in two parts. Firstly, context information helps judge the moment to launch interaction migrations. Secondly, comparison between contexts and device parameters improves service matching accuracy which will be further discussed in Section 3.4. Moreover, context manager is responsible for the algorithm of service combination selection about which more details will be discussed in Section 3.5.

### 3.4. Local service matching

Local service matching in Fig. 1 is performed on each interaction device who receives migration requests from context manager. Matching degrees are then sent back to context manager for further selection of service combinations. Interaction devices are designed to calculate matching degrees since the context manager will be a network bottleneck if large amount of device and service information need to be transmitted back to context manager. A *device profile* is maintained on each interaction device to compare with service descriptions and contexts. The service matching procedure contains following three parts.

#### 3.4.1. Service property matching

The top priority in check list is to judge whether the current interaction device is capable to provide the service required, that is to determine whether the device can meet the basic properties of required service. Our solution is to keep device capability information in device profile, including supported service types and properties for each service type. Hence, interaction device merely needs to judge whether requested services exist in supported service categories and whether corresponding service properties are within capabilities of the HCI service. If service property matching fails, the device will not reply user device. Otherwise, following two matching processes will be performed and matching degree will be returned.

### 3.4.2. *User preference matching*

Our goal is to select potentially the most satisfying services for users. Therefore, this part calculates to what extent the interaction device satisfies user's expectation, namely user preference, when considering certain services. User preferences are enclosed in the migration request with format discussed in Section 3.2.2 while device features for its services are recorded in device profile.

### 3.4.3. *Context matching*

The main contribution of our method is the utilization of context. We first retrieve context information from context manager, and then evaluate environment and user context information by context evaluation function, pre-set in interaction devices, to measure whether devices are suitable to provide services under current circumstance. The context evaluation function $f_c(x)$ for a context $c$ varies according to the actual semantics of the context. For example, *Euclidean distance*

$$f_{position}(x) = \|\vec{x} - \vec{p}\|_2, \tag{1}$$

where $p$ denotes the position of a device, performs well for position context to measure whether a user is close to the device, while *cosine similarity* metric

$$f_{cosine}(x) = \frac{\vec{x} \cdot \vec{o}}{\|\vec{x}\|_2^2 \cdot \|\vec{o}\|_2^2}, \tag{2}$$

where $o$ denotes the orientation of a device, is suitable for context variables like user face orientation and screen orientation. Interaction devices can also specify weight $w^c_{d,s_i}$ for each context $c$ to denote how significant the context will influence service $s_i$'s selection on device $d$.

Final matching degree is summed up from the results of user preference matching and context matching. Matching degrees are calculated for each requested service. After collecting matching degrees, context manager starts global service combination selection.

## 3.5. *Global combination selection*

In Section 3.4, we presented a local service matching procedure by which matching degrees can be derived for each requested service. In simple scenarios, the information of service matching degrees is enough to be used for a successful interaction migration. However there are usually limitations that may lead to unreasonable migration. For example, since the local service matching procedure considers little about the inter-service contexts such as relative distance between two target devices, it may result in a bad user experience if the distance is too long. Therefore, we propose a global selection procedure, to find the best service combination. We focus on the relative locations as an example of processing interrelations among services. In fact, more general inter-service relations can be modeled in our framework using similar approaches. In order to represent the relation between different services in terms of relative distance, we develop a service-coverage model and corresponding search algorithm. The *coverage* idea is similar to target coverage scheduling proposed in [36]. Each interaction service in our framework resembles directional sensors in [36], but we use soft evaluation rather than hard classification for effective region of interaction services.

### 3.5.1. *Effective region*

In reality, different services on different devices have different range to provide effective interaction service. We use *effective region* to represent the range within which user can interact with a service

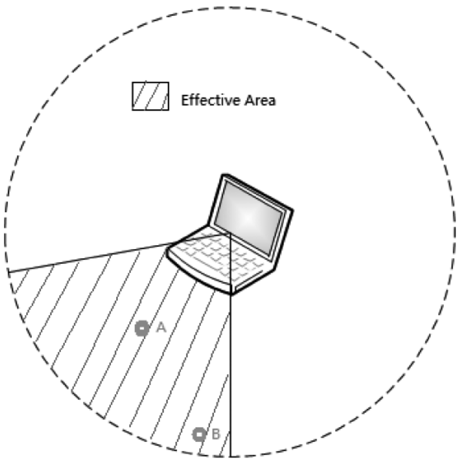

Fig. 3. The effective area is modeled as a sector.

under Quality of Service(QoS) guarantee. The effective region for a device's interaction service without any physical barrier can be reasonably modeled as a sector centered at the device. The sector angle represents the device's interaction angle. As illustrated in Fig. 3, a slashed area denotes the computer's effective region within which users can use the computer effectively.

For a given device, different service may have different effective region. For example, visual service may have a smaller effective region than audio service. Even for a given device's interaction service, HCI effectiveness may vary from point to point within its effective region. In Fig. 3, users may feel more convenient to use the computer at point *A* than at point *B*. Therefore, we use function $EV_i$ for service $i$ to represent the HCI effectiveness at point $p$:

$$EV_i(p) = \begin{cases} 0, & \text{if } p \text{ is out of effective region} \\ (0,1], & \text{otherwise} \end{cases} \tag{3}$$

The effectiveness is zero if $p$ is out of effective region while within the effective region, the function returns a device- and service-dependent score between $(0, 1]$.

