Chapter IX
UbiSrvInt:
A Context-Aware Fault-Tolerant Approach for WP2P Service Provision

Soe-Tsyr Yuan
National Chengchi University, Taiwan

Fang-Yu Chen
AsusTeK Computer Inc., Taiwan

ABSTRACT

Peer-to-Peer applications harness sharing between free resources (storage, contents, services, human presence, etc.). Most existing wireless P2P applications concern merely the sharing of a variety of contents. For magnifying the sharing extent for wireless service provision in the vicinity (i.e., the wireless P2P environments), this chapter presents a novel approach (briefly named UbiSrvInt) that is an attempt to enable a pure P2P solution that is context aware and fault tolerant for ad-hoc wireless service provision. This approach empowers an autonomous peer to propel distributed problem solving (e.g., in the travel domain) through service sharing and execution in an intelligent P2P way. This approach of ad-hoc wireless service provision is not only highly robust to failure (based on a specific clustering analysis of failure correlation among peers) but also capable of inferring a user's service needs (through a BDI reasoning mechanism utilizing the surrounding context) in ad-hoc wireless environments. The authors have implemented UbiSrvInt into a system platform with P-JXTA that shows good performance results on fault tolerance and context awareness.

INTRODUCTION

In recent years, new services have mushroomed all over the web world, and people can easily attain a great number of services from the Internet. A service usually performs in the role of computation facility or information provider. Popular examples include search services, agent services, entertainment services, transaction services, *etc*. Service composition then refers to the technique of creating complex services with the help of smaller, simpler and easily executable lightweight services or components (Chakraborty, 2001). That is, we can handily create novel, interesting and customized services by bundling existing services together to meet the demands of our customers.

On the other hand, mobile devices are in widespread use now, and myriad mobile ad hoc networking technologies (e.g., Bluetooth, IEEE 802.11) unfold dramatically. Clever design of mobile devices includes dramatically reduced size, enlarged storage, economic power consumption and accelerated CPU speed. This design not only improved the performance but also advanced the functionality of the mobile devices. The overwhelming majority of mobile devices launched recently are all capable of supporting wireless Internet access as one of their key features. The next era of network enables the integration of various heterogeneous networks and makes it possible for people to surf between them through different kinds of wireless device anytime, anywhere and anyway. People are striding forward to a completely new Wireless Age.

Accordingly, it can be envisioned that in the forthcoming future everyone (who is walking on the street, dining in the restaurant or working in the office) outfits with hand-held or wearable mobile devices as the standard equipments to access any nearby available network for wanted services. As you move around, a software agent residing in your wireless devices autonomously searches and collects information about what is available from your current location. You may carry with you some useful lightweight services downloaded from the Internet or any wayside provisioning server. You may provide services on hand for nearby people who need them and equally attain desired services from nearby people who possess them. You may, moreover, compose those available wireless services to form an aggregated service tailoring to your contextualized needs, exhibiting moment of values of the services. In other words, the demand to create novel functionalities out of composing wireless services in the vicinity is extremely indispensable.

The aforementioned envisions manifest the significance of the problem of wireless service provision that aims for providing contextualized customized services to meet the concrete needs or requirements of a given client who is equipped with wireless mobile devices by utilizing resources available in its vicinity.

Wireless service provision in the vicinity requires a certain service platform installed at the side of mobile devices. Most existing service platforms (Casati et. Al., 2002) (Mao et. al., 2001) (Mennie et. Al., 2000) (Schuster et. al., 2000) (Gribble et. al., 1999) have been designed on a wired environment that is of high stability and bandwidth, performing against the nature of ad hoc networks. Furthermore, their centralized approaches exerted for service provision have their innate drawback while transplanting them to the wireless environment. The drawback is three-fold:

- **Fault-tolerance:** In centralized architectures, if the server shuts down, everything else does as the server is the central point of failure.
- **Scalability:** The scalability is limited to the capacity of the central server. Should a large amount of requests be addressed to the server, the server easily becomes the bottleneck of traffic.

- **Extensibility:** Centralized architectures are also often hard to expand owing to the limited resources of the central server.

There have been a few published researches (Benatallah et. al., 2002) (Chakraborty et. al., 2002) (Sheng et. al., 2002) addressing the problem using decentralized P2P approach recently. However, when applied to wireless service provision they encountered certain problems mainly resting on the employment of the mediator (broker or coordinator) technique. This hybrid P2P architecture has drawbacks similar to centralized architectures. Exemplars of the hybrid P2P drawback primarily rest on existence of centralized nodes:

- **Quality of wireless connection:** This refers to the poor condition of the connection between mobile devices and the central node (e.g., intermittent disconnection and transmission latency). Information replied from a central node may take too long to reach the mobile device that originates the request. The client subsequently cannot attain the desired information in time, but be bothered by stale and useless information.
- **Real-time information:** Another question is about information updating. Information is unlikely to be always up-to-date on the central node in a dynamically changing environment, such as traffic information.
- **Infeasibility of mobile super peer:** Supposing a super peer in hybrid P2P architectures can be mobile, the aforementioned problem can be partially resolved. However, this leads to other problems. Qualified mobile devices (providing extraordinary computing power, storage capacity and sufficient bandwidth to take charge of a server's duty) are very uncommon in reality. Super-peers, if any, at proper place in proper time are not always reachable from all mobile devices.

Accordingly, (semi) centralized approaches cannot be served as a good solution to compose wireless services on the move. A better solution to wireless service provision is believed to have the duty segmented and delegated to peers (who are willing to and able to execute the proportioned duties, and bringing about the desired properties of salability and extensibility). That is, *a pure P2P solution could be further explored so as to unfold alternative forms of wireless service provision via* mobile ad hoc networking technologies (e.g., Bluetooth, IEEE 802.11).

Yet another inappropriateness of existing decentralized service provision architecture is that they did not take into account the issues of fault tolerance and context awareness. These two issues however are crucial especially dealing with mobile devices within wireless environments. The reasons are two folds: (1) Unreliability of between-peer wireless connection often results in unavailability of services. (2) Mobility of peers often engenders changes in user contexts and accordingly causes different needs. *This chapter aims to provide an approach for P2P mobile service provision, which is not only highly robust to failure but also keenly aware of the surrounding context in wireless environments.*

In this chapter, we present an approach named UbiSrvInt (abbreviation of Ubiquitous Services Integration) that is a pure P2P solution for wireless service provision that has the salient features of fault tolerance and context awareness (that are further described as follows):

- **Fault tolerance:** There are several solutions for handling system failures. The most common way is to re-execute it. It is indeed simple but very inefficient and not applicable in wireless mobile environments. The existing solutions (Chakraborty et. al, 2002) (Dialani et. al., 2002) to this problem for distributed service-based architecture are to employ checkpoints to guard against such faults. However, this method increases

the traffic overhead of propagating checking message. It is too complex and only operable while using process-based service description. Such mechanisms adhered to service composition systems primarily rest on post-failure recovery. *In this chapter, a foresighted mechanism is exerted in the approach so as to improve the efficiency by delegating service execution to low-fault correlation peers.* Hence, it can reduce the percentage of failure taking place. Furthermore, the elimination of the central node can make the design of the system immune to single point of failure.

- **Context awareness:** Context awareness refers to the capability of adapting the involved decisions in accordance with the current user context and thus it is one of the most important preferred features from a mobile user's perspective (as mobile situations of the user change over time). In our approach, each peer acts as an autonomous entity whose behavior is governed and adapted by its beliefs, desires and intentions that are captured from user profiles and real-time contextual information in user's vicinity such as time, location, weather and so on. Thereby peers have the ability to reason and help users to get the right services in the right place at the right moment.

We have implemented *UbiSrvInt that is to be installed on each peer so as to realize pure P2P wireless service provision in certain application domains* (e.g., travel services, museum services, *etc.*). With UbiSrvInt functioning at each peer, the peer can avail itself of the available services of the peers in the vicinity, in a self-organized robust intelligent way. *UbiSrvInt is unique in its combined consideration of context awareness and fault tolerance for P2P-based customized service provision in unreliable mobile ad-hoc networks.*

The remainder of this chapter is organized as follows: Section 2 presents the contextualized fault-tolerant approach for P2P mobile service composition. A brief description of the implementation of UbiSrvInt is then provided in Section 3. Section 4 provides the performance evaluation of UbiSrvInt. Finally, Section 5 concludes this chapter with future fruitful research.

UBISRVINT

The UbiSrvInt approach serves as the foundation (equipped in each mobile device) upon which P2P mobile services are discovered, executed, and composed with a pure P2P interaction model. It is a general-purpose approach attempting to support a large cross-section of P2P mobile services. This section is unfolded with a description of the basic concepts (Section 2.1) followed by the detailed descriptions of the approach components (Section 2.2) (but with a strong emphasis on the component that is in charge of fault tolerance as addressed in Section 2.3).

Basic Concepts

The functionalities of the approach can roughly be structured into four layers shown in Figure 1 (a detailed structure of the approach will be shown in

Figure 1. Basic concepts of UbiSrvInt

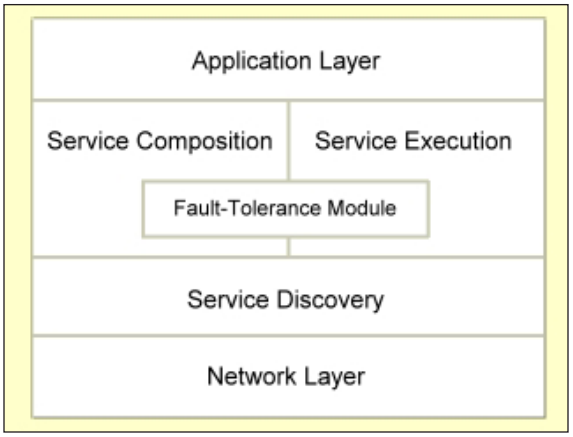
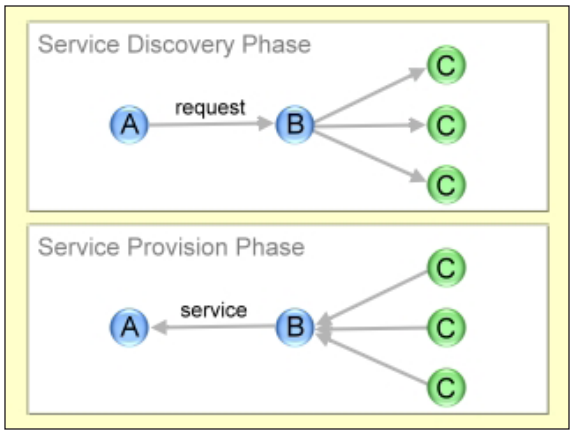


Figure 3). The network layer concerns controlling the routing of messages, masking the differences in characteristics of different transmission and sub-network technologies to provide wireless transparent transfer of data between peers. The service discovery layer is responsible for discovering the inferred services that are available nearby. With above discovered services as the inputs, the service composition layer carries out the integration of those services in a feasible order and ties in with the whole process of discovery. The service execution layer takes charge of the execution of assigned service components to yield services. The top of the approach is the application layer. It encapsulates different GUI facilities to serve as the means for the users to access different composite services.

One of the important characteristics of the approach is the use of a fault-tolerance module that provides capabilities for preventing failures in advance and recovering failures once any component fails. There are many useful ways to improve dependability, however, redundancy is the simplest technique typically employed in P2P systems. *Due to the time criticality feature of the P2P mobile services, physical redundancy is believed to be the most appropriate redundancy approach (as opposed to the time redundancy approach), replicating additional service components so as to assure the continued composite service in case the crash occurring to some of the service components.*

Furthermore, the probability of a wireless mobile peer failing while a transaction is in progress is dependent upon several factors such as network routings, access points, operating system types and releases, device brands, service types, and user behaviors. Therefore, all peers have their respective probability distributions of failures, which may be mutual correlated. *In UbiSrvInt fault tolerance is achieved through dispatching multiple replications of service components to different peers in the vicinity that fail independently.*

Figure 2. Triple component physical redundancy



For instance, when peer A subscribes the service produced by peer B, a set of peers that fail with low correlation are delegated to produce the service required by peer A (as shown in Figure 2). In the example of Figure 2 this mechanism provides threefold active replications of the service component to prevent a single component failure. Peer A only interacts with peer B. Peer B handles peer A's request and sends back the service. Meanwhile, peer B replicates the component and dispatches them to peer set Cs. Each peer C executes the allotted component and returns the result of execution to peer B.

There are two possible worlds of exploiting this physical-redundancy concept: (1) The process of service provision may be interrupted because the provider crashes and consequently halts service or the provider omits to respond to incoming requests (as addressed in *fail-silent* (Powell et. al., 1988) or *fail-stop* (Schlichting et. al., 1983)). In this situation the faulty peer stops functioning and produces no ill output. For instance, peer A sends its request to peer B which handles the request and delegates peer set Cs to execute service. Peer B waits only for the first reply and returns it to peer A. (2) Unlike the first situation that assumes all peers are harmless and no incorrect response, the second situation allows the occurrence of the reality where peer sometimes continues to operate

but produces wrong results to output (as addressed in *Byzantine faults* (Lamport et. al., 1982), which is obviously more troublesome to deal with). For instance, peer A sends its request to peer B which handles the request and delegates peer set Cs to execute the service. Peer B then acts as voter in this world, picks the majority winner of the three inputs obtained from peer set Cs, and returns the voting result to peer A.

The Architecture of UbiSrvInt

UbiSrvInt is composed of several components (as shown in Figure 3). As follows show what each component does and how the components interact with each other in the architecture: (1) Reasoning Agent collects context information of its surroundings from an external context handling system (e.g., provisioning server or context provider) and a user profile so as to reason rationally for furnishing the user with the tasks of adaptive service requests. (2) Sub-Tasking Agent is responsible for segmenting each task of service request into several subtasks. The sub-tasking information (i.e., the knowledge of the subtasks required to execute a particular task) is presumed to be attained from external systems (e.g., provisioning servers) or other peers. (3) Discovery Agent is in charge of discovering the services conforming to the functionality listed on the subtasks list of Sub-Tasking Agent. (4) Composition Agent receives services acquired by Discovery Agent and integrates them into a composite service in a proper order (based on certain task-related knowledge to perform service composition). (5) Execution Agent then works on service execution that would involve Fault-Tolerance Module (FTM). (6) FTM is the focus of our approach that provides capabilities for preventing failures in advance and recovering failures once any service component fails.

