Achieving Dependable Composite Services Through Two-Level Redundancy

Hailong **Sun,** *School of Computer Science and Engineering, Beihang University, China*
Jin Zeng, *China Software Testing Center (CSTC), China*
Huipeng Guo, *School of Computer Science and Engineering, Beihang University, China*

BIOGRAPHY

Hailong Sun is an Assistant Professor with the School of Computer Science and Engineering, Beihang University, Beijing, China. He received his Ph.D. in Computer Software and Theory from Beihang University, and B.S. degree in Computer Science from Beijing Jiaotong University in 2001. In 2008, he worked for AT&T Labs-Research as a visiting scholar for about half a year. His research interests include web services, service oriented computing and distributed systems. He is a member of IEEE and ACM.
Jin Zeng is a developing and testing engineer at China Software Testing Center (CSTC) in Beijing of China. He received his Ph.D. in Computer Software and Theory from the School of Computer Science and Engineering of Beihang University in 2010. His research interests include service computing, dynamic evolution of composite service, business process management, and workflow evolution. Now he is engaged in researching cloud testing and middleware testing technologies.
Huipeng Guo is a postdoctor with the School of Economics and Management, Beihang University, Beijing, China. He received his Ph.D. in Computer Software and Theory from Beihang University in 2009. His main research interests include dependable software and service computing.

KEYWORD

Dependability; service composition; redundancy; web service; service computing; SOA

ABSTRACT

Service composition is a widely-accepted method to build service-oriented applications. However, due to the uncertainty of infrastructure environments, service performance and user requests, service composition faces a great challenge to guarantee the dependability of the corresponding composite services. In this chapter, we provide an insightful analysis of the dependability issue of composite services. And we present a solution based on two-level redundancy: component service redundancy and structural redundancy. With component service redundancy, we study how to determine the number of backup services and how to guarantee consistent dependability of a composite service. In addition, structural redundancy aims at further improving dependability at business process level through setting up backup execution paths.

INTRODUCTION

Service oriented computing is known as a new computing paradigm that utilizes services as fundamental elements for developing applications. A traditional way of software development is based on the "divide-and-conquer" manner, in which software is divided into several modules and modules are implemented separately. Instead, service oriented computing provides a novel approach to building new software appli-

cations through the reuse of existing services. A service is a self-describing entity that can be discovered and accessed through standard-based protocols. Thus software development with service oriented computing is more about integrating existing services to meet application requirements. Intuitively, software productivity can be much improved with service oriented computing technologies. The underlying technologies generally include SOA (Service Oriented Architecture) and Web services. Following the service-oriented architecture, Web service supports better interoperability, higher usability, and increased reusability compared to traditional software technologies such as RMI and CORBA. According to(Zhang, Zhang, & Cai, 2007), the lifecycle of an SOA solution consists of modeling, development, deployment, publishing, discovery, composition, collaboration, monitoring and management. Especially, service composition is widely considered as an effective method to support the development of business applications using loosely-coupled and distributed web services over the Internet. As a result, a software application built in this approach exists in the form of a composite service, which is composed of a business process structure and a set of component services. One of the challenges faced by service composition is how to make a system adaptable to rapid changing user requirements and runtime environments. One way to address this issue is late-binding, which means that only abstract services are specified in the process of modeling and development while concrete services are chosen in runtime. To be more specific, service oriented software is finally implemented and instantiated in runtime. In other words, the development and running of service oriented software can not be separated clearly like traditional software. Therefore, service composition incorporates both design time and runtime issues.

Due to the highly dynamic and uncontrolled Internet environments, composite service dependability is one of the most important challenges to deal with in the field of service composition. The dependability property of composite services is of great importance to users, which includes many critical factors, such as availability, reliability and so on. A service with higher availability promises more functional? time and a more reliable service reduces the probability to fail when it is invoked. A key issue to service composition is that the dependability of a composite service may change dynamically over time. Users however, desire that a composite service delivers consistent dependability, which makes it possible for the users to obtain the expected results. This is particularly important when the application is mission-critical, e.g., a disaster management. When the dependability of a composite service degrades, users may severely suffer from the decreased performance.

The intrinsic dynamicity of composite services stems from the fact that a composite service is composed of many component services that potentially belong to different providers distributed over the open Internet. First, each component service is subject to a specific environment and various changing factors, such as varying system load and available bandwidth. Second, a more complicated situation is that multiple component services simultaneously fail to deliver the required dependability.