### *3.5.2. User active scope*

Interaction migration happens only when the user enters a different area which is call in this paper the *User Active Scope*(*UAS*). The *UAS* information is maintained by context manager. We assume that context manager can properly divide user space into different *UASs* according to the information about building inside structure which can be initially set in the system. Meanwhile, users are inclined to move within a *UAS* for effective interactions.

It is flexible to define a *UAS* which depends on both environment and applications. For example, a room can be a single UAS in some situation while in other applications, it may be divided into several *UASs*.

In Fig. 4, a *UAS* is covered by several continuous squares. A *UAS* square is an atomic unit which users can step in or out of to interact with some devices. The *UAS* square will be used in our service combination selection algorithm presented in Section 3.5.3. It is obvious that a large *UAS* square can speed up our selection algorithm but decreases its accuracy.

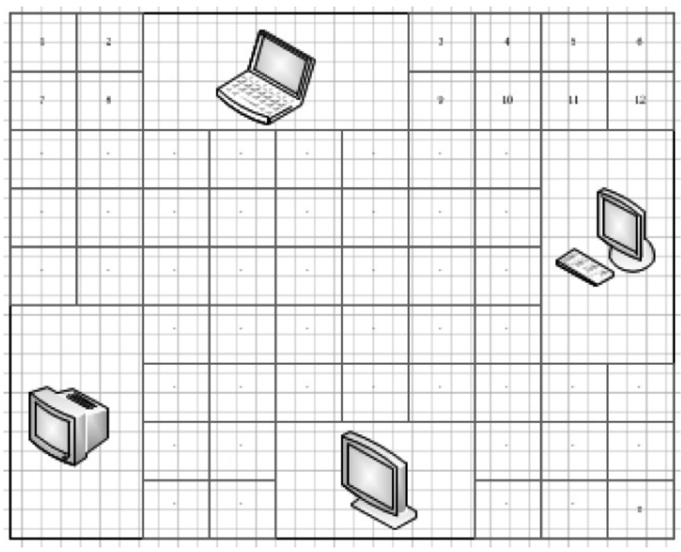

Fig. 4. *User Active Scope* is represented by continuous squares.

### *3.5.3. Service combination selection algorithm*

Service location concerns HCI migration. Actually, two individually best matched services usually cannot be combined to provide HCI services in one application if they are not close enough. Therefore, we develop a global combination selection algorithm to take service location into account. Our algorithm includes following two procedures:

1. Service coverage coloring procedure:
   Suppose there are $M$ different services in a *UAS* and for each square unit we use a $M$-bit string to represent whether users can effectively interact with the corresponding service in the square. If the effective region of service $i$ covers square $p$, then $sc[p][i]$ is set to 1; otherwise 0. We call this step as coloring which is performed on the context manager. The coloring procedure running on context manager includes following three steps:

   (a) The context manager retrieves information about user's current *UAS*;
   (b) Divides the *UAS* into $N$ continuous squares;
   (c) For each of the $M$ services, colors the squares within its effective region.

   Figure 5 shows the result of the coloring procedure for the *UAS* in Fig. 4. For a given *UAS*, the coloring procedure only needs to run once after its device position changes.
2. Service combination selection procedure:
   Suppose there are $K$ categories of services required to be migrated. Different devices can provide the same category of services. So we need to find out the best service for each required service category. We denote services in current *UAS* as set $S$, service category set as $C$. Therefore, we have $\forall c \in C, c \subseteq S$ and $\forall s \in S, \exists c \in C,$ s.t. $s \in c$. A subset $S'$ denotes those returned(matching) services. Denote the service category set of $S'$ as $C'$. Then $\forall c' \in C', c' \subseteq S'$ and $\forall s' \in S', \exists c' \in C',$ s.t.$s' \in c'$. Matching degree $d_s, s \in S'$ denotes local matching degree for service $s$. Denote position(square) set as $P$. Then the coverage information for service $s$ and position $p$ can be denoted as $sc[p][s], p \in P, s \in S$. We now suppose context manager has derived matching degrees for the current migration through local service matching procedure. The context manager then performs as follows:

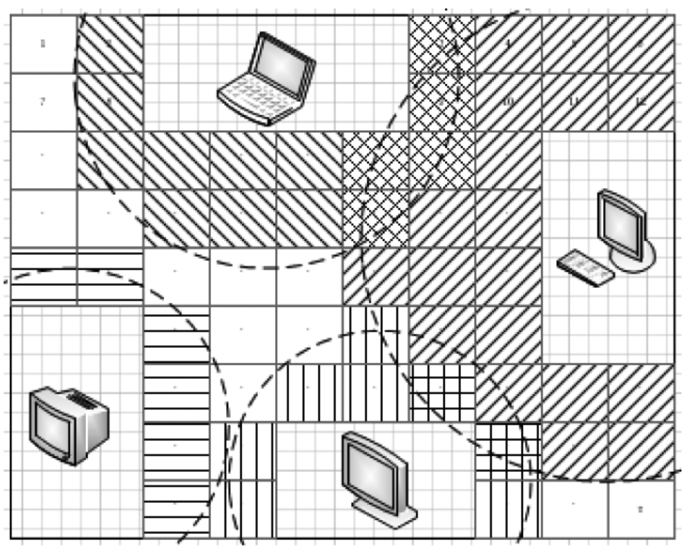

Fig. 5. *Results of the coloring procedure* is represented by continuous squares.

