Applied Mathematics & Information Sciences
An International Journal

http://dx.doi.org/10.18576/amis/100534

An Efficient Failure Recovery Scheme for Service Composition in Pervasive Computing

Chengyuan Yu and Linpeng Huang**

Department of Computer Science and Engineering, Shanghai Jiao Tong University, 200240, Shanghai, P. R. China

Received: 23 Apr. 2016, Revised: 22 Jun. 2016, Accepted: 23 Jun. 2016
Published online: 1 Sep. 2016

Abstract: During the execution of service composition, if one component service fails, a failure recovery mechanism is needed to ensure that the running process is not interrupted and the failed service can be replaced quickly and efficiently. In this paper, we propose an efficient failure recovery scheme for rapid reconstruction of services compositions. Sufficient conditions about substitution and keeping state-consistent between services are proposed. Further, the algorithm for keeping state-consistent between services is proposed. The innovation of this paper is that the failure service will be substituted and the failure service' state will be transformed into the substituting service' state to improve the performance of the failure recovery scheme. And the prototype system is implemented. Simulation experiments demonstrate the good performance of the proposed failure recovery scheme.

Keywords: pervasive computing, substitution, compatibility, service composition

1 Introduction

Pervasive computing [1] envisions seamless compositions of services from a wide variety of physical devices within our working and living spaces. These devices are intended to react to its environment and coordinate with each other via wireless or wired networks. Pervasive computing environments are envisioned to embody a number of devices with a very rich set of functionalities. By modeling the available resources as services and by designing a mechanism to access the available services, a pervasive computing application can be transformed into a service-oriented application.

In a service oriented environment, to achieve user's goals it is needed to find appropriate service. Furthermore, if service registry does not include desired service, composer component composes existing services and exploits it as a composite service to satisfy user's goal. A composite service is a combination of smaller services to provide value-added services that a single service cannot achieve. Services are in essence loosely-coupled and hosted by different providers and may not subject to rigid development, verification, or testing processes. Interoperability issues further reduce the reliability of services compositions. And any update of any service might affect critically the overall composition consistency, reliability and availability.

In this paper, we advocate the use of Process Algebra to describe and compose service at an abstract level. Process Algebras are adequate to describe services, because it can formally describe dynamic processes. Compared to the automata-based approaches, its main benefit is its expressiveness, particularly due to a large number of existing calculi enabling one to choose the most adequate formalism depending on its final use. Additionally, another interest of Process Algebra is that its constructs are adequate to specify composition due to its compositionality property.

We propose an efficient failure recovery scheme for service composition in pervasive computing. It can rapidly find the substitute service and keep state-consistent between a failed service and its substitute. Firstly service is characterized by Process Algebra. Secondly service composition and service substitution are introduced. Finally, sufficient conditions about substitution and algorithm for keeping state-consistent between services are proposed. The innovation of this paper is that the failure service will be substituted and the failure service' state will be transformed into the substituting service' state to improve the performance of the failure recovery scheme.

* Corresponding author e-mail: ycy8525@sjtu.edu.cn, huang-lp@cs.sjtu.edu.cn

This paper is organized as follows. Section 2 introduces relate works. Section 3 introduces background. Section 4 introduces the model of service, the concept of compatibility and substitution. Further, the sufficient conditions about substitution and keeping state-consistent between services and the algorithm of transforming the state between services are proposed. Section 5 presents the prototype and the performance of failure recovery scheme was evaluated. Section 6 concludes our paper.

2 RELATED WORK

Service failures and recovery strategies have been topics of keen interest to service researchers [2][3][4]. Luo, M. Y. and C. S. Yang propose an innovative mechanism that enables a web request to be smoothly migrated and recovered on another working node in the presence of server failure [5]. Salfner, F. proposes an approach to avoid failures as well as repair mechanisms by failure prediction on the system state [6]. B. Yao and W. K. Fuchs use a recovery proxy that caches client requests and service responses [7]. When a client reconnects, previous client requests or service responses are retrieved from the proxy. The one.world project [8] proposes a check-pointing mechanism that allows developers to capture the execution state of a component, and later to restore it to gracefully resume the execution of the component after a failure, such as power loss. It also enhances the robustness of pervasive computing systems by providing transaction-level persistence. Dedecker et al. [9] have proposed a domain-specific language, called AmbientTalk, introducing a distributed exception-handling mechanism [10] to deal with mobile hardware characteristics, such as connection volatility. This mechanism consists of a set of language constructs that enables to handle exceptions at different levels of granularity in the application code: message, block and collaboration.

in [30], context is applied to recovery of the web service business transaction and exceptions are classified into four categories: network exception, physical exception, service exception and user exception. Based on the classifications they define different start points of recovery process to simplify the recovery process of business transaction. Also evaluations on contexts of exceptions are performed to extract the feasible instances of compensation paths and the sensibility weights of path costs are utilized to select the optimal compensation instance.

in [31], they provide a behavioral signature model for service restructuring. It is considered for Web services to be choreography equivalence if they conform to the same behavioral signature model. Based on behavioral signature models and graph searching technologies, they provide an algorithm for restructuring the process model from within the service while assuring service choreography equivalence.