The underlying assumptions behind the approach are three folds: (1) Heterogeneous devices can effortlessly connect to each other through

Figure 3. The picturesque view of the UbiSrvInt infrastructure

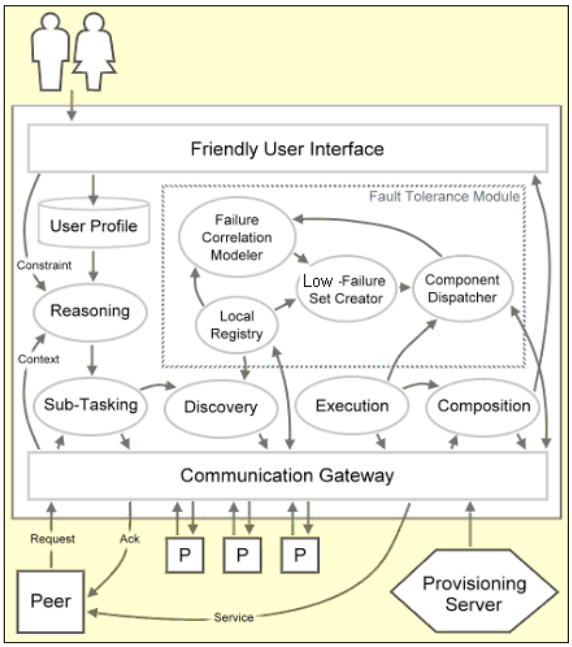
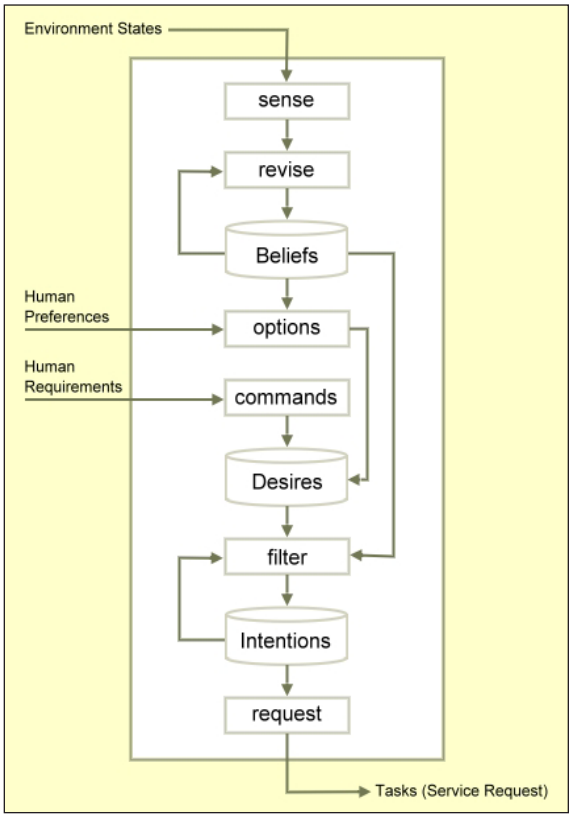


Figure 4. The BDI reasoning process



the component of Communication Gateway of the infrastructure. (2) Mobile services can be unfolded on the heterogeneous devices through transformation between different forms. The component descriptions of UbiSrvInt are then detailed in the following subsections.

Reasoning Agent

Reasoning is an implement to achieve the goal of taking possible courses of actions to provide adaptive services. The Reasoning Agent follows the belief-desire-intension (BDI) conception (Bratman et. al., 1988) and is responsible for receiving and interpreting service requests for its user, as shown in Figure 4.

The Reasoning Agent collects *contextual surrounding information* either from an external context handling system (e.g., provisioning server or context provider) or from a user profile in order to reason rationally. The types of context information can be identified into five categories: (1) Environmental contexts (date, time, location, nearby people, weather, light, humidity, altitude, velocity, etc.) (2) Informational contexts (news, traffic, transportation timetable, movie timetable, stock prices, sports scores, weather forecasts, etc.) (3) Personal contexts (sex, age, personality, health, mood, behaviors, schedule, activity, preference, economic status, etc.) (4) Social contexts (relationships, experiences, appointments, community activity, accessible personal contexts of people, etc.) (5) Resource contexts (network bandwidth, status of nearby printers, battery power, available services, etc.)

A variety of different context information can be gathered by external context handling systems (e.g., a sideway provisioning server available at a museum entrance) that often outfit sensors for collecting context information, such as environmental contexts, informational contexts and some resources contexts. As to personal contexts and social contexts, they are sensitive and therefore should be protected from strangers or unfamiliar friends and thus retained in the user profiles at the side of mobile devices. Such kinds of information are only conveyed between peers according to the level of their relationships in order to keep the privacy of users. The P2P infrastructures enable the possibility of ubiquitous computing of all kinds of context information in contrast to a centralized approach, such as Client-Server infrastructures.

From the aspect of peers, the various context information is gained from outside except personal contexts, hence we generally called them *environment states* to distinguish from the contexts gained from inside. We assume that the state of a peer's environment can be characterized as a set of *environment states* $S = \{s_1, s_2, ...\}$. At any moment, the environment is in one of these states. The function *sense* represents the peer's ability to observe its environment and attain the environment information, outputting a percept. The perception of the peer is assumed to be represented by a non-empty set $P = \{p_1, p_2, ...\}$ of *percepts*. The *sense* function subsequently is formulated as follows:

$$sense: S \rightarrow P,$$

which maps environment states to percepts. The environment information is regarded as a true *belief* of the peer that will change over time. Let *Bel* be the set of all possible beliefs and the peer's belief revision function is a mapping that determines a new set of beliefs from the current percept and current beliefs:

$$revise: \wp(Bel) \times P \rightarrow \wp(Bel)$$

Note that the symbol $\wp$ denotes the powerset, the set of possible subsets, of any given set.

The user can directly invoke specific service requests with certain constraints or conditions. The requirements made by the user are defined as a set *Hr* of *human requirements*. Let *Des* be the set of all possible desires. The peer's *desire* can then be directly declared as shown in the *commands*

function that receives the requirements from the user and then output the desires:

$commands: Hr \rightarrow \wp(Des)$.

Alternatively, the peer's desire can be indirectly derived from the user profile information. That is, Reasoning Agent can attain the information of the user's *preference* from the User Profile. The user's preference is defined as a set *Hp* of *human preferences*. Reasoning Agent subsequently can infer what the *options* available to the user are when there is no clear and definite instruction from outside. The option generation function, *options*, maps a set of human preferences and a set of beliefs to a set of desires:

$options: \wp(Bel) \times \wp(Hp) \rightarrow \wp(Des)$

The chosen options or commands become intentions, which then constitutes the formal service requests. Intentions are then fed back into the agent's future practical reasoning. Let *Int* be the set of all possible intentions. In order to make the final decision, the Reasoning Agent's deliberation process is represented in the *filter* function:

$filter: \wp(Bel) \times \wp(Des) \times \wp(Int) \rightarrow \wp(Int)$

which updates the agent's intentions on the basis of its previously-held intentions and current beliefs and desires. As the result of deliberating, it infers user's *intension* and outputs it as a formal service request for Sub-Tasking Agent. The service requests are regarded as *tasks,* which will be allocated afterward. The generated tasks are defined as a set of *tasks T.* The *request* function is assumed to simply return any executable intentions in a priority:

$request: \wp(Int) \rightarrow T,$

in which each intention is correspondent to a directly executable task. The state of Reasoning Agent at any given moment is, deservedly, a triple (B, D, I), where $B \subseteq Bel$, $D \subseteq Des$, and $I \subseteq Int$.

Sub-Tasking Agent

Sub-Tasking Agent receives the task information from Reasoning Agent and subsequently decides how to segment the task appropriately. That is, Sub-Tasking Agent is responsible for dividing a generic task into several subtasks.

Sub-Tasking Agent attains the sub-tasking information (i.e., the different subtasks required to execute a particular task) from an external system, such as Provisioning Server or other peers. We assume that the task-related knowledge can be characterized as a set *Kh* of all given *know-how* about sub-tasking a task. Subsequently, it generates a subtask list served as the input to Discovery Agent. Let *Ts* be the set of all possible subtasks, in which ts_i denotes *subtask* i of task *T.* The generated subtask list is represented as

$T = [ts_1, ts_2, ts_3, ..., ts_N]$

The service sub-tasking function, *decompose*, of the Sub-Tasking Agent is a function

$decompose: T \times \wp(Kh) \rightarrow \wp(Ts),$

which maps the current task and corresponding know-how to determine a feasible set of subtasks.

There are two possibilities for the Sub-Tasking Agent to execute *decompose* function. The first situation is that Sub-Tasking Agent gets internal service request from Reasoning Agent. This kind of service requests will then be considered as initiating tasks. The other situation is that Sub-Tasking Agent works on task decomposition for external service request from other peers. In this situation it is Reasoning Agent that should decide whether the request can be handled or not. The factors that affect this decision are as follows: *user settings, remaining power, resource capabil-*

ity (e.g. computing capability, available storage, bandwidth, etc.), and number of tasks in hand. The capability of each peer can be represented as a capability vector *CV* of the form

$$\aleph \equiv CV(e_1, e_2, ..., e_M), e_i \in [0,1]$$

using the symbol $\aleph$, in which e_i denotes the standard normalized score of factor i. Let *J* be the set of two possible results of judgment, yes or no. Then the function *judge* is a mapping

$$judge: T \times \aleph \rightarrow J, J \in \{0,1\}$$

If the judgment result is positive, Sub-Tasking Agent will perform decomposition of the external service request to generate subtasks. Afterward, Composition Agent will execute the partial composition of those subtasks, which is introduced later.

Discovery Agent

In a WP2P environment mobile peers can share with each other nearby myriad types of resources (e.g., storage, cycles, contents, services and human presence). The locations of nearby peers are represented by a set of *peer locations* $L = \{l_1, l_2, ...\}$. Since peers can join and leave the environment at any time, the unpredictable movement makes resources discovering a challenge.

In this research services are the primary resource type concerned for the purpose of wireless service provision. A service is referred as the execution result of a service component of a peer. Let *Srv* be the set of all possible services. Discovery Agent is responsible for discovering the services conforming to the functionalities listed in a subtasks list $[ts_1, ts_2, ts_3, ..., ts_N]$. The service discovery function, *lookup*, of Discovery Agent is a mapping as follow:

$$lookup: Ts \rightarrow \wp(L),$$

in which Discovery Agent gets relevant peer location information from Local Register. The detail of Local Register will be described later. The service acquirement function, *acquire*, is a mapping as follows:

$$acquire: Ts \times L \rightarrow Srv,$$

which attains a service (i.e., just a look-up) via a remote call to an external nearby peer. The service will subsequently be delivered to Composition Agent immediately.

Composition Agent

Composition Agent is responsible for managing the order of integration of the discovered services. It receives services acquired by Discovery Agent and integrates them into a composite service in a proper order. Like Sub-Tasking Agent, Composition Agent requires task-related knowledge to perform service composition. The composite service is represented by a set named *CSrv* of *composite services*. The service composition function, *compose*, is now a function

$$compose: \wp(Srv) \rightarrow CSrv,$$

which produces an aggregate service from several pieces of input services.

Execution Agent

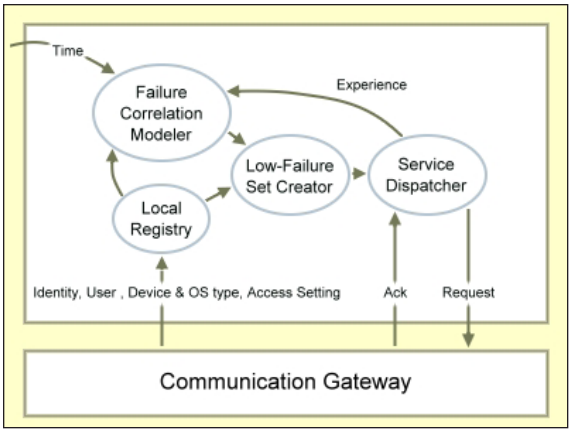
The Execution Agent as implied by its name is responsible for service execution. The process involves the FTM, which is the core of UbiSrvInt (and will be detailed in Section 3). If a peer plays the role of a service dispatcher as the peer B shown in Figure 2, it invokes Service Dispatcher to send a service component to peers (that fail independently) and waits for the result. On the other hand, if the peer plays the role of a service component receiver (as any peer in the peer set Cs shown in Figure 2), it executes the assigned component and returns its result to the dispatcher peer.

Peers that receive the same component execute works in parallel and return their results as soon as possible in order to save time. Because the receiver peers are failure independent to each other the dispatcher peer is supposed to get one result at least. The service provisioning function, *provision*, of the Execution Agent of receiver peers is a simple function

provision: $Ts \rightarrow Srv$,

which returns the execution result of a subtask as a service is sent to the one of dispatcher peers.

Figure 5. Fault-tolerance module (FTM) architecture



FAULT-TOLERANCE MODULE

The Fault-Tolerance Module (FTM) is the core mechanism of our approach. The main idea of the module is inspired by Introspective Failure Analysis (Weatherspoon et. al., 2002), which presumed that the probability of a component failing is independent of the duration of the coordination in progress, that all components have an identical probability of failure, and that components fail independently. However, their assumptions cannot faithfully reflect the reality (i.e., failures are often correlated) and will be relaxed in this paper.