(a) For each square $p (p \in P)$, find the most suitable service $i$ for service category $c$ satisfies:

$$\begin{aligned} s_p[c] &= \max\{EV_i(p) * d_i\}, \text{ s.t.} \\ sc[p][i] &= 1, i \in S', i \in c, c \in C'; \end{aligned} \tag{4}$$

(b) Calculate the overall matching degree for square $p$ by:

$$t(p) = \sum_{c \in C'} (w_c * s_p[c]) \tag{5}$$

$t(p)$ is the overall matching degree for square $p$ and $w_c$ is the weight of the service category $c$. $w_c$ is determined by applications.

(c) Select the square $sel$ with the maximum overall matching degree:

$$sel = \text{argmax}\{t(p)\}; \tag{6}$$

The square $sel$ will be the best position where user can have most effective interaction, and the corresponding services will compose the best service combination.

## 4. Scenario study: Video calls at smart office

In this section, we study a specific scenario, having video calls with a customer at smart office, to illustrate how our context-aware service selection framework works to provide better user experiences.

### *4.1. Scenario description*

#### *4.1.1. The smart office environment*

As smart devices become more and more common in our daily life, the concept of "Smart Office" becomes a hot idea in pervasive computing. Thus we start our scenario study in such a typical smart environment.

One smart office may consist of following devices:

Table 1
Devices and services used in simulation

| Device Name | Service Name | Category |
|---|---|---|
| Projector | projector_display | VO |
| Loudspeaker | loudspeaker_sound | AO |
| Computer01 | comp01_display | VO |
| | comp01_sound | AO |
| | comp01_camera | VI |
| | comp01_micro | AI |
| Computer02 | comp02_display | VO |
| | comp02_sound | AO |
| HD VideoCam | ext_video_camera | VI |
| Wireless Micro | microphone | AI |
| Computer03 | comp03_display | VO |
| | comp03_sound | AO |
| Computer04 | comp04_display | VO |
| | comp04_sound | AO |

VO:Video Output VI:Video Input
AO:Audio Output AI:Audio Input

1. Displays, projectors and etc., categorized as visual device, for the reason that they can output visual images.
2. Loudspeakers, wireless headphones and etc., categorized as audio device, for the reason that they can output audios.
3. Microphones, categorized as voice device, for the reason that they can input voices.
4. Video cameras, categorized as video input devices.
5. Desktop computers, laptops, tablet computer, categorized as compound device since they can provide multiple services.
6. Printers and other irrelative kinds of devices.

Figure 6 shows the layout of our smart office. There are four working blocks in the office with a desktop computer in each block. Computer01 has embedded videocam and microphone. Near computer02, there is an external video camera and a wireless microphone. The devices and corresponding services contained are listed in Table 1.

To construct this smart environment, these devices must satisfy two basic requirements. Firstly, they themselves must be smart, which means that certain computing ability is required so that they can "think" independently; Secondly, they must be connected together, either in wired or wireless way, so that they can communicate with each other to act altogether. Development in mobile and wireless networks has presented several approaches to connect these devices [37].

A smart office should also be aware of status of both users and environment. Thus sensors become its another important part. For examples, heat sensors can be used to collect information of temperatures; surveillance cameras can be used to track users' motion and their face orientations. The data collected by these sensors are called context information.

### *4.1.2. Scenario description*

Suppose there is a user, named Bob, having video call on his smart phone. Due to the small screen of his phone, he may not be satisfied. So he walks into the office which has better devices. In such a scenario, our framework should be able to detect the right intention of Bob, find the devices that can better satisfy Bob's need and migrate interaction to new devices.

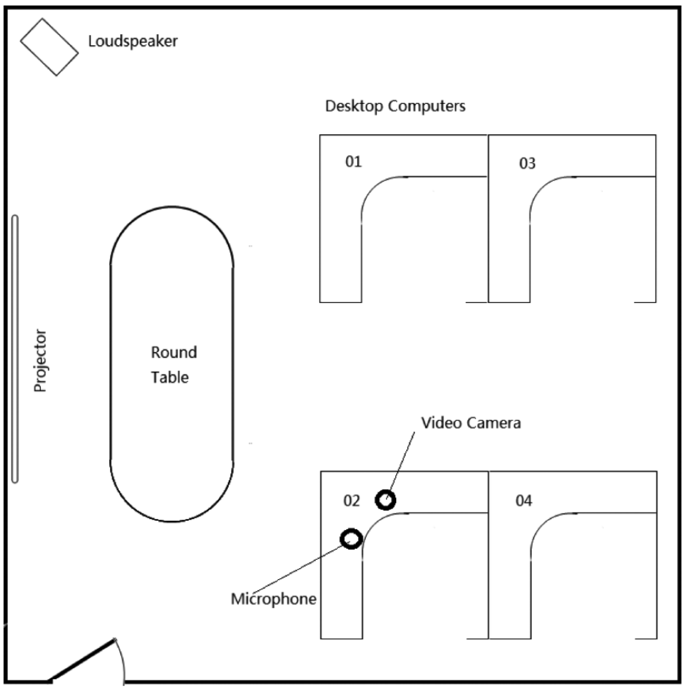

Fig. 6. Layout of the smart office.

## 4.2. *HCI migration request*

HCI Migration Request is sent by user device. User device is a core device in our framework. It can be used to identify the user and decide whether to start an application migration when the user's context is changing. One typical user device may be a smart phone, for the reason that it is portable, easy to obtain user preferences and owns good process ability. People frequently use them to check emails, surf the internet and connect with different devices. All these activities imply users' preferences towards certain services and devices, which can be recorded as *User Preference* by smart phone and facilitate service matching.