In THROWS [11], Neila Ben Lakhal, to achieve failure capturing and recovery, classify services as vital services and not-vital services. Aborting a vital service from the services compositions will induce aborting the whole services compositions, if there is no alternative web service to retry the execution of the failed service. In the other hand, aborting a not-vital service will not be reflected on the services compositions, thus these services compositions could complete successfully even though not all its components were committed. Hence, the probability of failures occurrence is reduced and the whole composition availability could be increased. In THROWS, the system maintains a CEL (Candidate Engine List) for every service.

Yuna Kim proposes a framework of Allowing User-Specified Failure Handling [12]. In this framework, an instance of executable business process for each user can be dynamically generated by receiving the failure acceptance specification that conforms to the user's preference. Failures occurring in a composite service are handled in an identical way to all users, although every user can demand different ways of failure handling on particular failures even in the same service. For example, in a travel agency service, let us consider a case where flight booking has been completed but hotel reservation is not available. Some people want to cancel the flight booking while the other people want to ignore the failure and proceed ahead of this process. It will improve the availability, reliability of service composition, but it is not the best way. Because there are many services that have the same interface in the environment of pervasive computing, finding a new service will be better.

In the modeling of services in process algebra has been discussed a lot in recent researches. Brogi [13] and Camara [14] chose the process notation CCS [15] to establish a formal foundation for WSCI [16] and BPEL [17] respectively. Lucchi [18] addressed the semantics of BPEL in terms of web which is an extension of Π-calculus [19]. Butler et al [20] targeted at giving a precise semantics to BPEL by define a straightforward mapping from BPEL to the compensation oriented language StAC [21].

In this paper, we not only propose the sufficient conditions about substitution and keeping state-consistent between services, but also propose the algorithm of transforming the state between services. If a failure occurs in a service, not only the service which can substitute the failure service, will be found, but also the state of failure service will be transformed into new service. The process is transparent to customers.

3 BACKGROUND

3.1 Process Algebra

In this subsection, our purpose is to give a flavour of what a Process Algebra is made up of. The algebra chosen in

this paper is CCS [15] whose set of operators is small yet sufficiently expressive for the presentation of our approach.

1) Basic operators: we firstly give a brief introduction to syntax of CCS process. A process (ranged over by P, Q, R, M etc.) in CCS is defined by:

$$P ::= 0 \,|\, \alpha .Q \,|\, P+Q \,|\, P \| Q \,|\, P \backslash sm$$

$$\alpha ::= a?(x) \,|\, a!(x) \,|\, \tau$$

0 dentoes termination. Action is either to receive or send a message through a channel a. a? denotes receiving message, a! denotes sending message. A CCS process executes a sequence of the form α.Q. P + Q behaves like either P or Q but not both. The coexistence of processes P and Q whose execution is interleaved is written$P \| Q$(run P and Q in parallel). The "$\|$" symbol, called parallel composition, is therefore used to define a global process made up of several subprocesses.

The restriction operator, noted P\sm where P is a process and sm is a set of channel names, imposes that sending of a message through channel $m \in sm$ by one sub-process of P can occur only if another sub-process of P does a reception through the same message channel m. In some cases we need an additional symbol τ denoting hidden actions. τ actions allow some level of abstraction in the description of the processes, since they can hide an arbitrarily complex sequence of actions whose details are kept private.

2) Operational semantics: some axiomatic and operation semantics of CCS adapted from [15] are defined by a transition system as below. They describe the possible derivations of a process.

Rule1: $\overline{e.P \xrightarrow{e} p}$

Rule2: $\frac{P \xrightarrow{e} P'}{P+Q \xrightarrow{e} P'}$

Rule3: $\frac{P \xrightarrow{e} P'}{P \| Q \xrightarrow{e} P' \| Q}$

Rule4: $\frac{P \xrightarrow{e?} P', Q \xrightarrow{e!} Q'}{P \| Q \xrightarrow{\tau} P' \| Q'}$

For instance, the Rule1 states that a process e.P can evolve to a process P by performing e and the Rule2 states that a process involving a choice can evolve following one of the processes of the choice − this discards the other alternative. Rule3 and Rule4 state that parallel processes evolve through synchronization on action e or an internal τ action.

3) Bi-simulation

The notion of Bi-simulation was introduced and now plays a central role: Modal Logic, Concurrency Theory, Set Theory, Formal Verification, etc. [26]. In Modal Logic the notion was introduced by van Benthem [27]. In Concurrency Theory Milner and Park introduced the notion for testing observational equivalence of the Calculus of Communicating Systems (CCS). In Set Theory, Forti and Honsell [28] introduced Bi-simulation.