This chapter is devoted to discussing the dependability of composite services, which is a critical issue for service-oriented applications. Redundancy is a well-known method to obtain the needed dependability in distributed systems. According to the aforementioned analysis, we aim at dealing with this issue from two aspects respectively: component service redundancy and structural redundancy. The component service redundancy is adopted to select a substituent service for a failed component service; while the structural redundancy is designed for dealing with simultaneous failure of multiple component services, which involves changing the business process behind a composite service. In all, we propose a two-level redundancy mechanism to achieve highly-dependable composite services.

This chapter will be organized as follows. First, we describe the background problem, the state of the art, and the general solution to the dependability issues in service composition. Then we provide a two-level redundancy framework for achieving dependable composite services. Finally, we summarize this chapter and point out further research directions.

BACKGROUND

There are many attributes associated with a web service. And the dependability can be considered as an aggregate property of availability, reliability, safety, integrity and so forth(Avizienis, Randell, & Landwehr, 2004). These attributes of services reflect from different perspectives the capability of a service. And the dependability of composite services is difficult to achieve because the components are autonomous, heterogeneous and usually come from different administrative domains.

In traditional software development theories and technologies, many software reliability models have been presented such as Jelinski-Moranda model, Littlewood-Vereall model and Nelson model (Goyeva-Popstojanova, Mathur, & Trivedi, 2001; Kai-yuan, 1995; Ramamoorthy & Bastani, 1982; Tsai et al., 2004) to solve the reliability problem. But these models do not suit service-oriented software. First, service providers and users are distributed, and the processes of service publication, search and invoking are separated. When certain components fail, a service user cannot modify them but has to choose other services with the same function. Second, components with the same function can be viewed as independent and can be used to improve the availability of composite services. Finally, the available state of a composite service may change due to the autonomy and dynamic of components. In a word, the distinct characteristics of service composition require new reliability models.

To achieve dependable composite services, many efforts have been made in terms of modeling, service selection, service composition, service replication, monitoring and adaption.

The problem of service dependability in terms of modeling technologies has been investigated (Hamadi. & Benatallah., 2003), and a Petri net-based algebra is proposed to model control flows. Discriminator operator is used to place redundancy orders to different suppliers offering the same service to increase reliability. ASDL (Abstract Service Design Language) is proposed for modeling Web Services and providing a notation for the design of service composition and interaction protocols at an abstract level (Solanki, Cau, & H.Zedan, 2006).

With dynamic service composition method (Baresi & Guinea, 2006; Mennie & Pagurek, 2000; Sun, Wang, Zhou, & Zou, 2003), only necessary functions are defined in the design period and component services are bound and instantiated at runtime. Composite services need to communicate with service registry dynamically so as to find necessary component services according to the pre-defined strategies. Service selection methods based on QoS attributes (Casati, Ilnicki, Jin, Krishnamoorthy, & Shan, 2000; Liu, Ngu, & Zeng, 2004), or based on semantic(Verma et al., 2005) are developed to improve the flexibility of composition and dynamic adaptability. But dynamically searching and several remote interactions with the service registry will affect system efficiency. And the quality of services cannot be guaranteed because the existing service registries cannot guarantee the authenticity of data and the state of registered services.

The service management research aims at improving availability by monitoring components and recovering them after failure. The Web Services architecture is extended, and high availability and autonomic behavior is gained through tracking the health of components, integrating self-monitoring, self-diagnosis and self-repair mechanisms, and so on(Birman, Renesse, & Vogels, 2004). In some other work, process description, monitoring methods and recovery strategies are proposed to achieve self-healing service composition (Guinea, 2005).

Web Service replication is also studied in this regards. An infrastructure, WS-Replication, is proposed for WAN replication of Web Services (Salas, Pérez-Sorrosal, Patiño-Martínez, & Jiménez-Peris, 2006). A middleware supporting reliable Web Services based on active replication is presented (Xinfeng Ye & Shen, 2005).

Adaptive methods such as VOC and VOCε (Harney & Doshi, 2006, 2007)are proposed to improve dependability of service. VOC method can avoid unnecessary inquiries and reduce overhead by calculating the potential value of changed information. VOCε is the improvement of VOC, which only monitors the service out of expiration time so as to gain better efficiency. They are mainly concerned with efficient and economical verification of attributes.

Based on the concept of cooperative atomic action and web service composition action, a forward error recovery method is proposed to achieve fault tolerance and dependable composite service(Tartanoglu, Issarny, Romanovsky, & Levy, 2003). However, these monitoring and recovery technologies do not consider monitoring and recovery efficiency, costs and rewards, and also they do not assess the effect of the monitoring and recovery in quantitative forms.