After the user device decides to start an HCI migration, requests are sent to context manager and then dispatched to device terminals. The request is basically written in XML format. It is made up by three parts: service description, user preference and user identification information. Service description should contain descriptions of application's related services. These descriptions must contain the essential requirements, namely service properties, so that receivers can decide whether they can offer such services based on these requirements. User preference contains information for devices to decide whether their services can meet the user's need. Finally user identification information is provided to identify who initiates the request.

In this scenario, because Bob is having video call on his smart phone, which is powerless in presenting videos and sound tracks, *Video Output Service*, *Audio Output Service*, *Video Input Service* and *Audio Input Service* are four important service categories to be contained in the migration request. Bob's requirements can be explained as the migration of these four services in our framework. Figure 7 illustrates the structure of migration request within our scenario. Among four service categories, *Video Output Service* is deep investigated. Three properties, format, resolution and bandwidth, denote the

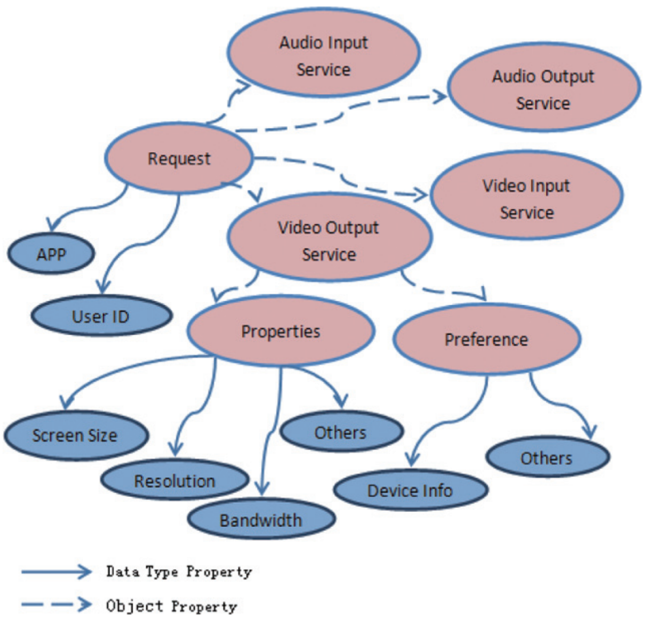

Fig. 7. Structure of service migration request in video call scenario.

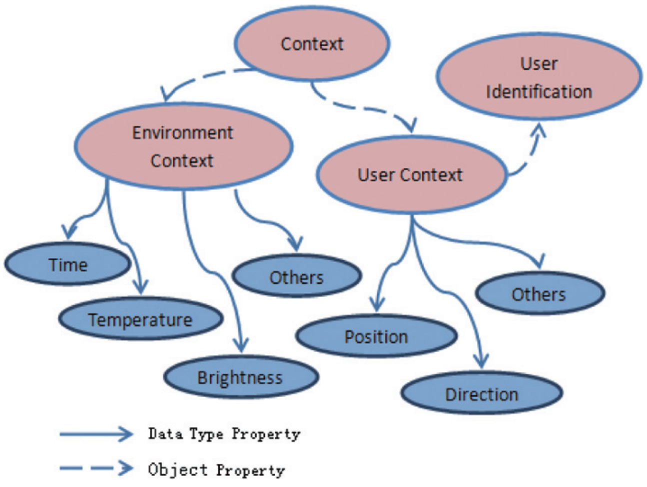

Fig. 8. Structure of context information in video call scenario.

supporting video format, video resolution and transmission bandwidth, respectively. An XML format of this request can be referred to in Appendix A.

## 4.3. *Context format*

As is mentioned above, the context environment is made up by the data collected from different sensors. These data are stored on a local server named "Context Manager", which is responsible for monitoring context information and providing access for interaction devices. Context information consists of two parts: environment and user context. In our scenario, several user contexts need to be considered by our

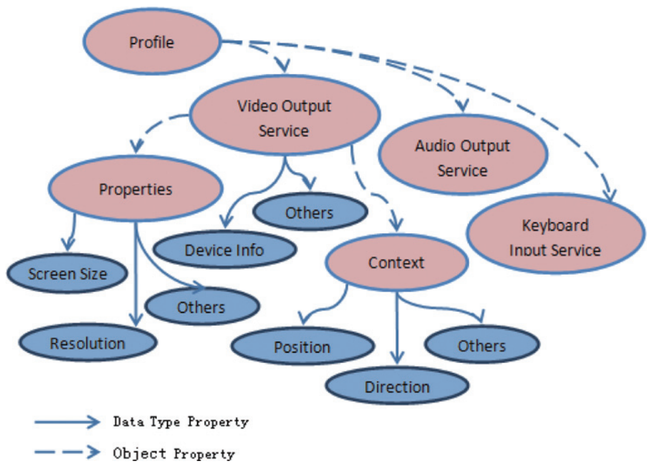

Fig. 9. Structure of device profile of computer02 in video call scenario.

framework such as user position and face orientation, because with these contexts, those devices with video display service whose screens are close or nearly face to user will achieve high matching degree in service matching. The structure of context information retrieved from context manager is relatively simple (as shown in Fig. 8). An XML format can be referred to in Appendix B.