A Bi-simulation is a binary relation R on P, satifying:

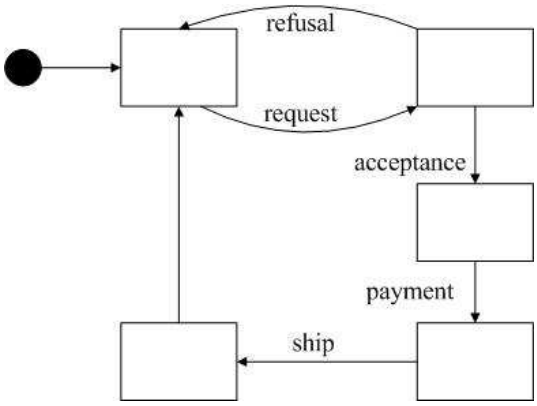


Fig. 1: a service for shopping

If $R(p,q)$ and $p \xrightarrow{a} p'$, then there is a q' such that $q \xrightarrow{a} q'$ and $R(p',q')$

If $R(p,q)$ and $q \xrightarrow{a} q'$, then there is a p' such that $P \xrightarrow{a} P'$ and $R(p',q')$

If $R(p,q)$ then $p \xrightarrow{a} 0$ if and only if $q \xrightarrow{a} 0$.

Intuitively, Bi-simulation guarantees that the branching structure of two processes is equal, provided that equal branches may be identified. The idea is that if one process is capable of performing an a-step to a new state, then any equivalent process should be able to do an a-step to a corresponding state. Bi-simulation serves as a means of checking equivalence between processes.

4 SUBSTITUTION

Substitution occurs when in a composite scenario the old service comes to be temporarily unreachable, or when a new release of a service is proposed which provides better functionalities, better Quality of Service, or has lower cost [29].

4.1 Specifying Service as Process

Service has internal states which would be affected by the messages it sends and receives. Some actions would be possible in some states and impossible in some others. Then we must know which messages would be received or sent in every state. Then even a simplistic service would involve complex interactions, such as shopping service shown on Figure 1.

The shopping service would be described as following CCS expression:

proc Shop = request?.shoppingProcess

proc shoppingProcess = refusal! .Shop + acceptance! .payment? .ship! .Shop

CCS is an effective and unambiguous way to represent such services' behaviours. The CCS expression of service

is adapted to formal reasoning and the description of large-scale systems.

4.2 Notions of Compatibility and Substitution

The paper shall focus only on behavioural aspects, assuming that the names of the exchanged messages are standardised and that semantic compatibility is guaranteed. Compatibility is moreover closely related to another problem, substitutability: when one service can be replaced by another without introducing some flaws into the whole system? Ensuring the substitutability of a previously used service by a new one is necessary in many situations, for instance, when the old service comes to be temporarily unreachable, or when a new release of a service is proposed which provides better functionalities, better Quality of Service, or has lower cost. This paper introduces the notion of compatibility proposed by [22].

We use the notation $\bar{m}$ to represent the opposite action of action m, i.e., we define $\overline{n!}$ as n? and $\overline{n?}$ as n!. We generalise this notation to sequences, defining the sequence of action $\bar{\alpha}$ as $\overline{a_1} \bullet \overline{a_2} \bullet ... \overline{a_n}$. $|\alpha|$ represents the length of sequence of action .

Theorem 4.1. if there is a Bi-simulation relation R between process P and process Q and $P \xrightarrow{\alpha} P_m$ for the sequence of actions α, then there is a Q_n such that $Q \xrightarrow{\alpha} Q_n$ and R(P$_m$, Q$_n$).

Proof. this theorem will be proved by mathematical induction.

When $|\alpha| = 1$, the theorem holds according to the definition of Bi-simulation.

Suppose the theorem holds for $|\alpha| =$ n. that is that if $P \xrightarrow{\alpha} P_m$, then there is a Q_n such that $Q \xrightarrow{\alpha} Q_n$ and R(P$_m$, Q$_n$).

Consider the case $|\alpha| =$ n+1.

We can suppose $\alpha = \beta \bullet x$ and $P \xrightarrow{\beta} P' \xrightarrow{x} P_m$ where $|\beta| =$ n, $|x| = 1$ and x is action a? or a!.

By the above assumption and $|\beta| = n$, we have $P \xrightarrow{\beta} P'$, then there is a Q' such that $Q \xrightarrow{\beta} Q'$ and R(P', Q'). According to R(P', Q'), $P' \xrightarrow{x} P_m$,$|x| = 1$ and the definition of Bi-simulation, we can infer that there is a Q_n such that $Q' \xrightarrow{x} Q_n$ and R(p_m, Q_n). Then according to $Q \xrightarrow{\beta} Q'$, $Q' \xrightarrow{x} Q_n$ and $\alpha = \beta \bullet x$, we can get $Q \xrightarrow{\alpha} Q_n$. So there is a Q_n such that $Q \xrightarrow{\alpha} Q_n$ and R(p_m, Q_n) when $|\alpha| =$ n+1.

That is, the theorem holds for $|\alpha| =$ n+1. By the principle of mathematical induction, we conclude that if there is a Bi-simulation relation R between process P and process Q and $P \xrightarrow{\alpha} P_m$ for the sequence of actions α, then there is a Q_n such that $Q \xrightarrow{\alpha} Q_n$ and R(p_m, Q_n).

Definition(reachable pair of states). pair of states $<$ P$_m$, Q$_n$ $>$ about process P and Q is reachable, if and only if $P \| Q \xrightarrow{\tau} P_m \| Q_n$.

Theorem 4.2. there are sequences of actions α and $\bar{\alpha}$ such that $P \xrightarrow{\alpha} P_m$ and $Q \xrightarrow{\bar{\alpha}} Q_n$, if and only if $P \| Q \xrightarrow{\tau} P_m \| Q_n$.