FTM is composed of four parts (as shown in Figure 5): (1) Local Registry keeps monitoring nearby peers in order to collect such information as peer locations, user characters, device brands and types, operating system types and releases, access control settings, *etc.* (2) Failure Correlation Modeler draws upon peer user information of several failure dimensions to develop models that perform time-series prediction of the failure correlation among *peer types*. (A peer type is a tuple enumerating peer properties from three different dimensions including *human sources, software version,* and *hardware environment*. These dimensions directly affect the availability and reliability of a peer.) (3) Low-Failure Set Creator receives as input the failure correlation model from Failure Correlation Modeler and produces the dispatch sets (of mutually highly failure-uncorrelated peers) using the most recent peer list recorded by Local Registry. (4) Service Dispatcher is responsible for delegating a service request to each peer listed in the dispatch set and waits for signed acknowledgement. Service Dispatcher records experience about the process of coordination and supply this information to Failure Correlation Modeler so as to refine the failure correlation model.

The main idea behind FTM is exerting physical redundancy for fault tolerance and fulfilled by a replication mechanism that mainly comprises two tasks: (1) the determination of a number of peer candidates for duplicating a service (enabled by the parts of Local Registry, Failure Correlation Modeler and Low-Failure Set Creator) (2) the execution of the replication process (enabled by the part of Service Dispatcher). The details of FTM's four main parts will then be detailed in Section 3.1-3.4.

Local Registry

Local Registry is responsible for monitoring peers in the vicinity. The information collected

includes peer identity, user information, device brands and types, operating system types and releases, and access settings. We assume that the identities of nearby peers can be represented by a set *PI* of *peer identity* and the access settings of the peers can be represented by a set $PA = \{L, M, H\}$[1] of *peer access setting*. Through a user's access setting UbiSrvInt is able to know if a user is willingness level to host a specified function. Access settings are of three different levels: (1) low: willing to provide available services that it has on hand to the request peer automatically anytime; (2) medium: allowing only familiar or trusty peers to access the available services; (3) high: ignoring all requests the user has received and thus any forms of service provisioning are disabled.

Local Registry accordingly produces a list of nearby[2] peers, which is represented by a set *PL* of all possible peer entries. The monitor function is formulated as follows:

$$monitor-vicinity: \wp(PI) \times \wp(PT) \times \wp(PA) \rightarrow PL,$$

in which *PT* represents the peer type which is the combination of several peer attributes. The ideas of peer type will be introduced in the following sections.

Failure Correlation Modeler

Failure Correlation Modeler takes charge of building compact models in order to group highly correlated peers together. *Evolution* and *sharing* are the two distinctive properties:

- **Evolution:** It is imperative for failure correlation models to learn, adapt and grow toward maturity in a kind of continuous fashion rather than merely a one-shot experience. Fortunately, continued on-line data stream perfectly plays as the training examples to evolve the failure correlation models over time. What remains to require is an incremental on-line learning algorithm that can process the on-line data so as to develop and then evolve the failure correlation models incrementally.
- **Sharing:** P2P communication enhances and glorifies the network effect through resources sharing. To make the best use of that, the models built by Failure Correlation Modeler is gifted with the feature of sharing. The models are constantly evolving and conveying themselves between peers to achieve the objective of Semi-Global so as to enable effective service provision. More details will be described afterward.

To cope with the required evolutionary property, the primary task of Failure Correlation Modeler is modeled as a time-series problem. The highly correlated peers exhibit similar temporal patterns scattering along various time frames. Hence similar series of input data that have a similar failure distribution will be clustered together. This time-series problem is then unfolded with the descriptions of its data sequence, pattern matching and clustering analysis as shown below.

Data Sequences

A task (service request) that has been assigned to a dispatched peer is called a *mission*. The charge of the dispatched peer is producing a service. Assuming that the outcome *o* of a mission is represented by 0 or 1 where 0 and 1 represent the outcomes of success and failure respectively:

$$o = \begin{cases} 0, success \\ 1, failure \end{cases}$$

A series of mission outcomes from a dispatched peer is regarded as an *experience*, which is the object of study addressing this problem. The time series data, *experience*, consist of a sequence of *N* observed pairs,

$E_{peer} \equiv (o_i, t_i), i = 1, 2, ..., N$

where o_i denotes the measured value of the *mission outcome* at *time* t_i and E denotes the set of experience.

A peer has several experiences corresponding to the number of the dispatched peers it has interacted. A peer classifies different experiences systematically according to different dispatched peers.

Sliding Windows

We divide the time period of the inputted time series data into fixed size units. In other words, a divided segment is an aggregate of successive time. Each segment could be visualized as a sliding window, whose size is twenty-four hours. We then subdivide the segment into smaller equal parts. Each divided equal part is called a *time slot* as shown in Figure 6.

For each time slot, we calculate *failure probability* in order to analyze failure correlation between different peers in different time periods. Supposed that the *number of experiences* (the number of outcomes) of each service request in a time slot is N_o, the failure probability of the time slot is then represented by P:

$$P_{peer} = f(t) = \frac{\sum_{i=1}^{N_o} O_i}{N_o},$$

where $\sum_{i=1}^{N_o} O_i$ denotes the total number of the failure occurrences in experiences of a specific dispatched peer in the time slot. Thus, a failure curve can be derived through counting failure probabilities of each time slot within a same sliding window. For different dispatched peers, the corresponding failure curves are preserved for further analysis.

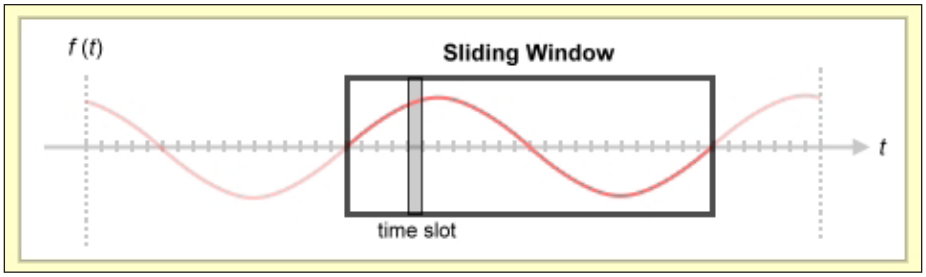
In this research we use a time scale of *day* in our analysis to predict the failure correlation. The time series data segmented by the time scale of day is called *day-patterns*, which is represented by a set *DP* of all possible day-patterns. UbiSrvInt preserves a day-pattern for each peer. The newly attained experience will be used to refine the original day-pattern into a more updated day-pattern. The simplest way to get the new day-pattern is to average the data. The function of refinement day-pattern, *refine-day-pattern*, is formulated as follows:

$refine - day - pattern: E \rightarrow DP$

Enhanced Day-Pattern Matching

Data sequence matching finds those sequences that are similar to one other, which is useful for the analysis of failure correlation among peers. Accordingly, Fault-Correlation Modeler aims to locate peers that mutual correlated along the time axis. It is assumed that the fault-correlated peers have similar shapes of day-pattern in the space constructed by a vertical axle of failure

Figure 6. Sliding windows



probability and a transverse axle of time. The similarity matching is then performed by computing the distance between sequences in the time domain.

Unlike common distance measures (e.g., Euclidian), a distance measure suitable for FTM has to be invariant of gaps, offsets and amplitudes. For instance, there are four day-patterns, A, B, C, and D shown in Figure 7. B and C as well as A and D exhibit the same tendency at the same time. Obviously, B has the higher failure correlation with C than that with A or D although B is of the closest distance to A (instead of C). Therefore, an *enhanced similarity* search method (Agrawal et. al., 1995) is necessary for matching pairs of subsequences *if they are of the same shape, but differ due to the presence of gaps (Figure 8(a)) within a sequence or differences in offsets (Figure 8(b)) or amplitudes (Figure 8(c)).*

Peer Types Grouping

Rather than analyzing failure correlations among peers in the real network environment, we profile *peer types* that enable the analysis of a more abstract level. This is more tractable because it greatly reduces the number of targets needed to be analyzed and clustered. We divide all peers into several groups according to their major properties. Three dimensions of properties for grouping peers are employed (and we believe these properties are staple sources of failure): (1) *User behavior*: Mobile devices are easily damaged from rough usage of users. On the other hand, human characteristics are believed to influence a user's behavior somehow. Therefore, it is possible that the characters of a user would correlate closely with the failure of her or his mobile device. Hence, we select *user characters* to be the first dimension of peer type, which includes constellation, blood type, age, sex and occupation, *etc. (2) Software*: Software is another factor causing failures. There are many kinds of software installed in a mobile device but the most likely reason for a system

Figure 7. Example of day-patterns

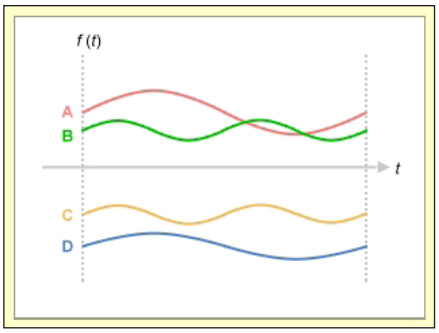
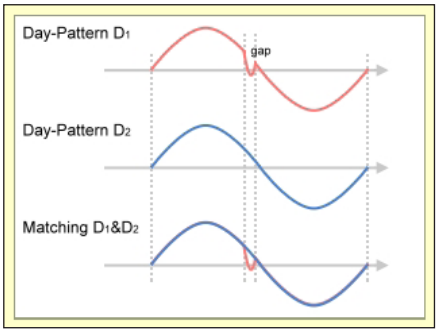
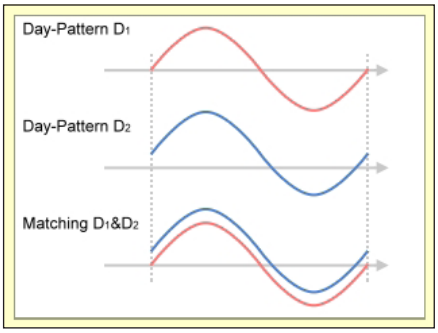


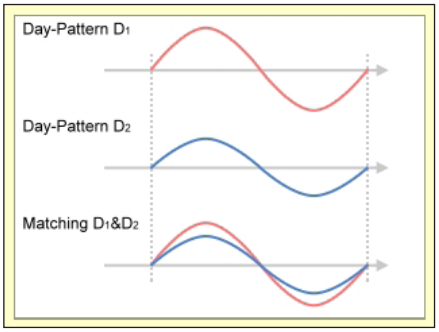
Figure 8. Enhanced similarity



(a) Gap Existence



(b) Offset Difference



(c) Amplitude Difference

to fail is always because of its operating system (OS). Therefore, we select *OS names and versions* to be the second dimension of peer type. (3) *Hardware*: No doubt a breakdown of hardware will definitely cause failures and interrupt service provisioning immediately. Equipment failure is the most basic one of our concerns. Accordingly, we select *device brands and types* to be the final dimension of peer type.

The *peer type* can be represented as an attribute vector *PT* of the form as follows:

$$\Im \equiv PT(f_{user}, f_{os}, f_{device}),$$

where f_{user}, f_{os}, *and* f_{device} denotes the values of nominal variables, user characters, OS types and releases, and device brands and types.

Clustering Analysis

The objective is to merge similar day-patterns of peer types into clusters. Supposing *M* denotes the number of peer types. For each peer type, its day-pattern is being collected continuously. Accordingly, *M* sequences of day-patterns are summarized and each day-pattern P_j ($j=1,2...,M$) consists of a sequence of pairs shown below:

$$(p_{ij}, t_i),\ i = 1, 2, ..., N, j = 1, 2, ..., M,$$

where p_{ij} denotes the *failure probability* of peer type *j* at time t_i that has elapsed since midnight.

The clustering process starts with computing the mutual distances between all day-patterns, and ends with a pre-defined number of clusters formed. The two day-patterns which are most similar will be merged to a new day pattern. Such merging process is repeated until a pre-defined number of clusters are generated. The final day-patterns then become the clusters.

The distance measure employed should be invariant to offset and amplitude as addressed in Figure 8. Suppose that we have two day-patterns y_i and z_i, i = 1, …, N. Combining the root-mean-square distance measure with *offset translation* and *amplitude scaling*, resulting in a distance normalization measure:

$$d_{nvm} = \frac{\sqrt{\sum\left(\frac{y_i - y_{\min}}{y_{\max} - y_{\min}} - \frac{z_i - z_{\min}}{z_{\max} - z_{\min}}\right)^2}}{N}$$

which can aid clustering patterns of similar shapes. The higher the value of the measure is, the higher failure correlation the two day-patterns have. The function of the complete distance normalization measure, *enhanced-distance-counting*, is then formulated as follows:

$$enhanced - distance - counting: DP_i \times DP_j \rightarrow D,$$

in which *D* denotes the distance between pair of day-patterns.

A number of methods can be used in clustering day-patterns, such as local k-means clustering, evolving clustering method (ECM) (Song et. al., 2001), evolving self organizing maps (ESOMs) (Deng et. al., 2000), or any other incremental distance-based clustering methods. The model produced by Failure Correlation Modeler is represented by a set *M* of all possible models. The *incremental-online-clustering* function is formulated as follows:

$$incremental - online - clustering: D \rightarrow M.$$

Semi-Global

When a new peer connects to the wireless service network (e.g. a new PDA user), it can rely on the experiences of other peers in the vicinity so as to promptly accommodate itself to the unfamiliar environment. Even though at the commencement of the whole service network where peers have only few experiences, peers can still gather these few experiences together to constitute a more reliable model and share.

Failure Correlation Modeler communicates with other ones so as to exchange models among peers. The collected models will be merged into a coherent model that will be then conveyed to other Failure Correlation Modelers. *The nearby peers can share the Semi-Global models in order to build a more precise Low-Failure set and prevent failure.*

Low-Failure Set Creator

Low-Failure Set Creator is responsible for creating dispatch sets grouping mutually highly failure uncorrelated peers together. The dispatch set is represented by a set *PS* of all possible peers. It uses the failure correlation models to choose several peer types that have the lowest probability of correlated failures. Subsequently, it attains a list of peers (registered in Local Registry) in which the most suitable peers for the chosen peer types can be selected. The *set-creating* function is formulated as follows:

$set - creating: m \times PL \rightarrow PS.$

Component Dispatcher

Component Dispatcher is in charge of delegating a specific service request to each peer listed in the dispatch set and waits for all acknowledgements from the service request receivers. If some of them are of no response, Component Dispatcher will send unacknowledged service request again to the other peers on the dispatch sets to ensure the dispatch is effective, realizing physical redundancy of fault tolerance.