## 4.4. *Device profile*

Device profile is rather essential to the local service matching algorithm as is mentioned in Section 3.4. Each device has their own device description named "Device Profile" to describe their abilities. The profile contains several service descriptions, each of which is used to describe one of the device's features when offering this service. Similar to the format of a request, service description in device profile also contains corresponding parts for matching service properties and user preferences, mentioned in Sections 3.2.1 and 3.2.2. What seems different is that context information is considered in this part. The context information to be concerned may vary with different devices and services. For example, in our scenario, computer02 can receive input from keyboard, display videos and play sound tracks. Then the three abilities are categorized into three different services – *Keyboard Input Service*, *Video Output Service* and *Audio Output Service*. Figure 9 illustrates the schema structure of device profile of computer02 and some essential properties and contexts of Video Output Service are illustrated.

In our scenario, when computer02 receives migration request and context information from context manager, it then starts local service matching process. It first compares the service name and properties between the request and its device profile to know whether it can offer this service. The result is that Video Output Service and Audio Play Service are within matching scope, Keyboard Input Service is unused and Video Input Service and Audio Input Service are not supported. Secondly, for these successfully matched services, device information regarding these services will be compared with user preferences in the request to measure the divergence to the user-expecting device. Finally, computer02 considers context information by calculating distance to the user and cosine similarity between screen and user face orientations. The final matching degree is then back to context manager. After receiving matching degrees, context manager will perform global service combination selection. In the following section, we illustrate the simulation result of this process for our scenario.

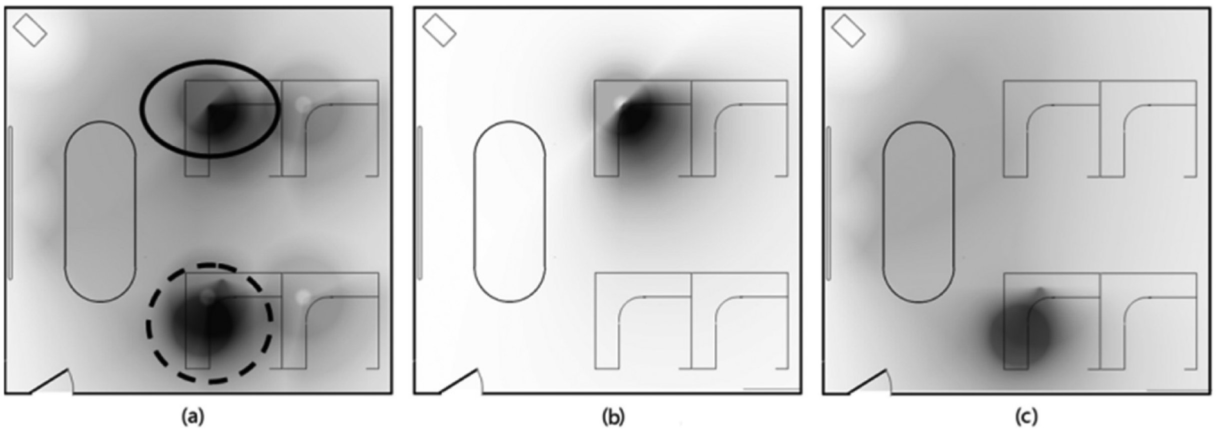

Fig. 10. Overall matching degrees Eq. (5) of each square in the room regarding to different selected services. The room setup can refer to Fig. 6. The room is divided into many squares (not shown due to the large number) and each square is colored by #000000 to #FFFFFF corresponding to the matching degree in that square. Deeper color means higher matching degree. The matching degrees are normalized to $[0, 1]$. (a) Matching degrees for the local optimal service combination of each squares in the room. (b) Matching degrees of each squares for the globally optimal service combination. (c) Matching degrees of each squares using selection algorithm in [1].

## 5. Experiment and result

This section will illustrate simulation results of our context-aware HCI service selection algorithm applying to the smart office scenario discussed in Section 4. We compare our results with the method presented in [1] and show that by modeling context information and user preferences alone is insufficient to discover optimal service combination. Our service selection algorithm which considers interrelations among services is much more effective in searching target service combination for interaction migration. As is mentioned above, we categorize interaction services into several groups and pick the best matches in each groups to form the best service combination regarding a position unit. In this simulation, we predefined four service categories: *Video Output Service*, *Video Input Service*, *Audio Output Service* and *Audio Input Service*. Each devices in the simulation contains several services from one or more service categories. Services provided by different devices may consist of various descriptions or $EV$ functions due to the diversity in device parameters. It is easy to extend this simulation to more complex situation by simply adding customized service descriptions and registering through uniform service interface. The devices and corresponding services contained is illustrated in Table 1.

### 5.1. Simulation result

Our simulation aims at giving solutions to the smart office scenario mentioned in Section 4. Our service selection algorithm first scans each continuous squares in user's *UAS* and find best service combination. Figure 10 shows overall matching degrees Eq. (5) of each square in the room regarding to different selected services. The room setup can refer to Fig. 6. The room is divided into many squares (not shown due to the large number) and each square is colored by #000000 to #FFFFFF corresponding to the matching degree in that square. Deeper color means higher matching degree in that square regarding to the selected services, thereby better user experience of the selected services in that square. The matching degrees are normalized to $[0, 1]$. Figure 10.a shows matching degrees for the local optimal service combination of each squares in the room. The graph shows two potential areas for interaction migration