Proof. $\Rightarrow$: this will be proved by mathematical induction.

When $|\alpha| = 1$, it holds since we can get$P \| Q \xrightarrow{\tau} P_m \| Q_n$ immediately according to Rule4.

Suppose it holds for $|\alpha| = n$. That is that if $P \xrightarrow{\alpha} P_m$ and $Q \xrightarrow{\bar{\alpha}} Q_n$, then $P \| Q \xrightarrow{\tau} P_m \| Q_n$ (according to Rule4).

Consider the case $|\alpha| = n+1$.

We can suppose $\alpha = \beta \bullet x$ and $P \xrightarrow{\beta} P' \xrightarrow{x} P_m$ and $Q \xrightarrow{\bar{\beta}} Q' \xrightarrow{\bar{x}} Q_n$, where $|\beta| =$ n and x is action a! or a?.

By the above assumption, we have $P \xrightarrow{\beta} P'$, $Q \xrightarrow{\bar{\beta}} Q'$ and Rule4, then $P \| Q \xrightarrow{\tau} P' \| Q'$. According to Rule4 and $P' \xrightarrow{x} P_m$ and $Q' \xrightarrow{\bar{x}} Q_n$, we can get $P' \| Q' \xrightarrow{\tau} P_m \| Q_n$. According to $P \| Q \xrightarrow{\tau} P' \| Q'$ and $P' \| Q' \xrightarrow{\tau} P_m \| Q_n$, we can get $P \| Q \xrightarrow{\tau} P_m \| Q_n$.

That is, it holds for $|\alpha| = n+1$. By the principle of mathematical induction, we conclude that if $P \xrightarrow{\alpha} P_m$, $Q \xrightarrow{\bar{\alpha}} Q_n$ and Rule4, then $P \| Q \xrightarrow{\tau} P_m \| Q_n$ for any sequence of action α.

$\Leftarrow$: In order to prove necessary condition holds, we would prove Inverse Negative Proposition of the necessary conditions holds. The Inverse Negative Proposition of the necessary condition is that there are sequences of action α and β satisfying $P \xrightarrow{\alpha} P_m$, $Q \xrightarrow{\beta} Q_n$ and $\alpha \neq \bar{\beta}$ such that $P \| Q \xrightarrow{\delta} P_m \| Q_n$ and $\delta \neq \tau$. $\alpha \neq \bar{\beta}$ corresponds to the following cases (only under three circumstances are the 2 sequences, a and b, not equal. they are respectively: the previous i characters are the same; a is the prefix of b; or b is the prefix of a.).

Case 1: suppose that $\alpha = \alpha_1 \bullet ... \bullet \alpha_{i-1} \bullet \alpha_i \bullet \gamma$, $\beta = \bar{\alpha}_1 \bullet ... \bullet \bar{\alpha}_{i-1} \bullet \beta_i \bullet \theta$, $P \xrightarrow{\alpha_1 \bullet ... \bullet \alpha_{i-1}} P_{i-1} \xrightarrow{\alpha_i} P_i \xrightarrow{\gamma} P_m$, $Q \xrightarrow{\bar{\alpha}_1 \bullet ... \bullet \bar{\alpha}_{i-1}} Q_{i-1} \xrightarrow{\beta_i} Q_i \xrightarrow{\theta} Q_n$ where γ and θ are arbitrary sequence of actions and $\alpha_i \neq \beta_i$. According to the above proof, we can get $P \| Q \xrightarrow{\tau} P_{i-1} \| Q_{i-1}$. Using Rule3 and Rule4, we can get $P_{i-1} \| Q_{i-1} \xrightarrow{\alpha_i \bullet \sigma} P_m \| Q_n$ or $P_{i-1} \| Q_{i-1} \xrightarrow{\beta_i \bullet \omega} P_m \| Q_n$, where the first action of σ is α_{i+1} or β_i and the first action of ω is α_i or β_{i+1}. So we can get $P \| Q \xrightarrow{\tau \bullet \alpha_i \bullet \sigma} P_m \| Q_n$ or $P \| Q \xrightarrow{\tau \bullet \beta_i \bullet \omega} P_m \| Q_n$. Because of $\alpha_i \neq \bar{\beta}_i$, $\alpha_i \neq \bar{\alpha}_{i+1}$ and $\beta_i \neq \bar{\beta}_{i+1}$ (service can not sent messages to themselves, so we have $\alpha_i \neq \bar{\alpha}_{i+1}$ and $\beta_i \neq \bar{\beta}_{i+1}$), we can infer that $\tau \bullet \alpha_i \sigma \neq \tau$ and $\tau \bullet \beta_i \omega \neq \tau$. Then in this case Inverse Negative Proposition holds.

Case 2: suppose $\alpha = \bar{\beta} + \gamma$ and $P \xrightarrow{\bar{\beta}} P' \xrightarrow{\gamma} P_m$ where $\gamma \neq \tau$. Using Rule3 and Rule4, we can get $P \Big\| Q \xrightarrow{\tau} P' \| Q_n \xrightarrow{\gamma} P_m \| Q_n$. Immediately we can get $P \| Q \xrightarrow{\tau \bullet \gamma} P_m \| Q_n$ and $\tau \bullet \gamma \neq \tau$. So in this case Inverse Negative Proposition holds.