Algorithm 2. Local registry

```
function LOCAL-REGISTRY (pt, pi, pa) returns a peer list
input:  pt, the peer type
        pi, identity of the peer
        pa, access setting of the peer
static: pl, nearby peer list

pl ← MONITOR-VICINITY(pt, pi, pa)
return pl
```

Algorithm 3. Failure correlation modeler

```
function FAILURE-CORRELATION-MODELER(e, pt) returns a model
input: pt, the peer type
       e, the experience
static: d, the distance between two day-patterns
        m, failure correlation model
        dp, day-pattern of any peer type

dp←REFINE-DAY-PATTERN(e)
for each peer type i do
      for each other peer type j do
          d←ENHANCED-DISTANCE-COUNTING(dp_i, dp_j)
      end
end
m←INCREMENTAL-ONLINE-CLUSTERING(d)
return m
function ENHANCED-DISTANCE-COUNTING (e_i, e_j) returns a distance
input: e_i, the experience of peer type i
       e_j, the experience of peer type j
static: d, the distance between two day-patterns

for   e_i and e_j do
      e←OFFSET-TRANSLATION(e)
      e←AMPLITUDE-SCALING(e)
end
d←DISTANCE-COUNTING(e_i, e_j)
return d
```

Algorithm 4. Low-failure set creator

```
function LOW-FAILURE-SET-CREATOR (m, pl) returns a dispatch set
input: m, failure correlation model
       pl, nearby peer list
static: ps, a set of mutual highly uncorrelated peer

for each peer on the peer list do
    if the access control of the peer is low then
        ps←SET-CREATING(m, pl)
    end
return ps
```

Algorithm 5. Component dispatcher

```
function COMPONENT-DISPATCHER (ps) returns an experience
input:  ps, a set of mutual low failure correlated peer
static: e, the experience
        t, time
        o, mission outcomes

  for each peer set do
    loop do
      WAKEUP-PEER()
      WAITING-ACK()
      if get ack before timeout then
        Dispatch-Component()
      if the dispatch number > the minimum limit then exit
    end
  end
  while not timeout do
    o ← GET-OUTCOME-REPORT()
    t ← GET-TIME()
  loop
  e ← REFINE-FAILURE-PROBABILITY(o, t)
return e
```

With the aforementioned descriptions of the FTM module, the overall processes of the module are then outlined by the following pseudo-codes (Algorithms 2-5).

IMPLEMENTATION

UbiSrvInt is implemented with the technology of JXTA (http://www.jxta.org) going with Personal Java that works for handheld devices such as iPAQ. JXTA is a modular platform that provides simple and essential building blocks for developing a wide range of distributed services and applications. Both centralized and de-centralized services can be developed on top of the JXTA platform. JXTA services can be implemented to interoperate with other services giving rise to new P2P applications. The overall operations of UbiSrvInt are to be depicted by a use case diagram and an activity diagram (represented in UML) as shown in the Appendix (Figure A1& A2).

Ontologies are also extensively utilized in UbiSrvInt for capturing myriad types of resources and knowledge (context information, user and device profiles, composition knowledge and reasoning rules). Resource Description Framework (RDF) / Resource Description Framework Schema (RDFS) is employed to represent the ontologies in order to preserve the semantics of the above-mentioned resources. In our implementation Jena (http://www.hpl.hp.com/semweb/jena.htm), a full-featured Java API, is exerted to create and manipulate the RDF graphs required.

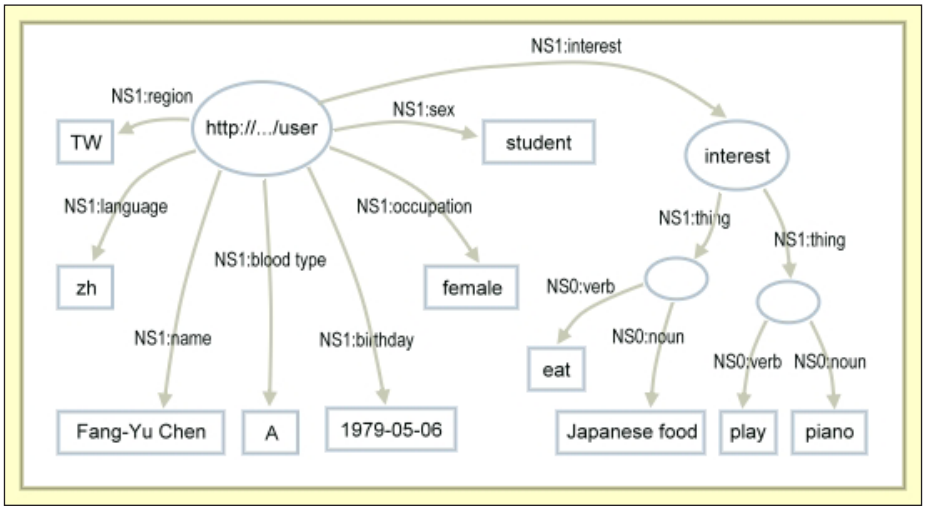
For a better understanding of the aforementioned resources and knowledge, as follows are the exemplar representations employed in the parts of Reasoning Agent, Sub-Tasking Agent and Fault-Tolerance Module.

Reasoning Agent

In order to provide users with personalized and contextualized service, a user intention reasoner is implemented conforming to the BDI conception. The knowledge involved in the BDI reasoning includes user profile, context information, and reasoning rules (that are subsequently represented with RDF graphs[3]).

The *resource*, user, is shown as an ellipse and is identified by a Uniform Resource Identifier (URI), in Figure 9 "http://...../user". Resources have several *properties* represented by arcs, labeled with the names of properties. There

Figure 9. User profile in a RDF graph



are eight properties in this case. The name of a property is also a URI, but as it is rather long and cumbersome, the following diagrams show it only in a brief form. A property is compound by two parts, a namespace prefix and a local name in the namespace. Every property has a value which is either a *literal* shown in each rectangle or a resource shown in each ellipse. RDF can take other resources as their value as the "interest" in this case for instance. Here we use a combination of two properties, NS0:verb and NS0:noun, to represent the structure of user's preferences, "interest".

Figure 10. Context information represented with a RDF graph

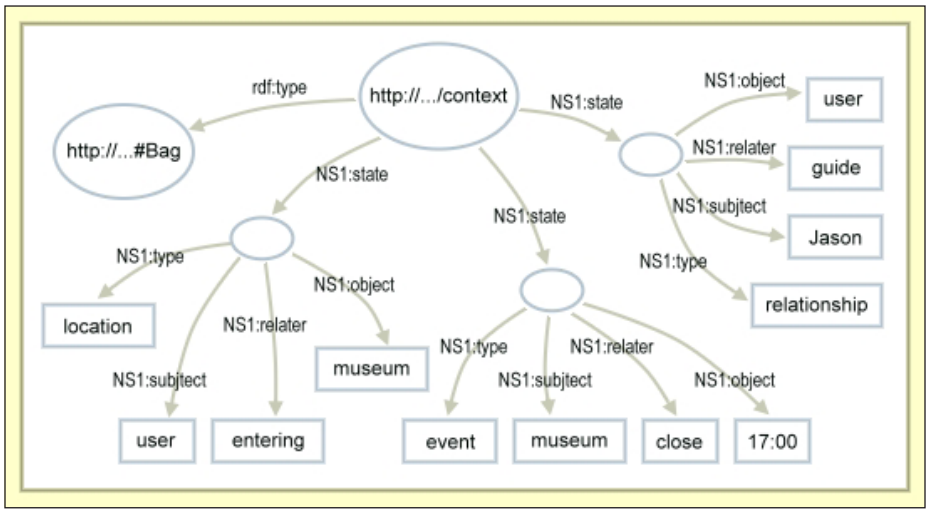
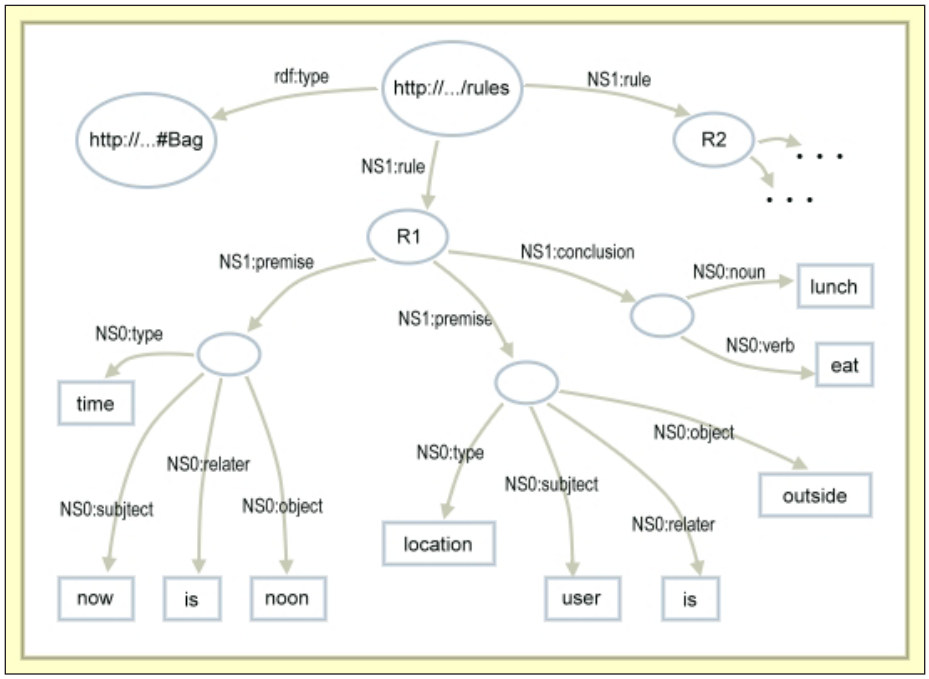


Figure 11. Reasoning rules in a RDF graph



(<*Verb*> <*Noun*>)

Note that each property NS1:thing of resource "interest" is a *blank node* since it has no specific name.

Ranganathan's Context Model (Ranganathan et. al., 2003) is used to describe the context information captured from the surrounding of user. Each clausal model of context is presented in a tuple of the form: (<*ContextType*> <*Subject*> <*Relater*> <*Object*>) similar to English sentence. A simple exemplar is shown in Figure 10 in the format of RDF graph.

As to the BDI reasoning rules, the same expressions are used to represent the premises and conclusions of a single rule, which is exemplified as shown in Figure 11.

Sub-Tasking Agent

Subtask Agent decomposes a required service by user into several pieces of subtasks and then integrates the received pieces of services to fulfill this service request. The service, i.e. task, decomposition knowledge used here is also described using RDF document.

RDF defines a special kind of resources for representing collections of things. These resources are called *containers*. In Figure 12, we use a specific kind of container, *BAG*, to represent an unordered collection of the decomposition knowledge. In this case, the resources is named "http://...../knowhow", having an rdf:type property whose value is rdf:Bag, "http://.....#Bag". In this case, the two members of the bag are represented by the property NS0:task. The ordering of properties is not significant in a bag. Hence we could switch the values between two properties and the resulting graph would represent the same information.

Fault-Tolerance Module

Fault-Tolerance Module plays a critical role in service providing. It determines the dispatched peer lists by calculating the failure correlation

Figure 12. Decomposition knowledge in a RDF graph

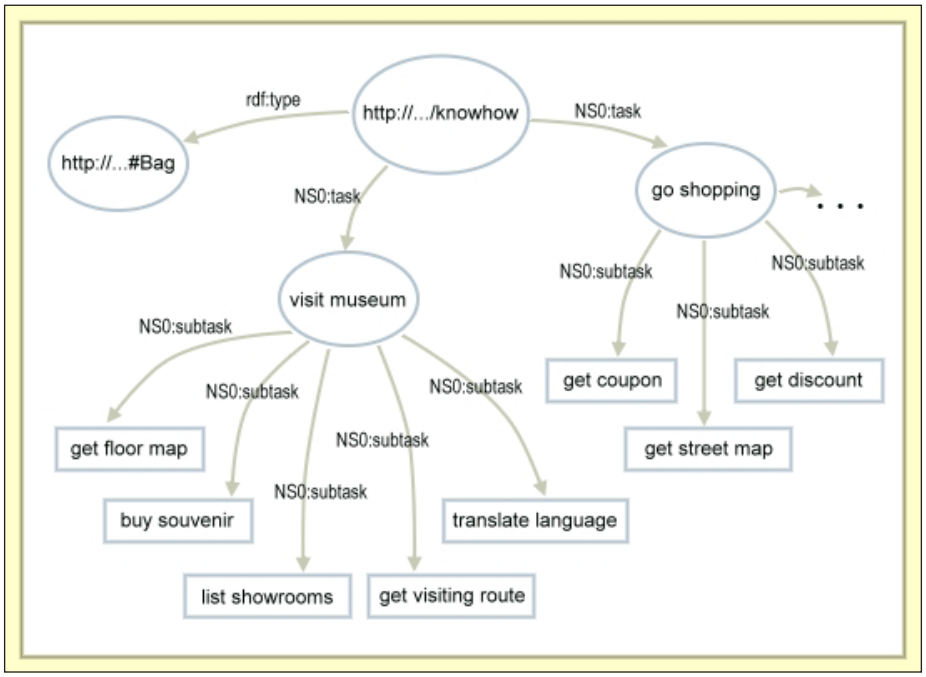
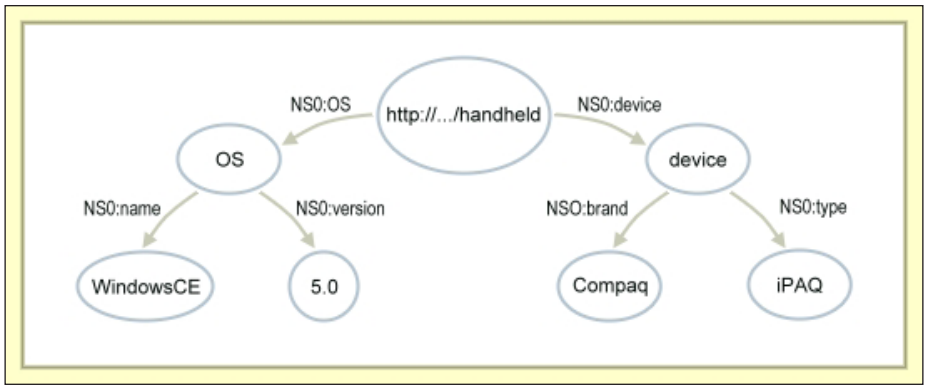


Figure 13. Handheld device profile in a RDF graph



between different peer types, thus to create low failure correlation peer sets for service dispatching. Peers are grouped into different peer types regarding three dimensions: user characters, OS name and version, and device brand and type. Again, the knowledge of user profile and handheld device profile are described and stored in RDF forms as shown in Figure 9 and Figure 13.