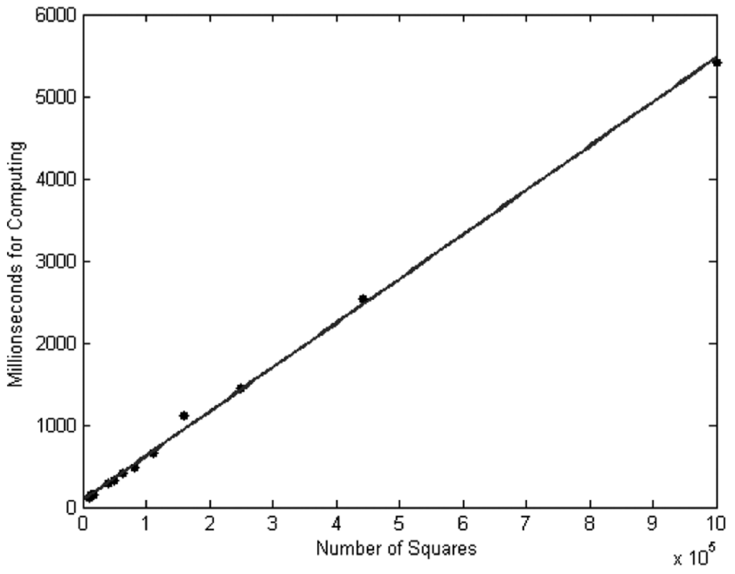

Fig. 11. Computing time for increasing number of squares divided with a fixed number of services.

(marked by solid and dashed circles in Fig. 10.a). The best service combination selected in solid circle is {*comp01_display*, *comp01_sound*, *comp01_camera*, *comp01_micro*}, and the best one in dashed circle is {*comp02_display*, *comp02_sound*, *ext_video_camera*, *microphone*}. It is reasonable to have multiple high-score areas in a *UAS* because different service combinations may satisfy user's migration requests. For final decision of interaction migration, one may lists all those potential service combinations in matching degree order, or just pick the service combination with the highest score.

After assigning local matching degrees to each squares, the best service combination is then picked from the square with highest matching degrees (global selection procedure). In our simulation, the squares with the best matching degrees occur near devices *"Computer01"* whose service combination is {*comp01_display*, *comp01_sound*, *comp01_camera*, *comp01_micro*}. Target device for interaction migration is then *"Computer01"*. Figure 10.b illustrates matching degrees of each squares for the globally optimal service combination. An observation is that the chosen services are close to each other (actually in the same device). This is an ideal situation for interaction migration because users can easily have access to devices and thus increasing user experience.

For algorithm comparison, we apply the service selection algorithm in [1] to this smart office scenario and present the result in Fig. 10.c. The selected services in Fig. 10.c is {*projector_display*, *loudspeaker_sound*, *ext_video_camera*, *microphone*} and matching degrees are shown in colors. Each individual service selected is the best one that matches user's requests (for example, the selected device *"Projector"* is powerful in video output due to its large screen and high resolution). However, the overall matching degrees in the room of the selected service combination is much less than what we select (Fig. 10.b). The reason is our algorithm takes interrelation of services into consideration which benefits discovering good service combination.

## *5.2. Scalability*

Response latency is always a crucial factor in HCI technology. For service-oriented interaction migration, selection algorithm may be a bottleneck for overall migration process. Efficent service selection algorithm needs to be scalable with increasing number of services. In our algorithm, we compute matching degrees for each squares divided in *UAS* by selecting the best services within each service categories. Let the number of service categories be $C$, each service category contains $S$ services and the total number of squares divided in *UAS* be N. The overall complexity of our service selection algorithm is $O(CSN)$. Since $C$ is linearly related to $S$, the computing time grows quadratically in average as more services are added. Another key factor influencing time usage is the number of squares. For a fixed size of area for interaction migration, more squares divided usually mean more accurate result for service selection but more computing costs. Fortunately, the computing time grows linearly with the number of squares (as is shown in Fig. 11). Moreover, the area of a square usually is not necessary to be too small because users may feel no difference within two different squares if they are smaller than users' sensitivity criterion.

## 6. Conclusion

In this paper, we propose a service-oriented framework for interaction migration and provide a context-driven method for service selection under multi-modal environment. Our method extracts user preferences from users' interaction history to describe user-expecting devices when interacting. Moreover, two procedures – local service matching and global combination selection are proposed to discover good combination of interaction services. By integrating user preferences, service and environment contexts, our service matching algorithm is intelligent to handle context changes and provide better user experience. We also present simulation results and evaluation methods. Future work will be done to improve continuity of migrated interaction and conflict resolution for busy devices. We are also interested in applying more flexible service description format to support more accurate semantics and user-customizing service types.

## Acknowledgments

This work was supported by the National Natural Science Foundation of China (NSFC) with Grant Nos. 61003219 and 61073148, and the Doctoral Fund of Ministry of Education of China under Grant 20100073120022.

## Appendices

### *A. Request Format in XML*

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<xsd:element name=" Request " />
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="App" type="xsd:string"/>
           <xsd:element name="UserID" type="xsd:string"/>
      <xsd:element ref="Video Output Service " />
```