Case 3: suppose $\beta = \bar{\alpha} + \gamma'$ and $Q \xrightarrow{\bar{\alpha}} Q' \xrightarrow{\gamma'} Q_n$ where $\gamma' \neq \tau$. Using Rule3, Rule4 and $\gamma' \neq \tau$, we can get $P \Big\| Q \xrightarrow{\tau} P_m \| Q' \xrightarrow{\gamma'} P_m \| Q_n$. Immediately we can get $P \| Q \xrightarrow{\tau \bullet \gamma'} P_m \| Q_n$ and $\tau \bullet \gamma' \neq \tau$. So in this case Inverse Negative Proposition holds.

Then The Inverse Negative Proposition of the necessary condition holds, in other words necessary condition holds. So Theorem 4.2 holds.

Definition(Compatibility). Two processes P and Q are compatible if, for any reachable pair of states < P', Q' >, we have: emission(P, P') = reception(Q, Q') and emission(Q, Q') = reception(P, P').

emission(P, P_s) = {message $\Big| P \xrightarrow{\alpha} P_s$ and P_s = $\sum e_i.P_i$ and message $\in \{e_1!, \ldots, e_m!\}\}$. emission(P, P_s) as the set of actions which process P can send when it is in state P_s.

reception(P, P_s) = {message $\Big| P \xrightarrow{\alpha} P_s$ and P_s = $\sum e_i.P_i$ and message $\in \{e_1?, \ldots, e_n?\}\}$. reception(P, P_s) as the set of actions which process P can receive when it is in state P_s.

Definition(Substitutability). A service A' described as process $P_{A'}$, can substitute a service A described as process P_A, if $P_{A'}$ is compatible with any process P_B which is compatible with P_A.

Theorem 4.3. service A described as process P_A, and service B described as process P_B, can substitute each others, if there is a Bi-simulation relation R between process P_A and process P_B.

Proof. proof by contradiction will be used. Suppose that there is a Bi-simulation relation R between process P_A and process P_B, and there is a process P_C which is compatible with process P_A, but not compatible with process P_B. Then there is at least one reachable pair of states < P_{Cn}, P_{Bm} > of process P_C and P_B, satisfying emission(P_C, P_{Cn}) ≠ reception(P_B, P_{Bm}) or emission (P_B, P_{Bm}) ≠ reception(P_C, P_{Cn}). According to the Definition of reachable pair of states, Theorem 4.2 and reachable pair of states $< P_{Cn}, P_{Bm} >$, we can get that there are sequences of actions α and $\bar{\alpha}$ so that $P_B \xrightarrow{\alpha} P_{Bm}$ and $P_C \xrightarrow{\bar{\alpha}} P_{Cn}$. According to Theorem 4.1, Bi-simulation relation R between process P_A and P_B and $P_B \xrightarrow{\alpha} P_{Bm}$, then there is a state P_{Ak} such that $P_A \xrightarrow{\alpha} P_{Ak}$ and R(P_{Ak}, P_{Bm}). Using Definition of reachable pair of states and Theorem 4.2, we can infer that pair of states < P_{Cn}, P_{Ak} > is reachable from $P_C \xrightarrow{\bar{\alpha}} P_{Cn}$ and $P_A \xrightarrow{\alpha} P_{Ak}$.

Because there is a Bi-simulation relation R between P_{Ak} and P_{Bm}, we can infer that emission (P_B, P_{Bm}) = emission (P_A, P_{Ak}) and reception(P_B, P_{Bm}) = reception(P_A, P_{Ak}) from the definition of Bi-simulation. Because emission(P_C, P_{Cn}) ≠ reception(P_B, P_{Bm}) or emission (P_B, P_{Bm}) ≠ reception(P_C, P_{Cn}), we can infer reachable pair of states < P_{Cn}, P_{Ak} > satisfying emission(P_C, P_{Cn}) ≠ reception(P_A, P_{Ak}) or emission (P_A, P_{Ak}) ≠ reception(P_C, P_{Cn}). According to the definition of Compatibility, we can infer that P_A and P_C are not Compatibility and it conflicts with the supposition. So, P_B is compatible with any P_C which is compatible with process P_A. Then, service B can substitute service A. Similarly, we can proof that service A can substitute service B. then this Theorem holds.

So far, there are a lot of efficient algorithmic solutions to the problem of determining a Bi-simulation relation on processes, such as [23][24][25][26]. Then it can be directly used to determine whether there is a Bi-simulation relation on services.

Fig. 2: Composite service of online purchase

4.3 keep state-consistent between a failed service and its substitute

When a failure occurs in service, generally failure handlers or compensation operators will be called and then the failure service will be substituted by other service. But it has some disadvantages. For example, we consider an application dedicated to the online purchase. This application is carried out by a composite service as illustrated by figure 2. Let us consider a case where payment has been completed but delivering service is not available. In general, money is given back to the customer, and the customer requirements specification is undone. Obviously, it is not the best way.

The failure recovery scheme proposed by this paper not only substitutes the failure service, but also keeping state-consistent between failure service and substituting service to improve customer's experience.