On the other hand, in Fault-Tolerance Module we do not exert a sophisticated incremental clustering method since what we emphasis in the experiments is the fault-tolerance mechanism. A combination of the *k*-means clustering (Hartigan et. al., 1979) and a data compression technique (Chen et. al., 2004) is employed to effectuate the incremental clustering method.

The objects of the clustering method are day-patterns, and naturally are the targets of the data compression. The rationale of the data compression is simply to merge the experiences attained from outside into a compact data set, i.e. the day-pattern model. By this way, the clustering time can be effectively shortened by restraining the size of primitive clustering data in preparation. Also, the features of data can be preserved in a good condition at the same time. The day-pattern model is learning and adapting itself over time to evolve toward maturity.

EVALUATION RESULTS

Owing to the limitation of space, in this chapter the evaluations of UbiSrvInt are two folds: fault tolerance and context awareness. In this section, the performance of UbiSrvInt on fault tolerance will be furnished. As to context awareness of UbiSrvInt, small demo scenarios (as shown in Section 5.4) are exerted to show the functionality of UbiSrvInt on context awareness (as the precision of the reasoning is completely dependant on domain-specific knowledge and thus it is negligible to evaluate the reasoning precision). Those demo scenario depicts the vision of ad-hoc wireless service provision that provides personalized and contextualized tourist services according to a user's profile and his/her surrounding context. For complete details of the demo scenarios, please see (Chen et. al., 2004).

Since UbiSrvInt is unique in its capability of fault tolerance (as far as as-hoc wireless service provision is concerned), this section aims to investigate if the idea of FTM is feasible in WP2P service sharing networks.

- We first need to *justify that the benefit of the replication mechanism (physical redundancy of fault tolerance) devised and provided by FTM is greater than the overhead*

it generates. We also need to *find proper values of the replication factor (k – the dispatch number per request) to meet acceptable levels of availability and to avoid unnecessary high cost.*

- We then *examine if the Semi-Global mechanism indeed brings into full play the synergy of experience sharing of peers and make a decisive contribution to the success of FTM.*

For the aforementioned attempted investigations, several metrics are exerted to measure the overall service performance and overhead of UbiSrvInt. Performance is measured by *solvability* and *efficiency* of services. Overhead is then measured primarily by *load* along three dimensions: *network traffic, processing cost,* and *clustering cost*. The detailed explanation and calculation of these metrics are shown below:

- **Performance:** *Performance is generally regarded as the overall throughput of completed services that peers provide through time.* A service is defined as a task, which can be decomposed into several subtasks, i.e. lightweight services. A service is considered successful only if it is through a complete successful process of discovery, execution and integration of all the subtasks. We look at throughput from two perspectives:
  - *Service Solvability* is measured in the services success rate, which represents the effectiveness of the system. The success rate is defined as the percentage of successful services through time.

$$\textit{Service Solvability} = \frac{\textit{Cumulative number of successful services}}{\textit{Cumulative number of requested services}}$$

  - *Service Efficiency* is measured in the reciprocal of the service response time per successful service. For a successful service, we define the service response time as the time between when a formal request message is sent and when the first reply message of the request is received.

$$\textit{Service Efficiency} = \frac{\textit{Cumulative number of successful services}}{\sum(\textit{The time service received} - \textit{The time service requested})}$$

- **Load:** *Load is conceptually regarded as the amount of efforts that peers must engage for attaining successful results.* One factor concerned in load measurement in networks is the flow of messages exchanged. Besides, the processing cost of service requests and the processing cost of clustering analysis are also considered. In examining the amount of efforts that peers must engage for attaining successful results, three perspectives are then unfolded to formulate the load measurements:
  - *Network traffic* is measured by counting the number of message sent by peers to the network. An overhead of network traffic incurred by peers with respect to the total resulting successful services is then defined as follows:

$$\textit{Relative Network Traffic} = \frac{\textit{Cumulative number of forwarded messages}}{\textit{Cumulative number of successful services}}$$

  - *Processing cost* is conceptually measured in the number[4] of subtasks executed by peers. An overhead of processing cost incurred by peers with

respect to the total resulting successful services is defined as follows:

$$\text{Relative Processing Cost} = \frac{\text{Cumulative number of executed subtasks}}{\text{Cumulative number of successful services}}$$

- *Clustering cost* is measured by calculating the total time spent on the core process of FTM, clustering analysis. An overhead of clustering cost incurred with respect to the total resulting successful services is defined as follows:

$$\text{Relative Clustering Cost} = \frac{\text{Cumulative time spent on clustering}}{\text{Cumulative number of successful services}}$$

The remainder of this section is unfolded as follows: (1) We first describe the parameters of the experiment setting, followed by a description of the experiment process (Section 4.1) (2) The feasibility analysis of FTM in UbiSrvInt is provided in Section 4.2. (3) The examination of the effect of the Semi-Global mechanism is then furnished in Section 4.3. Discussions of the performance of UbiSrvInt on fault tolerance will be then interspersed in the respective subsections.

Experiment Settings

The performance of FTM is affected by a wide range of parameters. We divide these parameters into global parameters and local parameters. A quick look at these parameters is shown Table 1.

Global parameters describe the background of environment used for running several miniature peers from a macro point of view. They are application dependent environment constants, which include the number of peers in the network, the total number of unique services that can be provided in the network, and the average

Table 1. Experimental parameters

Parameter	Value	Description
$T_{simulation}$	2 hr.	Total simulation time
$S_{network}$	20	Total network size
$\lambda_{mobility}$	0.02	Leave (Join) rate per peer per minutes
$S_{service}$	10	Total size of service space
$N_{subtask}$	4	Average number of subtask of a service
$S_{capacity}$	20	Service capacity per peer
$\lambda_{request}$	0.2	Request rate per peer per minute
$R_{choosing}$	0.1	Request choosing rate. ($R_{choosing} = 1/S_{service}$)
k		Dispatch number per request
λ	$[0, \lambda_{max}]$	Failure rate per peer per minute
λ_{repair}	2	Repair rate per peer per minute
T_{wait}	40 sec.	Respond timeout
$T_{abandon}$	4 min.	Service timeout
S_{window}	24 hr.	Size of sliding window
S_{slot}	1 hr.	Size of time slots in sliding windows

number of subtasks that must be done for a single service, *etc.*

On the other hand, local parameters are parameters specific to the functionalities of UbiSrvInt; the values of the parameters may differ from peer to peer. These parameters describe the behaviors of peer, which include the storage capacity of services for each peer, the average time to request for a service per peer, and the average time of failure and recovery of a peer, *etc.*

Due to space limitation, we only explain certain parameter value settings and their correlative assumptions in our experiments (for complete descriptions of the parameters, please see (Chen et. al., 2004)):

- The size of service space $S_{service}$ is set to 10. What we called a service space is a pre-existing collection of services allowing peer access. The services in the service space are all composed by several subtasks. We assume that the subtask number $N_{subtask}$ of a particular service follows a normal distribution with a mean of 4 and variance of 1 (without loss of generalization). To reduce the effect of uncertain variables, the level of hierarchical service decomposition is set to 1. Thus a service only needs to be decomposed once by the request peer. We assume that peer can get the decomposition knowledge from outside world entities, such as provisioning servers or other peers.
- The service capacity $S_{capacity}$ denotes the number of subtasks a peer can provide. We assume that service capacity per peer follows a normal Distribution with a mean of 20 and variance of 4 (without loss of generalization) for simulating the actual environment where the variety of mobile devices influences its capability of providing service.
- As to the behavior of peers, we assume that each peer process service requests with an exponentially distributed rate $\lambda_{request}$ of 0.2 per minutes (without loss of generalization). That is to say, the average elapsed time to generate a demand of a service per peer is 5 minutes. Each service request R_i, $0 \le i < S_{service}$, is chosen uniformly at random inside the service space. Thus, with the size of service space, $S_{service}$, of 10, the average choosing rate $R_{choosing}$ of a service request is 0.1.
- We simulate the failure occurrence for each peer with λ. To complement, we also set a repair rate λ_{repair} for a peer. Both rates are simulated according to exponential distributions. We assume that *each peer has a different failure rate* λ_i, $0 \le i < S_{network}$. Accordingly, we set a range corresponding to different peer types, where that *top failure rates* $\lambda_{\max}$ *may be different in different experiments* (*denoting stability of ubiquitous service environments*). The parameter λ_i is attained from the weighted effects of user characters, OS versions and releases, and device brands and types respectively as follow:

$$\lambda_i = \lambda_{max} (w_{user} E_{user} + w_{os} E_{os} + w_{device} E_{device})$$

$$w_{user} + w_{os} + w_{device} = 1, 0 \le E_{user} \le 1, 0 \le E_{os} \le 1, 0 \le E_{device} \le 1$$

  in which E_{user} denotes the user effect, E_{os} denotes the OS effect, and E_{device} denotes the device effect. In the simulation, w_{user}, w_{os}, and w_{device} are set to 0.2, 0.3, 0.5 separately. As to the repair rate, the mean time is fixed to 0.5 minutes because we suppose that most mobile device have a speedy recovery capability.
- We assume that occurrences of events in the simulation are all *Poisson processes*, including the arrival requests, unexpected failures and the corresponding repairs from failures. A Poisson process is characterized by its rate function $\lambda(t)$, which is the expected number of "events" or "arrivals" that occur per unit

time. Poisson processes are of the characteristics: (1) *Orderliness*: Events don't occur simultaneously. (2) *Memorylessness*: Any event occurring after time *t* is independent of any event occurring before time *t*. The exponential distribution is generally used to model Poisson processes, where events occur with constant probabilities per unit. Particularly, failures in an electronic device are usually memoryless and hence are well modeled by an exponential distribution.

In the simulation, the experiments will be conducted several times with different setting of configurations, which are defined by a set of global and local parameters as shown in Table 1. Most experiments are conducted given a simulation time $T_{simulation}$ of 2 hours. *In the experiments, we use the aforementioned metrics to evaluate the behavior of UbiSrvInt for a changing dispatch number (k), dynamic vs. stable environments, and with vs. without the Semi-Global mechanism respectively.*

The results of the experiments are primarily illustrated with cumulative time sequence diagrams. That is, for each metric we accumulate observed values over time, and mark it on diagram every 10 minutes to draw a curve so as to see how trend changes. In other words, each data point on the curve denotes an overall value of the time point labeled on x-axis.

Feasibility Analysis of FTM

Since FTM is exerting physical redundancy for fault tolerance, FTM first determines a number of peer candidates (the dispatch number k) for duplicating a service and then execute the replication process (dispatching the service to the peer candidates). This section aims to *examine how the system behaves as the dispatch number k increases and then justifies the feasibility of FTM (i.e., to show the benefit is greater than the overhead).*

Since a higher *k* value denotes the higher level of replication in the mechanism of FTM, the mechanism exists only when *k* is greater than one. In the first set of experiments, we inject failures into each peer with a failure rate per peer per minute of λ_i which ranges from 0 to 0.1, i.e.