In contrast to existing work, we try to deal with the dependability issue of composite services through the redundancy mechanism at the two levels of component services and business processes. The two types of redundancy complement with each other to form an integrated solution to the dependability problem.

A TWO-LEVEL REDUNDANCY MECHANISM FOR DEPENDABLE COMPOSITE SERVICES

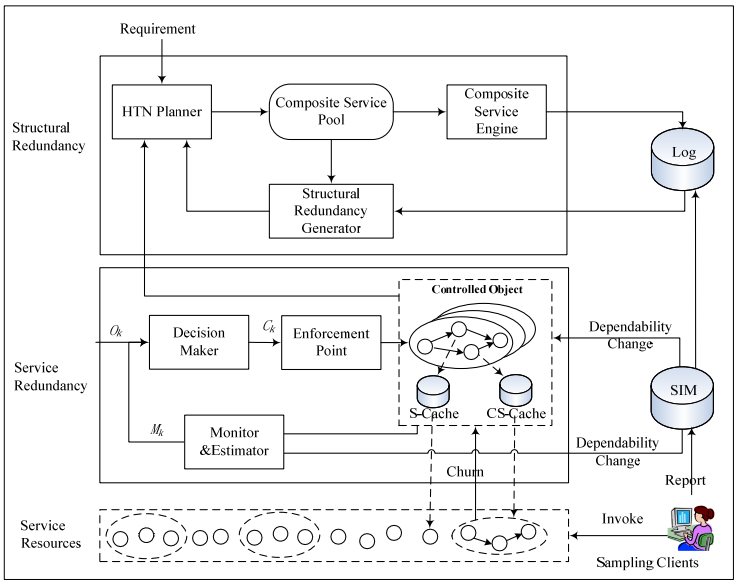
Problem Analysis

As we have mentioned, a composite service is composed of a business process structure and a set of component services. Therefore, we hope to analyze the dependability of a composite service from two angles: component service and business process structure.

First, the dependability of a component service is affected by many factors including implementation, runtime and network environment, and external attacks. The malfunction of a component service will cause the relevant composite service to be not able to provide dependable service to end users. Hence there must be a redundancy mechanism for dynamic replacement of component services when needed. In addition, the dependability of a component is not invariant throughout its lifecycle. The dynamic changing of a component service's dependability will result in the changing of the respective composite service. However, from a user's point of view, a software application should deliver dependable function consistently. Thus, it is an important issue to ensure the dependability of a composite service in dynamic environments.

Second, the business process of a composite service can also affect the dependability. In some cases, no matter how to select component services it is impossible to obtain a dependable composite service. This can be attributed to the following facts. (1) A business protocol is defined inappropriately. (2) No dependable component services are available for the current business process. This kind of problems can only be found after the software application runs for a period of time. However, after a new business process is defined, if we can incorporate redundancy for a sub process in advance, this will increase the dependability of the corresponding composite service. In this chapter, we call this mechanism structural redundancy, which means that given a sub-process, we set up a certain number of backups with equivalent functions but different structure.

Figure 1. Two-Level Redundancy Framework



A Two-Level Redundancy Framework for Dependable Composite Services

To address the dependability issue in service composition, especially the aforementioned problems, we propose a two-level redundancy framework, as shown in Figure 1. The bottom level is concerned with the dependability of component services. At this level, we propose a component service redundancy mechanism to improve the dependability of component services. Our method supports three redundancy modes including active, passive and hybrid redundancy. However, redundancy services will cause extra costs, thus we propose heuristics to determine which services should be selected as backups. Additionally, to provide consistent dependability of a composite service to users, we propose an adaptive control method based on Kalman Filter. The upper level focuses on business process structure's influence on the dependability of a composite service. At this level, we propose to use structural redundancy to further improve the dependability. In essence, the goal of structural redundancy is to add backup execution paths to a fragile segment of a composite service. In general, the two kinds of redundancy discussed in this chapter complements with each other to form a complete solution to the dependability issue.