From Theorem 4.1 and Theorem 4.3, we can get that if there is a Bi-simulation relation R between P and Q, then service P and Q can substitute each other, and we can quickly keep state-consistent between P and Q. If the sequence of actions α (in the model of service, message correspond with action) such that $P \xrightarrow{\alpha} P'$, has been recorded, then we can quickly get the state Q' such that $Q \xrightarrow{\alpha} Q'$ and R(Q', P'). Then we can be enlightened that if the sequence of messages of a service can be recorded, then it could be used to keep state-consistent between services where a Bi-simulation relation R exists. The algorithm of transforming the state of the old service into the new service is described as figure 3.

```
reachState(service A,string sequenceOfMessage)
{   message = getFristMessage(sequenceOfMessage);
    while(message != null)
    {
       if message.type == inputMessage
          send(message, A);
       if message.type == outputMessage
          wait until message == outputMessageOfServiceA;
       message = getFristMessage(sequenceOfMessage);
    }
}
```

Fig. 3: The algorithm of transforming the state of the old service into the new service

4.4 service pool

In order to improve the availability and reliability of service composition, we introduce the concept of service pool. The concept of service pool is regarded as an abstract business service. It does not refer to any concrete service, but represents a group of services which perform a specific common function. In this paper, services in the same service pool are Bi-similar with each other. So services in the same service pool can substitute each other and using the algorithm of transforming the state of the old service into the new service to keep state-consistent between services.

5 PROTOTYPE AND EVALUATION

A prototype of the system is constructed to evaluate the performance of the proposed failure recovery scheme. Figure 4 shows the overall architecture. The architecture consists of three main parts: Service Pools Layer, Agents Layer, and Failure Recover Modules.

Service Pools Layer: it is responsible for dynamic creation and organization of service pools. And it is also able to classify all available services, so that services in the same service pools, can substitute with each other. Further, it can rank and dynamic adjust the ranking of the service in the same service pools to ensure that the best services will be selected first.

Agents Layer: it can build the new agents, redirect the agents to new services, delete the old agents, record the sequence of messages received from clients and send by services, and provide interface to retrieve the sequence of messages whenever it is necessary. Agent can directly interchange with a service, it sent messages which were sent by client, to the service, or sent messages which were sent by service, to the client.

Failure Recover Modules: it consists of Monitor Module, Coordination Module and State Transformation Module. It is responsible for discovery service failure, searching new availability service to substitute the failure service, transformation the state of failure service into the new service.

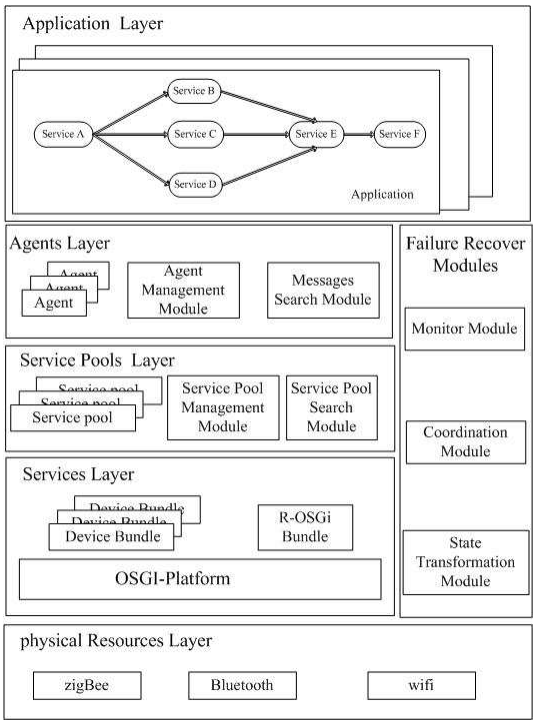


Fig. 4: The architecture for rapid reconstruction of services compositions

5.1 The process of the reconstruction of service composition

If Monitor Module captures a service failure, it will send a message which contains the failure information to the Coordination Module. Then Coordination Module sends a service failure message to Service Pool Management Module. Service Pool Management Module will revaluate the quality the failure service and adjust the ranking of the service in the same service pools, and call Service Pool Search Module for return the new availability service to Coordination Module. After receiving the new availability service, Coordination Module will call Messages Search Module to get the sequence of messages which were received from client and send by service before the failure occurring. Then State Transformation Module will be invocated by Coordination Module. State Transformation Module will transform the state of failure service into the new availability service returned by Service Pool Search Module, using the sequence of messages returned by Messages Search Module. After that, Coordination Module will call Agent Management Module for redirecting the agent to the new availability service. Then service failure has been recovery. The process of failure recovery is shown in figure 5.