Figure 14. Performance results in dynamic ubiquitous service environments

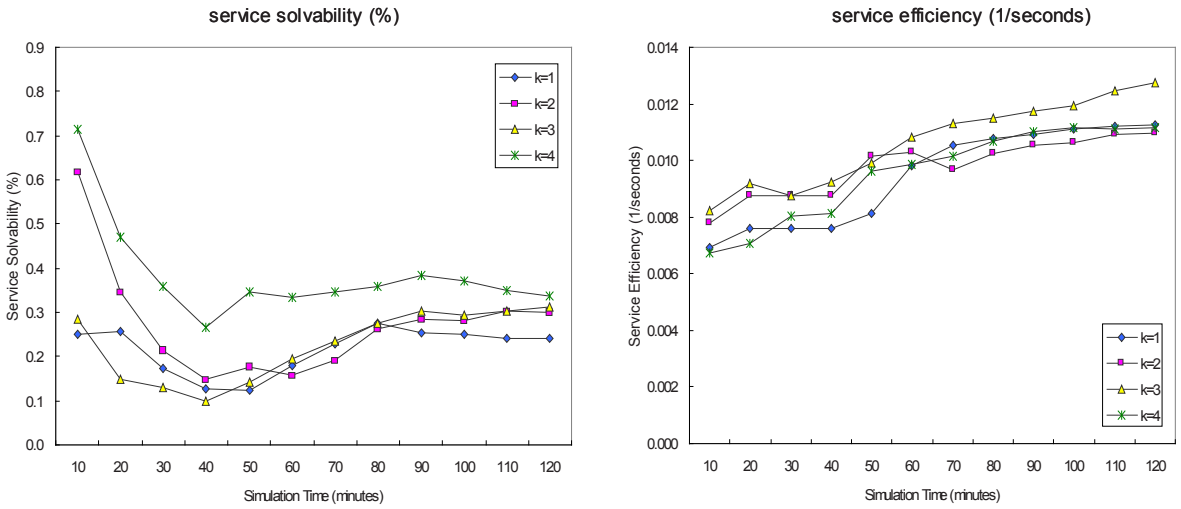


Figure 15. Load in dynamic ubiquitous service environments

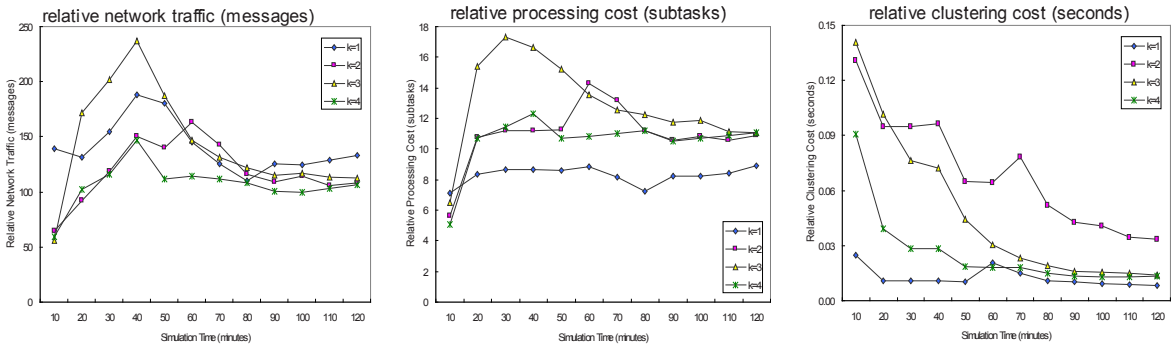


Figure 16. Performance results in stable ubiquitous service environments

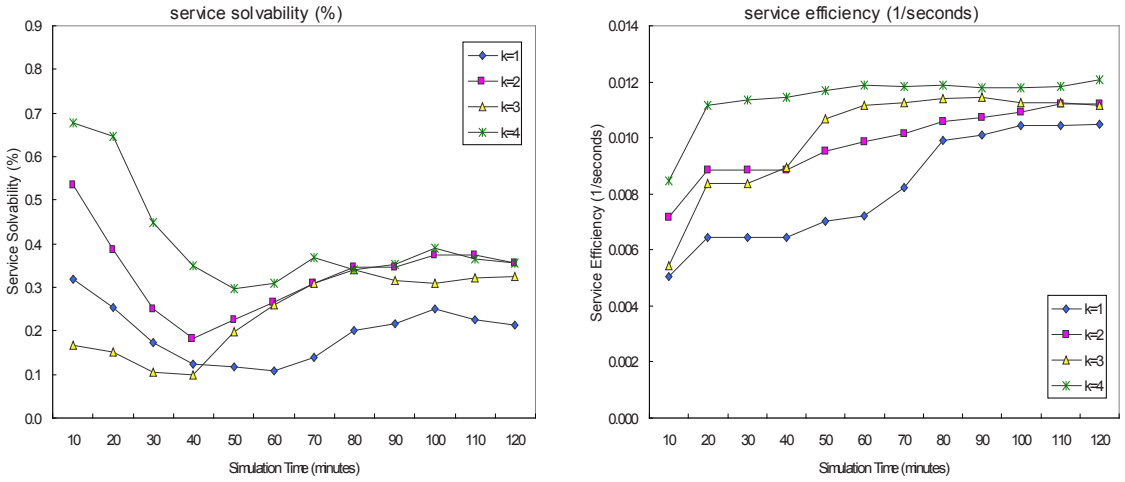
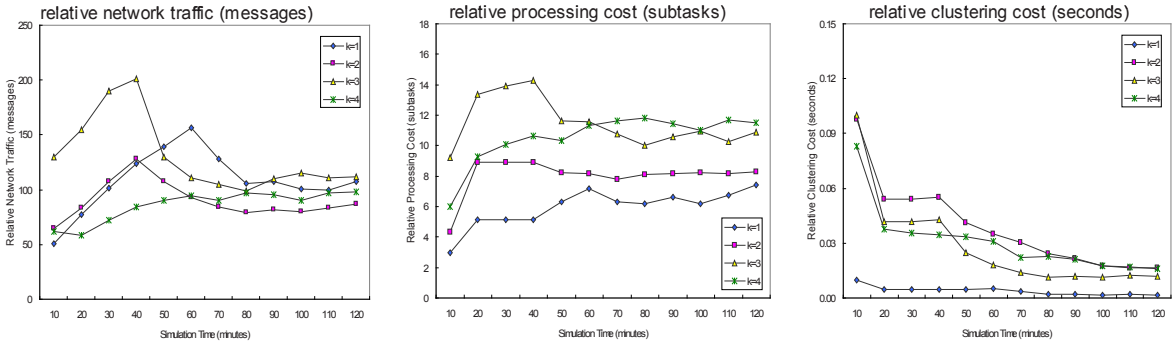


Figure 17. Load in stable ubiquitous service environments



λ_{max} *is set to 0.1 (characterizing a fairly dynamic ubiquitous service environment).* Note that the actual λ_i of each peer depends on the peer type it belongs to. Each experiment is carried out with a different configuration of *k* value. In the second set of experiments, we then conduct a series of experiments resembling to the first set but *under relatively stable environments, where* λ_{max} *is 0.05.* Accordingly the failure rate λ_i per peer per minute lies in the range [0, 0.05]. The first and second sets of experiments are both conducted with the existence of the Semi-Global mechanism.

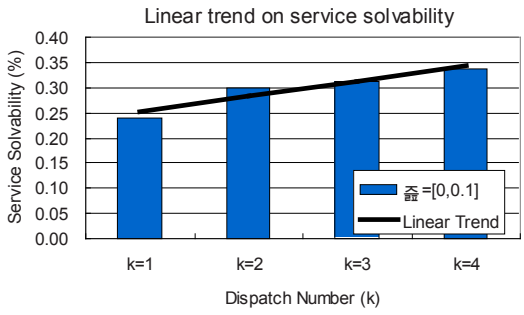
The evaluation results can briefly be itemized as follows:

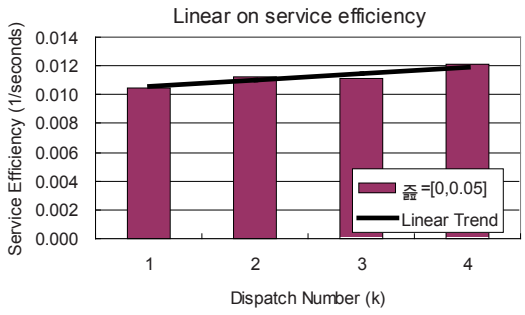
- **Performance in *dynamic* ubiquitous service environments** (Figure 14):
  - Service solvability:
    —k=4 >> k=3 >> k=2 >> k=1 (The performance in service solvability grows in proportion to the magnitude of *k*.)
    —k=4 >> k=1 (The final overall service solvability in k value of 4 is more than 1.4[5] times higher than the one in k value of 1.)
    —k=4 >> k=1 (Under a total average of the final overall solvability of 0.3, the difference in proportion between both sides - k=4 and k=1 - is more than 32%, which is highly significant)
    Note: ">>" represents a relationship of outperforming (with respect to a designated metric or measurement) between two cases of experiments using two different replication factors (k).
  - Service efficiency:
    —k=3 >> (K=4 ≒ k=2 ≒ k=1) (In general, all curves are on the rise as the time passed by. At the ends of the experiments, the final overall service efficiency for all *k* values are almost equivalent except for the value of 3. It is apparent especially in *k* value of 4 and *k* value of 1. As to the *k* value of 3, it gains the distinctly highest final overall service efficiency of all).
    Note: "≒" represents an indistinguishable relationship (with respect to a designated metric or measurement) between two cases of experiments using two different replication factors (k).
- **Load in *dynamic* ubiquitous service environments** (Figure 15):
  - Relative network traffic:
    —k=1 >> k=2 ≒ k=3 ≒ k=4 (In general, the relative network traffic curves are ordered in an inverse way against the service solvability curves. As the *k* value increases, the relative load came from network traffic goes down.)
    —k=1 >> Average of k=2,3,4 (The overheads of the final overall relative network traffic in the experiments with replication mechanism are close eventually in the experiments, which are lower than the overhead in *k* = 1 over 18%[6] on average at the ends of the experiments.)
  - Relative processing cost:
    —(k=4 ≒ k=3 ≒ k=2) >> K=1 (The overall relative processing cost has the lowest load when *k* value is set to 1. Besides, the final loads are about the same when the other *k* values are set.)
    —Average of k=2,3,4 >> k=1 for more than around 2 units of subtasks (The overhead of the overall relative processing cost incurred - for *k* of values other than 1 - is only around 2 units of subtasks per successful service request at the ends of the experiments. This is considerably low in our simulation environment.)
  - Relative clustering cost:
    —The difference between different values of *k* is only few milliseconds,

which is quite slight. Accordingly, we can neglect the impact of the clustering overhead on performance.

- **Performance in *stable* ubiquitous service environments** (Figure 16):
  - Service solvability:
    —(K=4 ≒ k=2) >> k=3 >> k=1 (In general, the overall performance of FTM still far outperforms the one without the replication mechanism, i.e. k = 1. But the level of replication in FTM has relatively limited impact on the performance in comparison with that of a dynamic environment. The experiment using a k value of 4 is still performs well above the others, but the difference between that using a k value of 2 is a nuance in the later half of the experimental durations. This might suggest a lower clustering number is good enough to maximum effect on the service efficiency and we will elaborate the discussion at the end of this section.)
    —k=4 >> k=1 (The final overall service solvability in k value of 4 is around 1.7 times higher than the one in k value of 1 at the ends of the experiments. Furthermore, the difference in proportion between both sides - k=4 and k=1 - has been raised up to around 46%.)
  - Service efficiency:
    —K=4 >> (k=3 ≒ k=2) >> k=1 (For growing the value of the dispatch number k, the performance of FTM is improved in terms of the absolute values of service efficiency throughout the simulation. The rationale is that less failure rates yield more dispatched peers contributing their provisions of subtasks and thus a request peer can acquire what it needs earlier from any one who is dispatched. The experiment using a k value of 4 then persistently has the best efficiency.)
    —k=4 >> k=1 (The overall service response time per successful service request in k value of 4 is at least 12.5 seconds faster than that in k value of 1 in the durations which represent at about 100 minutes faster in reality.)
- **Load in *stable* ubiquitous service environments** (Figure 17):
  - Relative network traffic:
    —K=3 >> k=1 >> k=4 >> k=2 (The overheads from network traffic under replication mechanism are relatively smaller than the one without replication mechanism in general, in spite of the different levels of replication.)
  - Relative Processing Cost:
    —K=4 >> k=3 >> k=2 >> k=1 (For higher values of the dispatch number

Figure 18. Summarized results of FTM's feasibility





k, FTM gains higher processing costs relatively.)
—Increase is around 1.36 units per upgrading of *k*. (On average, the increase in the overall relative processing cost when upgrading a level of replication is around 1.36 units at the ends of the experiments. This is believed to be a reasonable tradeoff with processing cost for a higher efficiency.)

- Relative clustering cost:
—Differences are only few milliseconds. (The average cost is slightly higher than that of *k* value being 1, although not a significant difference with only few milliseconds apart.)
—Average of k=4,3,2 is around 0.0147 seconds. (The costs for K being 4,3,2 start from a higher value and drop significantly to an average value of 0.0147 seconds per successful service at the ends of the experiments.)

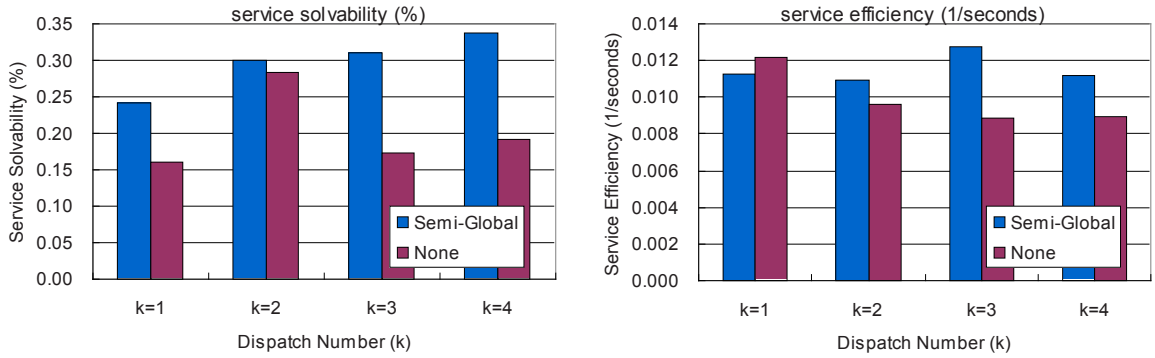
From Figure 14-17, we can conclude the following observations (that subsequently justify the feasibility of FTM):

- **Under dynamic environments:** The feasibility of FTM is very positive because of the much better performance and the relatively lower overheads. Moreover, it shows that the efficiency of $k = 3$ is apparently higher then the others as its solvability is not far from $k = 4$, and its relative overheads are close to the others. The clustering number, i.e. r*eplication level, can best effect service solvability*, as the linear trend on the final overall service solvability as shown in Figure 18. This results from the fact that *higher failure rates yield less peers accomplishing task correctly. Hence the increase of number of dispatched peers results in the higher service solvability*. In the meanwhile, the relative processing cost does not distinctly increase as *k* value increases.
- **Under stable environments:** The feasibility of FTM is affirmed once again since the enormous performance gained is sufficient to counteract the overhead involved in processing. The result in $k = 2$ is notable. We found that the solvability of $k = 2$ is almost as good as $k = 4$. And it has the lowest relative overhead of network traffic and a second lower relative processing cost (only slightly higher than $k = 1$). Moreover, *dispatch number k best effects service efficiency*, as the linear trend on the final overall service efficiency shown in Figure 18. The processing cost increases as *k* value increases. This results from the fact that *lower failure rates yield more peers performing task correctly.* Hence, we can manipulate the clustering number to maximum effect on the service efficiency by far.
- To extend the meaning of the results attained from stable environments, the quality of the replication is more effective than its quantity when the clustering data is relatively rich in contents. In stable environments, the behaviors of peers are more unobvious in terms of failure occurrence. Hence, a coarser failure correlation clustering is sufficient to classify peers into proper clusters to facilitate the replication mechanism. By contrast to dynamic environments, the quality of the replication will be more effective than its quantity if and only if a refined clustering performs, that is, plenty clustering data in content involved.