```
           <xsd:element ref="Audio Output Service " />
           <xsd:element ref="Video Input Service " />
           <xsd:element ref="Audio Input Service " />
    </xsd:sequence>
    </xsd:complexType>
</xsd:element>
<xsd:element name=" Video Output Service" />
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref=" Properties" />
      <xsd:element ref=" Preferences " />
        </xsd:sequence>
      </xsd:complexType>
</xsd:element>
<xsd:element name=" Properties " />
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="Format" type="xsd:string"/>
      <xsd:element name="Resolution" type="xsd:string"/>
      <xsd:element name="Bandwidth" type="xsd:string"/>
      <xsd:element name="Others" type="xsd:string"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
<xsd:element name=" Preferences " />
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="Device Info" type="xsd:string"/>
      <xsd:element name="Others" type="xsd:string"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
</xsd:schema>
```

*B. Context Format in XML*

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<xsd:element name=" Context " />
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="Environment Context"/>
      <xsd:element ref="User Context"  />
        </xsd:sequence>
    </xsd:complexType>
```

```
</xsd:element>
<xsd:element name=" Environment Context" />
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="Time " type="xsd:string"/>
      <xsd:element name="Temperature" type="xsd:string"/>
      <xsd:element name="Brightness" type="xsd:string"/>
      <xsd:element name="Others" type="xsd:string"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
<xsd:element name=" User Context" />
  <xsd:complexType>
    <xsd:sequence>
      <xsd:attribute ref="User Identification"/>
      <xsd:element name="Position" type="xsd:string"/>
      <xsd:element name="Orientation" type="xsd:string"/>
      <xsd:element name="Others" type="xsd:string"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
</xsd:schema>
```

*C. Device Profile in XML*

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<xsd:element name=" Profile" />
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="Video Output Service" />
      <xsd:element ref="Audio Output Service" />
      <xsd:element ref="Keyboard Input Service" />
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
<xsd:element name="Video Output Service" />
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="Properties" />
          <xsd:element ref="Context" />
        </xsd:sequence>
  </xsd:complexType>
</xsd:element>
<xsd:element name=" Properties " />
```