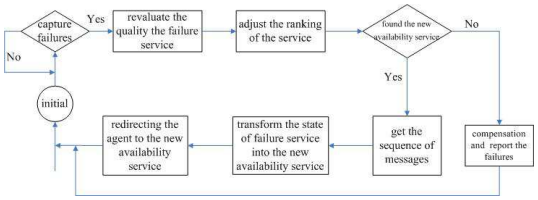


Fig. 5: the process of the reconstruction of service composition

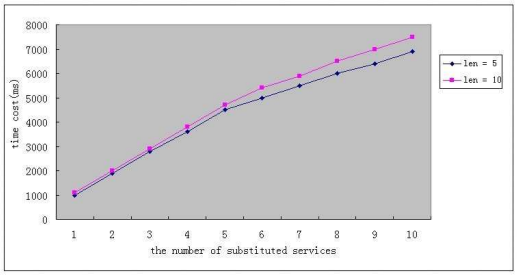


Fig. 6: The time cost of failure recovery

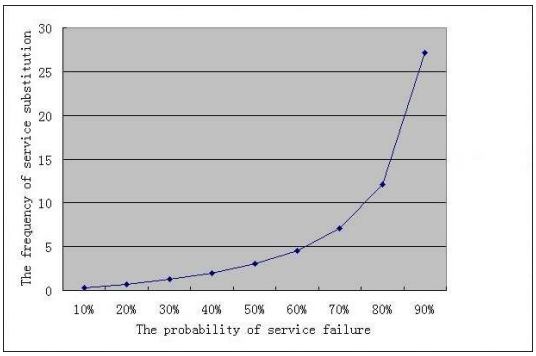


Fig. 7: The frequency of service substitution

5.2 Evaluation

In this section, In order to test the performance of substituting service and transforming the state of old service into the new service, the empty services which only receive and send messages without any process, was used. As shown in Figure 6 and Figure 7. In the Figure 6, the concept of len is defined as the length of the sequence of messages which were received or send by the service before failure occurring in the service. For example, len = 5 means that there are five messages which were send or received by the failure services before the failure occurring. Each result is the average of 1000 simulated executions. If the number of messages which were send or received by the failure services before the failure occurring, is no more than ten, The time cost of substituting a service and transforming the state of old service into the new service is about 700ms, which is completely acceptable for users.

Since the longer the message, the longer the time required to keep state-consistent, performance of len=10 is lower than that of len=5. since several substitute services will be acquired and stored in cache each time it is searched for, following failures of this service could be substituted by the substitute services already stored. thus when failures occur repeatedly, the average time of the failure recovery will be shortened.

In Figure7, the service composition consists of three services in sequence. for convenience, we suppose the failure probability of every component service is equal, and that of the substitute service and original service is equal too. The result of the experiment shows that when the failure probability of service is comparably low, this method is fairly effective. But when the failure probability of service is comparably high, lots of substitutions will be needed, and the consequence is that the performance of the service composition will be degraded rapidly.

6 CONCLUSION

In this paper, we propose an efficient failure recovery scheme for rapid reconstruction of services compositions in pervasive computing. Sufficient conditions about substitution and keeping state-consistent between services are proposed. Further, the algorithm for keeping state-consistent between services is proposed. The innovation of this paper is that the failure service will be substituted and the failure service' state will be transformed into the substituting service' state to improve the performance of the failure recovery scheme. Finally, the prototype system is implemented. It can rapidly find the substitute service for failure service, and keep state-consistent between the failure service and its substitute. Simulation experiments demonstrate the good performance of the proposed failure recovery scheme.

Acknowledgement

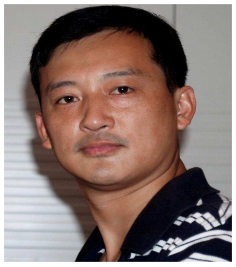
The work described in this paper was supported by the National Natural Science Foundation of China under Grant No. 9111800461232007, the 973 Program of China (No. 2009CB320705) and the Innovation Program of Shanghai Municipal Education Commission (No. 13ZZ023).

References

[1] M.Weiser, The computer for the 21st century. Scientific American 265(3): 94-104. 1991.
[2] Gronroos, C. (1988). Service quality: The six criteria of good perceived service quality. Review of Business, 9(Winter), 10-13, 1988.