The Effects of the Semi-Global Mechanism

In this set of experiments, we follow the same configuration settings used in the first set of experiments (Section 4.2), except that the experiments are conducted without the Semi-Global

Figure 19. The positive effects of the semi-global mechanism



mechanism in order to examine its influence on FTM. For this reason, all the other parameters are set exactly the same: the top failure rate λ_{max} is set to 0.1 and the dispatch number *k* varies from 1 to 4.

In these experiments, the cumulative time sequence diagrams of the evaluation metrics are not deployed here one by one since we only interested in the final overall effect of the Semi-Global mechanism in this phase. Hence we only capture the final results obtained at the ends of the experiments, i.e. after 2 hours, to see the final overall performance and load in each experiment. And the experimental results are compared and illustrated with bar charts instead.

The evaluation results (as shown in Figure 19) can briefly be itemized as follows:

- Service solvability:
  —Average of K=4,3,2,1 with Semi-Global is 1.5 times more than that of no Semi-Global. (The mean of the final overall performances of service solvability in experiments with Semi-Global is around 1.5 times higher then the mean in experiments without Semi-Global at the ends of the experiments.)
  —k=3 with Semi-Global is 1.75 times better than that of no Semi-Global and it is 43% difference in proportion)
  —k=4 with Semi-Global is 1.85 times better than that of no Semi-Global and it is 46% difference in proportion.
- Service efficiency:
  —k=3 with Semi-Global is 35 seconds faster than that of no Semi-Global. (The experiments with Semi-Global show the higher performance comprehensively in terms of service efficiency. Especially in the case of k = 3, the difference is about 35 seconds per successful service.)

Following the above experiment results, we can conclude the beneficial results of the Semi-Global mechanism. *The replication factor is ineffectual without the complement of the Semi-Global mechanism.*

The Semi-Global mechanism improves the accuracy of clustering so as to find the most appropriate peer candidates to dispatch subtasks and subsequently attain a better service performance. That is, by way of dispatching peers who have the lowest failure correlation from each other, service-sharing system can achieve higher solvability and efficiency in general. In other words, *this experience sharing of the Semi-Global mechanism (in combination with physical redundancy of the replication mechanism) empowers UbiSrvInt with a unique capability of fault tolerance for wireless service composition.*

Possible Application Scenarios of UbiSrvInt on Context Awareness

As a result of the rising and the flourishing of tourism, enormous capital has been invested in the tourist industry. It is highly suitable to have tourism applications implemented on a P2P platform because of the characteristic of mobile tourists longing for various information and services on the fly. It is highly valuable to tourists for tourist services being personalized, contextualized and capable of providing immediate access in an ad-hoc wireless manner for tourists. From system perspective, they must be lightweight and reliable. Let us envisage a possible scenario set in the future.

For instance, Amy is visiting a museum in her sightseeing tour. Her personal digital assistant (PDA) on hand with wireless support automatically downloads "visiting route" service provided by museum while she is buying an entrance ticket. *On her way to museum lobby, her PDA broadcasts the requests for "tourist distribution", "showroom information", "floor map" and "Chinese translation" services to nearby tourists. In a short while, her PDA has already collected those services it needs to compose a personalized service for Amy.*

Amy gets a "Chinese version of visiting route" service without manual operation. Amy follows the route showed on her PDA, noticing that there are some useful facilities marked on the route such as information desk and female washroom. *Her PDA requires and collects "art introduction" automatically while she is entering a different showroom. That is, Amy can always get the right information at the right time whenever needed.* Of course, if she likes, she can choose vocal introductions rather than written ones.

The preceding scenario illustrates an ad-hoc wireless service provision providing personalized and contextualized tourist services according to Amy's profile and surrounding context. Moreover, suppose that Amy goes into the second showroom, one of the peers that provide "painting location", which is part of the "art introduction" service, is suddenly lost connection. The traditional fault-tolerance solution to this problem is to search for a candidate to re-execute the failed subtask and then compose them again. By that time Amy may leaved the second showroom already. In a WP2P environment and such a time critical situation, the time redundancy solution is unreasonable. *Exerting UbiSrvInt, the failure will be avoided by analyzing the dependability of peers as far as possible, therefore providing customized service without wasting any time.*

Alternative application scenario utilizing the social contexts (retained in the profiles of the users and processed by the BDI reasoning agent) extends the application scope of UbiSrvInt a step deeper. As follows shows a simple demonstration of the application:

1. Amy is a visitor (whose handheld profile is shown in Figure 20) of a museum in her sightseeing tour.
2. During Amy's visiting of the museum, the context information provided by museums provisioning server will be faithfully recorded and displayed on this Context Information Panel (as shown in Figure 21) to notify Amy. The Context Information Panel shows the current states of user's surrounding.
3. The BDI Reasoning Agent then obtains Amy's profile (as shown in the User Profile Setting Panel of Figure 20) and the context information (as shown in the Context Information Panel of Figure 21) to infer Amy's true intentions. The knowledge about reasoning rules can be downloaded from provisioning servers or nearby peers. And the results of reasoning will be presented in the Service Recommendation Panel for Amy's reference. (The Service Recommendation Panel can be set to automatic or manual. User can either authorize system to automatically reasoning while new context information is newly

Figure 20. Handheld profile setting panel (Automatic / Manual)

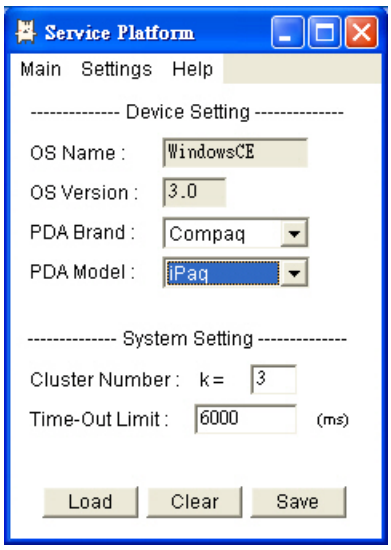
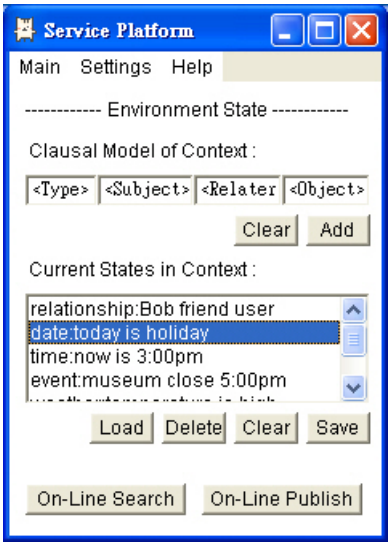
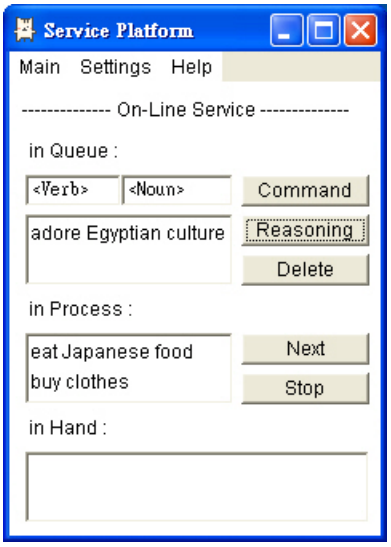


Figure 21. Context information panel (Automatic / Manual)



obtained, or choose a manual control to silence system when user needs undisturbed environments, such as during a meeting, taking a nap, *etc.*)

Figure 22. Service recommendation panel (automatic / manual)



4. Amy's friend, Bob, makes an appointment with Amy in the museum, but he is late. Before Bob's arrival, Amy takes the opportunity to browse the gift shop and choose a birthday present for Candy, her classmate, because this morning Amy got a memorandum about Candy's birthday party from the PDA. Amy also sends a digital post-card of Egyptian culture downloaded from the museum to Eva, her younger sister, who is a college student with a major in Archeology with the recommendation of the BDI reasoning agent (shown in Figure 22). The social context information is also stored in Amy's PDA in the format of tuples such as:

   (*<relationship> <Candy> <friend> <Amy>*)
   (*<profile> < Candy > <born> <1977/5/20>*)
   (*<relationship> <Eva> <sister> <Amy>*)
   (*<profile> <Eva> <love> <Egyptian culture>*)

5. After Bob arrives the museum, his PDA gets an official welcome message from the

provisioning server resident in the museum. The message may look like such a tuple: (<*location*> <*user*> <*entering*> <*Palace Museum*>). The Reasoning Agent in Bob's system gets the newly arrival context information, thus starts the inference. The incoming context information is shown on the Context Information Panel. The agent notices that Bob is a Spaniard and has a date with Amy, thus sensibly guesses that Bob may need (i) a Spanish translation of the museum tour guide and museum catalogue and (ii) a note message to inform Amy that he has arrived. The two services will be then automatically shown on the Service Recommend Panel. If Bob dislikes that, he can manually stop it, otherwise the services will be provided after they are fulfilled. The fulfilled services will be also recorded on the Service Recommendation Panel for user reference.

RELATED WORKS

This section gives some exemplars of existing relevant works on the infrastructures of e-service provision emphasizing on either service integration or service adaptation or service replication. The works about service integration, in general, rely on complied knowledge (e.g., process schema, process template, or state charts) in terms of (semi-)centralized architectures. On the other hand, the works for service adaptation mostly are condition-based and those for service replication then quest for load balancing. UbiSrvInt is distinct from these works in service provision based on inference-based adaptation and fault-correlation-analysis service replication (in addition to UbiSrvInt being a pure peer-to-peer approach).

eFlow (Casati et. al., 2002) is a platform developed at HP Laboratories for specifying, enacting, and monitoring composite e-services in order to aid in e-commerce. In eFlow, a composite service is described as a process schema that composes other basic or composite services. A composite service is modeled by a flow structure, which defines the order of execution among the nodes in the process. The graph may include service, decision, and event nodes. Service nodes represent the invocation of a basic or composite service; decision nodes specify the alternatives and rules controlling the execution flow, while event nodes enable service processes to send and receive several types of events. A service process instance is an enactment of a process schema enacted by the eFlow engine.

The Ninja (Mao et. al., 2001) project aims to develop a software infrastructure to support the next generation of Internet-based applications. In the Ninja project, a dynamic service composition platform has been designed and implemented with the goals of automation, scalability and fault-tolerance through use of cluster computing platforms, identification of common patterns of ad-hoc, application-specific compositions, classification of services into a strongly typed system, redundant control mechanisms for monitoring and recovery, and a continuous optimization process with feedback. Central to the architecture is the concept of path. A path is a flow of Application Data Units through multiple services and transformational operators across the wide area. A mechanism, Automatic Path Creation (APC) service, plays an essential role by seamlessly supporting any new communication device in the infrastructure.

A SELF-SERV platform (Benatallah et. al., 2002) for rapid composition of Web services has been developed based on Java and XML, in which web services are declaratively composed, and the resulting composite services are executed in a P2P and dynamic environment. SELF-SERV employs a declarative language for composing services based on state charts which support the expression of control-flow dependencies such as branching, merging, concurrency, *etc*. A P2P

service execution model, whereby the responsibility of coordinating the execution of a composite service, is distributed across several peer software components called *coordinators*. They are in charge of initiating, controlling, monitoring the associated services, and collaborating with their peers to manage service execution. The knowledge required while composing services is statically extracted from the state chart and represented in a simple tabular form.

Anamika (Dipanjan et. al., 2002) is a reactive service composition architecture for pervasive computing environments that is implemented over Bluetooth. Central to Anamika is the concept of a distributed broker that can execute at any node in the environment. A broker may be selected based on various parameters such as resource capability, geometric topology of the nodes and proximity of the node to the services that are required to compose a particular request. The architecture primarily deals with the discovery, integration and execution of the components of a composite request. The architecture introduces two distributed reactive techniques to carry out service composition in purely ad-hoc environments: Dynamic Broker Selection Technique, Distributed Brokering Technique. The former approach centers on a procedure of dynamically selecting a device to be a broker for a single request in the environment. And the latter approach distributes the brokering of a particular request to different entities in the system by determining their suitability to execute a part of the composite request.

PCAP (Sheng et. al., 2004) devises the design of a distributed, adaptive, and context-aware framework for personalized service composition in terms of users annotating existing process templates (leading to personalized service-based processes). Personalization is the like of execution constraints encompassing temporal and spatial constraints, which respectively indicate when and where the user wants to see a task executed. The execution policies include the service selection policy and the service migration policy. For a specific task, users can specify how to select a service for this task. The service can be a fixed one (the task always uses this service), or can be selected from a specific service community or a public directory (e.g., UDDI) based on certain criteria (e.g., location of the mobile user).

ServiceGlobe (Keidl et. al., 2003) presents a generic dispatcher in web service provision for the purpose of load balancing and high service availability in terms of automatic service replication. The dispatcher performs load balancing (or load sharing) using several servers on the back-end with identically mirrored content and a dispatching strategy like round robin using load information about the back-end servers.