```
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="Others" type="xsd:string"/>
        </xsd:sequence>
  </xsd:complexType>
</xsd:element>
<xsd:element name=" Context " />
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="Position" type="xsd:string"/>
      <xsd:element name="Orientation" type="xsd:string"/>
      <xsd:element name="Others" type="xsd:string"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
</xsd:schema>
```

## References

[1] M. Wang, X. Tang, Y. Shen and M. Guo, A method of context-driven HCI service selection in multimodal interaction environments, In *Proceedings of the Second International Symposium on Frontiers in Ubiquitous Computing, Networking and Applications*, 2011.

[2] M. Weiser, The computer for the 21st century, *Scientific American* **265**(3) (1991), 94–104.

[3] R. Bandelloni and F. Paternò, Flexible interface migration, in: *IUI*, J. Vanderdonckt, N.J. Nunes and C. Rich, eds, ACM, 2004, pages 148–155.

[4] A. Sarmento, *Issues of Human Computer Interaction*, Irm Pr, 2005.

[5] Y. Shen, M. Wang and M. Guo, Towards a web service based HCI migration framework, In *Proceedings of the 6th International Conference on Embedded and Multimedia Computing*(*EMC-11*), 2011.

[6] Paterno, Fabio, *Model-Based Design and Evaluation of Interactive Applications*, Applied Computing. Springer-Verlag, 1999.

[7] K. Luyten, J. Van den Bergh, C. Vandervelpen and K. Coninx, Designing distributed user interfaces for ambient intelligent environments using models and simulations, *Computers and Graphics* **30**(5) (2006), 702–713.

[8] F. Paternò, C. Santoro and A. Scorcia, A migration platform based on web services for migratory web applications, *J Web Eng* **7**(3) (2008), 220–228.

[9] R. Chinnici, J.-J. Moreau, A. Ryman and S. Weerawarana, Web services description language (WSDL) version 2.0 part 1: Core language. World Wide Web Consortium, Recommendation REC-wsdl20-20070626, June 2007.

[10] R. Khalaf, N. Mukhi and S. Weerawarana, Service-oriented composition in bpel4ws, In *WWW* (*Alternate Paper Tracks*), 2003.

[11] S.A. McIlraith, T.C. Son and H. Zeng, Semantic web services, *IEEE Intelligent Systems* **16**(2) (2001), 46–53.

[12] D.L. Martin, M. Paolucci, S.A. McIlraith, M.H. Burstein, D.V. McDermott, D.L. McGuinness, B. Parsia, T.R. Payne, M. Sabou, M. Solanki, N. Srinivasan and K.P. Sycara, Bringing semantics to web services: The owl-s approach. In *SWSWPC*, 2004, pages 26–42.

[13] M. Klusch, B. Fries and K.P. Sycara, Automated semantic web service discovery with owls-mx. In *AAMAS*, 2006, pages 915–922.

[14] A. Bandara, T.R. Payne, D. De Roure and T. Lewis, A semantic framework for priority-based service matching in pervasive environments, In *OTM Workshops* (*2*), 2007, pages 783–793.

[15] Z. Wu, Q. Wu, H. Cheng, G. Pan, M. Zhao and J. Sun, Scudware: A semantic and adaptive middleware platform for smart vehicle space, *IEEE Transactions on Intelligent Transportation Systems* **8**(1) (2007), 121–132.

[16] S.B. Mokhtar, A. Kaul, N. Georgantas and V. Issarny, Efficient semantic service discovery in pervasive computing environments, In *Middleware*, 2006, pages 240–259.

[17] D. Chakraborty, A. Joshi, Y. Yesha and T.W. Finin, Toward distributed service discovery in pervasive computing environments, *IEEE Trans Mob Comput* **5**(2) (2006), 97–112.

[18] A.K. Dey, *Providing architectural support for building context-aware applications*, PhD thesis, Atlanta, GA, USA, 2000. AAI9994400.
[19] B. Medjahed and Y. Atif, Context-based matching for web service composition, *Distributed and Parallel Databases* **21**(1) (2007), 5–37.
[20] C. Lee and A. Helal, Context attributes: An approach to enable context-awareness for service discovery, In *SAINT*, 2003, pages 22–30.
[21] P. TalebiFard and V. CM Leung, Context-aware mobility management in heterogeneous network environments, *Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications* **2**(2) (2011), 19–32.
[22] F. Morvan and A. Hameurlain, A mobile relational algebra, *Mobile Information Systems* **7**(1) (2011), 1–20.
[23] S. Gao, J. Krogstie and K. Siau, Developing an instrument to measure the adoption of mobile services, *Mobile Information Systems* **7**(1) (2011), 45–67.
[24] L. Zeng, B. Benatallah, A.H.H. Ngu, M. Dumas, J. Kalagnanam and H. Chang, Qos-aware middleware for web services composition, *Software Engineering, IEEE Transactions on* **30**(5) (May 2004), 311–327.
[25] R. Aggarwal, K. Verma, J. Miller and W. Milnor, Constraint driven web service composition in meteor-s. In *Services Computing, 2004. (SCC 2004). Proceedings. 2004 IEEE International Conference on*, Sept. 2004, pages 23–30.
[26] J. Xiao and R. Boutaba, Qos-aware service composition and adaptation in autonomic communication, *Selected Areas in Communications, IEEE Journal on* **23**(12) (Dec. 2005), 2344–2360
[27] H. Yu and S. Reiff-Marganiec, Automated context-aware service selection for collaborative systems. In Pascal van Eck, Jaap Gordijn, and Roel Wieringa, editors, *Advanced Information Systems Engineering*, volume 5565 of *Lecture Notes in Computer Science*, pages 261–274. Springer Berlin/Heidelberg, 2009. 10.1007/978-3-642-02144-2_23.
[28] K. Luyten, C. Vandervelpen, J. Van den Bergh and K. Coninx, Context-sensitive user interfaces for ambient environments: Design, development and deployment, in: *Mobile Computing and Ambient Intelligence: The Challenge of Multimedia*, Nigel Davies, Thomas Kirste and Heidrun Schumann, editors, number 05181 in Dagstuhl Seminar Proceedings, Dagstuhl, Germany, 2005. Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany.
[29] K. Luyten, C. Vandervelpen and K. Coninx, Task modeling for ambient intelligent environments: design support for situated task executions, In *Proceedings of the 4th international workshop on Task models and diagrams*, TAMODIA ’05, New York, NY, USA, 2005. ACM, pages 87–94.
[30] H.H. Hsu and C.C. Chen, Rfid-based human behavior modeling and anomaly detection for elderly care, *Mobile Information Systems* **6**(4) (2010), 341–354.
[31] A. Jaimes and N. Sebe, Multimodal human-computer interaction: A survey, *Computer Vision and Image Understanding* **108**(1–2) (2007), 116–134.
[32] E. Hjelmås and B.K. Low, Face detection: A survey, *Computer Vision and Image Understanding* **83**(3) (2001), 236–274.
[33] B. Fasel and J. Luettin, Automatic facial expression analysis: a survey, *Pattern Recognition* **36**(1) (2003), 259–275.
[34] J.K. Aggarwal and Q. Cai, Human motion analysis: A review, *Computer Vision and Image Understanding* **73**(3) (1999), 428–440.
[35] A.T. Duchowski, A breadth-first survey of eye-tracking applications, *Behav Res Methods Instrum Comput* **34**(4) (November 2002), 455–470.
[36] Y.H. Han, C.M. Kim and J.M. Gil, A greedy algorithm for target coverage scheduling in directional sensor networks, *Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications* **1**(2/3) (2010), 96–106.
[37] I. You and T. Hara, Mobile and wireless networks, *Mobile Information Systems* **6**(1) (2010), 1–3.

---

**Yao Shen** received his Ph.D. degree in computer science from Shanghai Jiao Tong University, China in 2008. Thereafter he joined Shanghai Jiao Tong University and became an assistant researcher of the Department of Computer Science and Engineering. His research interests include pervasive computing, wireless networks and distributed computing.

**Minjie Wang** obtained his B.S. degree in computer science from Shanghai Jiao Tong University, China. Currently, he is a master student in Embedded and Pervasive Computing Research Center in the same university. He is interested in pervasive computing, cloud computing, distributed systems and human-computer interaction.

**Xiaoxin Tang** received his B.S. degree in computer science and technology from South China University of Technology, China, in 2010. Thereafter, he became a PhD student at the Department of Computer Science and Engineering, Shanghai Jiao Tong University, China. His main research activities concern the design of algorithms and architectures for HCI services. Other interested research areas include parallel and high performance computing.

**Yi Luo** is a technical president of China Xinhua Network Co. Ltd (Xinhuanet), responsible for the technical work of Xinhuanet. He received the Master degree in Computer Science from Tsinghua University. His research interests include Internet computing, wireless and mobile computing, and distributed systems. As the technical principal, Yi Luo creatively developed and applied many new technologies, for example, Panguo research and Clouding Computing, in China Xinhuanet to support the online important government and society news.

**Minyi Guo** received his Ph.D. in computer science from the University of Tsukuba, Japan. Before 2008, he had been a Research Scientist of NEC Corp., Japan, and professor of University of Aizu, Japan. He is now the Head and Distinguished Professor of Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai, China. He has published more than 250 papers in international journals and conferences. His research interests include parallel and distributed processing, parallelizing compilers, pervasive computing, embedded software optimization, molecular computing, and software engineering. He is a senior member of IEEE, a member of the ACM, IPSJ and IEICE.