[3] McCollough, M., Berry, L., Yadav, M. (2000). An empirical investigation of customer satisfaction after service failure and recovery. Journal of Service Research, 3(2), 121-137, 2000.
[4] Tax, S. S., Brown, S. W., Chandrashekaran, M. (1998). Customer evaluations of service complain experiences: Implications for relationship marketing. Journal of Marketing, 62(April), 60-76,1998.
[5] Luo, M. Y. and C. S. Yang. Enabling fault resilience for web services. Computer Communications 25(3): 198-209. 2002.
[6] Salfner, F., G. Hoffmann, et al. Prediction-based software availability enhancement. Self-Star Properties in Complex Information Systems: 143-157. 2005.
[7] B. Yao and W. K. Fuchs, "Proxy-based Recovery for Applications on Wireless Hand-held Devices". In Proc. 19th IEEE Symposium on Reliable Distributed Systems (SRDS'00), October 16-18, 2000, pp. 2-10.
[8] G. R. One.world: Experiences with a pervasive computing architecture. IEEE Pervasive Computing, 3(3):22-30, 2004. ISSN 1536-1268. doi: 10.1109/MPRV.2004.1321024.
[9] J. Dedecker, T. V. Cutsem, S. Mostinckx, T. D'Hondt, and W. De Meuter. Ambient-oriented programming in AmbientTalk. In Proceedings of the 20th European Conference on Object-Oriented Programming (ECOOP '06), pages 230-254, Berlin, Heidelberg, 2006. Springer-Verlag. ISBN 3-540-35726-2.
[10] S. Mostinckx, J. Dedecker, E. G. Boix, T. V. Cutsem, and W. D. Meuter. Ambient-oriented exception handling. In C. Dony, J. L. Knudsen, A. B. Romanovsky, and A. Tripathi, editors, Advanced Topics in Exception Handling Techniques, volume 4119 of Lecture Notes in Computer Science, pages 141-160. Springer, 2006. ISBN 3-540-37443-4.
[11] Neila Ben Lakhal , Takashi Kobayashi , Haruo Yokota, THROWS: An Architecture for Highly Available Distributed Execution of Web Services Compositions, Proceedings of the 14th International Workshop on Research Issues on Data Engineering: Web Services for E-Commerce and E-Government Applications (RIDE'04), p.103-110, March 28-29, 2004
[12] Yuna Kim, Jong Kim . Allowing user-specified failure handling in web services composition .The Second International Conference on Ubiquitous Information Management and Communication, Pages 452-458,February 2008 .
[13] A. Brogi, C. Canal, E. Pimentel, and A. Vallecillo. Formalizing web service choreographies. Electronic Notes in Theoretical Computer Science, 105:73-94, 2004.
[14] J. Camara, C. Canal, J. Cubo, and A. Vallecillo. Formalizing WSBPEL business processes using process algebra. In Proc. of FOCLASA'05, ENTCS 154, pages 159-173. Elsevier, 2006.
[15] R. Milner. Communication and Concurrency. Prentice Hall, 1989.
[16] World Wide Web Consortium. Web Service Choreography Interface (WSCI) 1.0, 2002. http://www.w3.org/TR/wsci.
[17] F. Curbera, Y. Goland, J. Klein, F. Leymann, D.Roller, S.Thatte, and S. Weerawarana. Business Process Execution Language for Web Services (BPEL4WS 1.1), 2003. http://www-106.ibm.com/developerworks/webservices/library/wsbpel/.
[18] R. Lucchi and M. Mazzara. A pi-calculus based semantics for WS-BPEL. Journal of Logic and Algebraic Programming, 2006.
[19] R. Milner. Communicating and Mobile Systems: the -Calculus. Cambridge University Press, 1999.
[20] M. Butler, C. Ferreira, and M. Y. Ng. Precise modeling of compensating business transactions and its application to BPEL. Journal of Universal Computer Science, 2005.
[21] M. Butler and C. Ferreira. A process compensation language. In Proc. of IFM2000, LNCS 1945, pages 61-76. Springer, 2000.
[22] Lucas Bordeaux, et al. (2005). "When are two web services compatible?" Technologies for E-Services: 15-28.
[23] J. E. Hopcroft. An n log n algorithm for minimizing states in a finite automaton. In Kohavi and Paz, editors, Theory of Machines and Computations, pages 189-196. Academic Press, 1971.
[24] Hirshfeld, Y., M. Jerrum, et al. (1996). "A polynomial-time algorithm for deciding bisimulation equivalence of normed Basic Parallel Processes." Mathematical Structures in Computer Science 6(03): 251-259.
[25] Lasota, S. and W. Rytter (2006). "Faster algorithm for bisimulation equivalence of normed context-free processes." Mathematical Foundations of Computer Science 2006: 646-657.
[26] Dovier, A., C. Piazza, et al. (2004). "An efficient algorithm for computing bisimulation equivalence." Theoretical Computer Science 311(1-3): 221-256.
[27] J. van Benthem. Modal Correspondence Theory. PhD thesis, Universiteit van Amsterdam, Instituut voor Logica en Grondslagenonderzoek van Exacte Wetenschappen, 1976.
[28] M. Forti and F. Honsell. Set theory with free construction principles. Annali Scuola Normale Superiore di Pisa, Cl. Sc., IV(10):493-522, 1983.
[29] V. De Antonellis, M. Melchiori, B. Pernici, and P. Plebani. A methodology for e-service substitutability in a virtual district environment. In Proc. of Conf. on Advanced Information Systems Engineering (CAISE), pages 552-567. Springer, 2003.
[30] Jiuxin Cao , Junzhou Luo , Song Zhang .A Context-Aware Recovery Mechanism for Web Services Business Transaction. 2012 IEEE Ninth International Conference on Services Computing,pages 352-359, 24-29 June 2012.
[31] Zaiwen Feng, Rong Peng, Keqing He, Zhou He. Service Restructuring by Choreography-driven Equivalence. 2012 IEEE Ninth International Conference on Services Computing,pages 407-414, 24-29 June 2012.

Chengyuan Yu received his B.S and M.S degree in computer science from Jiangxi Normal University. He is currently working on towards Ph.D. degree in computer science at computer science and engineering department of Shanghai Jiao Tong University, Shanghai, China. His current research interests include ASM, formal analysis and verification service composition and dynamic service update in pervasive computing.



Linpeng Huang received his MS and PhD degrees in computer science from Shanghai Jiao Tong University in 1989 and 1992, respectively. He is a professor of computer science in the department of computer science and engineering, Shanghai Jiao Tong University. His research interests lie in the area of distributed systems, formal verification techniques and service oriented computing.