CONCLUSION REMARKS

In this chapter, a novel pure P2P approach solution (UbiSrvInt) for ad-hoc wireless service provision is presented. UbiSrvInt is unique in its context awareness and fault tolerance (that are fulfilled by the approach's components - Reasoning Agent and Fault-Tolerance Module - respectively). The aim of this chapter is to empower users with mobile wireless devices to access personalized and contextualized services composed within the reachable ad-hoc network of services in a pure Peer-to-Peer manner. Accordingly, we provide a general-purpose approach (UbiSrvInt) so as to facilitate the discovery, integration, and provision of a large cross-section of P2P mobile services. UbiSrvInt advances existing service provision infrastructures (centralized or mediator-based) by its capability in the removal of the bottlenecks of the centralized/mediator nodes (for reliability, scalability, extensibility, real-time information). Moreover, UbiSrvInt takes into account fault tolerance and context awareness (that are vital for attaining high usability of the approach for wireless service provision). As for the complex-

ity of UbiSrvInt, it is still manageable because most of the computation exerted is fairly effective and with the Moore's Law the power of mobile devices would be aggressively improved (in terms of the capabilities of computation and capacity) as time goes.

The performance of UbiSrvInt on fault tolerance is three-fold: (1) Under unreliable environments, sufficient dispatch number (replication level) should be provided to seek for distinctly higher service solvability. (2) As to reliable environments, lower replication level is good enough to gain favorable solvability and replication level could be raised to seek for distinctly higher service efficiency (if the extra process cost is either unconcerned by or unaware to users). (3) For synergy of Semi-Global, in both kinds of environments Semi-Global should be enabled for complementing FTM so as to gain an overall higher performance in all aspects. As to the functionality of UbiSrvInt on context awareness, it has also been justified through relevant scenario demos of UbiSrvInt. In short, through the experiments and evaluations[7] UbiSrvInt is justified for its claimed distinctive features of context awareness and fault tolerance.

Our future work includes the application of the UbiSrvInt approach to myriad service domains (e.g., travel, learning, *etc.*) in order to attain domain dependent statistics (such as level of increased satisfaction, efficiency and activity volume). For FTM, certain further investigation can be conducted, such as granularity of time slots in a sliding window (the more exquisite the day-patterns the more segmented time slots), varied experiment parameters (e.g., network size), and advanced adaptability through personalized reasoning rules about a desired balance of accuracy and efficiency. We hope our work can shed light on further advanced platform development for contextualized P2P mobile service provision.

IMPLICATIONS FOR U-COMMERCE

For u-commerce, the nature of services focuses on actively sensing different customer's role through different specific contextual attributes (e.g., time, location, resources, customer profiles) (Fano et. al., 2002) in order to meet customers' needs and change the interactions with the customers (Varshney et. al., 2000) in terms of dynamic configurations of services and devices. To support u-commerce and the services, the environment features the ubiquitous networks that are full convergence, technologically heterogeneous, geographically dispersed, context sensing, architecturally flexible and without a centralized control mechanism (Banavar et. al., 2002). However, the ubiquitous networks, in reality, are not as reliable as the wired networks and thus the issue of fault tolerance has to be considered in addition to context awareness. To meet this end, this chapter provides an approach for P2P mobile service provision, which is not only robust to failure but also aware of the surrounding context in ubiquitous networks. In other words, the IT infrastructure required for u-commerce and the services, in nature, have some differences from those of traditional e-commerce or m-commerce in terms of the two kinds of supports required (fault tolerant and context awareness). On top of the needed infrastructure supports, the values of the u-commerce services could then be realized to the fullest extent (together with different directions of exploration on customers' needs in ubiquitous contexts).

REFERENCES

Agrawal, R., Lin, K, Sawhney, H., & Shim, K. (1995). Fast Similarity Search in the Presence of Noise, Scaling, and Translation in Time-Series Databases. *In Proceedings of the 21st International Conference on Very Large Databases (VLDB'95)*, Zurich, Switzerland.

Banavar, G., & Bernstein, A. (2002). Software Infrastructure and Design Challenges for Ubiquitous Computing Applications. *Communications of ACM, 45*(12), 92-96.

Benatallah, B., Dumas, M., Sheng, Q., & Ngu, A. (2002). Declarative Composition and Peer-to-Peer Provisioning of Dynamic Web Services. *In 18th International Conference on Data Engineering (ICDE'02)*, San Jose, CA, USA.

Bratman, M., Israel, D., & Pollack, M. (1988). Plans and Resource Bounded Practical Reasoning. *Computational Intelligence, 4*(4), 349-355.

Casati, F., Ilnicki, S., Jin, L., Krishnamoorthy, V., & M. Shan. (2002). *Adaptive and Dynamic Service Composition in eFlow*. HP Lab Technical Report HPL-2000-39, Palo Alto: Software Technology Laboratory.

Chakraborty, D. (2001). *Service Composition in Ad-hoc Environments*. Ph.D Dissertation Proposal, Technical Report TR-CS-01-20.

Chakraborty, D., Perich, F., Joshi, A., Finin, T., & Yesha, Y. (2002). *A Reactive Service Composition Architecture for Pervasive Computing Environments*. Technical Report TR-CS-02-02, University of Maryland at Baltimore, USA.

Chen, F. Y., & Yuan, S. T. (2004). *A Study on Contextualized Fault-tolerant Service Composition in WP2P Environments*. Technical Report, Fu-Jen University, Taiwan.

Deng, D., & Kasabov, N. (2000). ESOM: An Algorithm to Evolve Self-Organizing Maps from On-line Data Streams. *In Proceedings of the IJCNN'2000 on Neural Networks Neural Computing: New Challenges and Perspectives for the New Millennium, 6*, 3-8.

Dialani, V., Miles, S., Morcan, L., Rourc, D., & luck, M. (2002). Transparent Fault Tolerance for Web Services Based Architectures. *In Proceedings of the 8th International Euro-Par Conference (EURO-PAR'02)*, Paderborn, Germany.

Dipanjan, C. *et al*. (2002). A Reactive Service Composition Architecture for Pervasive Computing Environments. *7th Personal Wireless Communications Conference*, Singapore.

Fano, A., & Gershman, A. (2002). The Future of Business Services in the Age of Ubiquitous Computing. *Communications of ACM, 45*(12), 63-87.

Gribble, S. D., Welsh, M., Brewer, E. A., & Culler, D. E. (1999). *The NINJA project pages*. http://ninja.cs.berkeley.edu.

Hartigan, J. A., & Wong, M. A. (1979). A K-Means Clustering Algorithm. J. Royal Statistical Society, Ser. C, *Applied Statistics, 28,* 100-108.

Jena, http://www.hpl.hp.com/semweb/jena.htm

Keidl, M., Seltzsam, S., & Kemper, A. (2003). Reliable Web Service Execution and Deployment in Dynamic Environments. *Lecture Notes in Computer Science, 2819*, 104-118.

Lamport, L., Shostak, R., & Pease, M. (1982). The Byzantine Generals Problem. *ACM Transactions on Programming Languages and Systems, 4*(3), 382-401.

Mao, Z., Brewer, E., & Katz R. (2001). *Fault-tolerant, Scalable, Wide-area Internet Service Composition*. Technical Report UCB/CSD-1-1129, Department of Computer Science and Electrical Engineering, University of Maryland, Baltimore County, USA.

Mennie, D., & Pagurek, B. (2000). An Architecture to Support Dynamic Composition of Service Components. *In Proceedings of the 5th International Workshop on Component-Oriented Programming (WCOP 2000)*, Sophia Antipolis and Cannes, France.

Powell, D., Verissimo, P., Bonn, G., Waeselynck, F., & Seaton, D. (1988). The Delta-4 approach to dependability in open distributed computing systems. *In Proceedings of The 18th International*

Symposium on Fault-Tolerant Computing, Tokyo, Japan.

Project JXTA. http://www.jxta.org

Ranganathan, A., & Campbell, R. (2003). A Middleware for Context-Aware Agents in Ubiquitous Computing Environments. *ACM/IFIP/USENIX International Middleware Conference,* Rio de Janeiro, Brazil.

Schlichting, R., & Schneider, F. (1983). Fail-stop Processors: An Approach to Designing Fault-Tolerant Computing Systems. *ACM Transactions on Computer Systems, 1*(3), 222-238.

Schuster, H., Georgakopoulos, D., Cichocki, A., & Baker, D. (2000). Modeling and Composing Service-based and Reference Process-based Multi-enterprise Processes. *In Proceedings of the International Conference on Advanced Information Systems Engineering (CAiSE2000)*, Stockholm, Sweden.

Sheng, Q. Z., Benatallah, B., & Maamar, Z. (2004). Enabling Personalized Composition and Adaptive Provisioning of Web Services. *The 16th International Conference on Advanced Information Systems Engineering*, Riga, Latvia.

Sheng, Q., Benatallah, B., Dumas, M., & Mak, E. (2002). SELF-SERV: A Platform for Rapid Composition of Web Services in a Peer-to-Peer Environment. *In Proceedings of the 28th Very Large DataBase Conference (VLDB'2002)*, Hong Kong, China.

Song, Q., & Kasabov, N. (2001). ECM - A Novel On-line, Evolving Clustering Method and Its Applications. *In Proceedings of the Fifth Biannual Conference on Artificial Neural Networks and Expert Systems (ANNES2001)*, Dunedin, New Zealand.

Varshney, U., Vetter, R. J., & Kalakota, R. (2000). Mobile Commerce: A New Frontier. *IEEE Computer,* (pp. 32-38).

Weatherspoon, H., Moscovitz, T., & Kubiatowicz, J. (2002). Introspective Failure Analysis: Avoiding Correlated Failures in Peer-to-Peer Systems. *In Proceedings of the International Workshop on Reliable Peer-to-Peer Distributed Systems (SRDS'02)*, Osaka , Japan.

ENDNOTES

1. L, M, H respectively indicates low, medium and high in the willingness of a user to host a specified function.
2. Peers of vicinity are referring to those mobile peers accessible through wireless communication (e.g., Bluetooth).
3. RDF is best thought of in the form of node and arc diagrams.
4. The number of subtasks is used to substitute for the total time of subtask execution because the services in the simulation are "virtual" and doesn't really be executed.
5. The value of 1.4 is attained from dividing the best final overall solvability, i.e. k=4, by the worst final overall solvability, i.e. k=1. (0.337/0.241)
6. The value of 18% is attained from dividing the difference between the highest final overall relative network traffic, i.e. k=1, and the average of the final overall relative network traffic of the others, i.e. k=2,3,4, by the highest final overall relative network traffic. ((133.5-108.9)/133.5)
7. Since UbiSrvInt is the first pure P2P approach solution for ad-hoc wireless service composition, instead of providing benchmark evaluations this chapter aims for justifying the usefulness and the effectiveness of the approach proposed. The usefulness is originated from the nature of fault tolerance and context awareness for ad-hoc wireless service provision. The effectiveness subsequently is confirmed from the feasibility affirmation of the Fault-Tolerance Module and the manifesting demo scenarios of Reasoning Agent.

APPENDIX: UBISRVINT'S USE CASE DIAGRAM AND ACTIVITY DIAGRAM (REPRESENTED IN UML)

Figure A1. UbiSrvInt's use case diagram

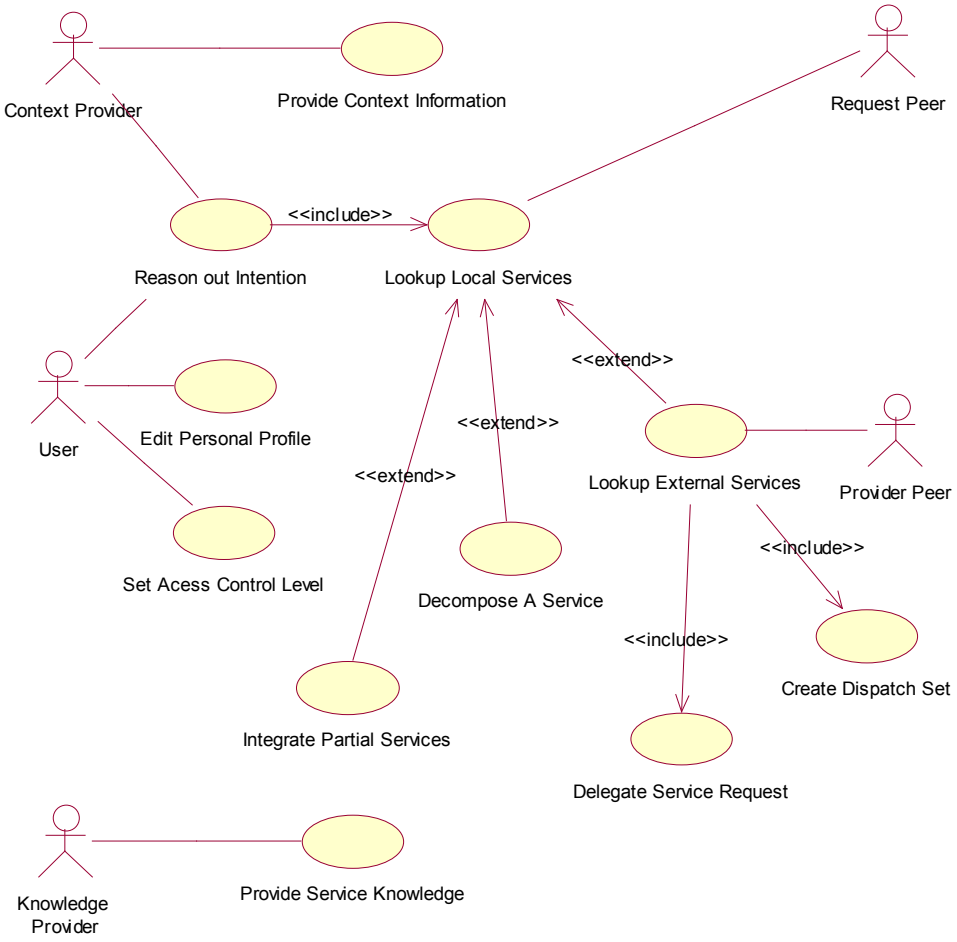


Figure A2. UbiSrvInt's activity diagram