Dependable Service Composition Based on Component Service Redundancy

Design of KAF Scheme

To obtain composite services with desirable dependability, we propose an innovative scheme called KAF that constructs a closed-loop control for adaptive maintenance of composite services(Guo, Huai, Li, & Deng, 2008). The KAF architecture is shown in the lower part of Figure 1, which consists of three main components. The Monitor&Estimator component monitors the current attributes of all services and estimates their future states based on the current and historical information. The estimated service information is needed by Decision Maker to produce corresponding maintenance strategy. The decision output is fed to Enforcement Point that implements the decision and dynamically adjusts the composition of the composite service. In addition, SIM (Service Information Manager) is an extended service registry supporting the description and update of QoS information. SIM is responsible for the reporting mechanism, collecting and managing the information of service. Some function of SIM includes service dependability value monitoring, service availability detection under different strategies (such as different frequencies, different incentive incentives, and so forth), evaluation and statistical functions of sampling client monitoring reports.

In KAF, two kinds of caching mechanisms including component services cache (or *S-Cache*) and composite services cache (or *CS-Cache*) are involved. And we call the cached services *controlled objects*. Guarantee strategy of composite services is the corresponding controller and is performed by Decision Maker and Enforcement Point. Monitor& Estimator, SIM and sampling clients together form a feedback loop. Composite services, guarantee strategy and the feedback loop constitute a closed-loop feedback control system.

In Figure 1, $O_k(0 \le k \le N)$ represents the desired values of the dependability attributes of a composite service, where N represents the number of dependability attributes. $M_k(0 \le k \le N)$ denotes the feedback values of dependability attributes from sampling clients, which is estimated by Estimator based on summarization of SIM from the report of sampling clients. $C_k(0 \le k \le N)$ are control values and are determined by the difference between O_k and M_k. C_k denotes the dependability value to be adjusted. Based on adjustment strategies, the Enforcement Point chooses candidate services and completes the reconstruction of services composition.

In KAF architecture, the feedback loop is composed of two mechanisms: execution monitoring and reporting. Execution monitoring obtains the service state information from the execution engine through monitoring. Reporting mechanism summarizes the real service information from sampling clients. And we assume all the sampling clients are honest client nodes selected by SIM. Estimator estimates dependability of component service base on the results of SIM and then computes dependability of composite service.

To adjust the dependability effectively, the Decision Maker determines service selecting and update strategy, such as increasing number of backup service, replacing undesirable services. In this chapter, guarantee strategy is a kind of meta-strategies. According to a meta-strategy, Enforcement Point runs the APB algorithm to select new services or to replace declining or failed services to implement the composite service's re-construction. In addition to the dependability of services, we must also consider the cost of replacement and rewards of expectation, so that to maximize the long-term revenue. To achieve this goal, we propose to use Markov decision process theory to support strategy choices.

Adaptive Control Mechanism Based on MDP and Kafman Filter

Due to the dynamic nature of component services, such as load, network conditions and churn, the dependability of composite service will be affected. Thus we need to select some redundancy services to ensure the composite service can still deliver the desired dependability in the cases of degraded component service dependability.

We design dependability factor $f = lnO/lnD$, where O is the desired value for dependable properties need to satisfied and D means the design value of dependability. Dependability factor reflecting the important degree of dependability is used to guide the choice of component service. Generally, f is larger than 1, which means D usually is larger than O. We can see that, when the dependability of component services declines rarely, low dependability factor can reduce the cost of constructing composite services. When dependability drops significantly, increasing of dependability factor can improve the quality of composite service, and the cost may increase.

To adjust the dependability factor of composite service reasonably, we select MDP (Markov Decision Process)(Puterman, 1994) to modeling the composite service maintenance process: RSC = $\langle S, A, \{A(x) \mid x \in S\}, Q, RV \rangle$, where S is the set of all possible states and A is the set of decision strategies, $\{A(x)|x \in S\}$ is the strategy when the state is x, $Q(B|x, a) = Q\{X_{t+1} \in B| X_t=x, A_t=a\}$ specifies the probability to the next state X_{t+1} when the state is $X_t=x$ and after strategy $A_t=a$ is executed, and RV is the reward of executing the strategy. If we assume that the state of system in moment t is $X_t=x \in S$, while the decision is $A_t=a \in A(x)$, under the decision of transfer function Q, system transfers to the state X_{t+1} with the reward of $RV(a,x)$. After the transfer, the system enters a new state, and then the system makes a new decision to continue decision-making process. Therefore we establish Markov decision process model of component service dependability's maintenance as follows.

- Dependability of a composite service *CS* is represented as $d = \alpha a + \beta r + (1-\alpha-\beta)t$, in which *a*, *r* and *t* are the availability, the reliability and the trust of a composite service in a certain period respectively. Note that α, β and (1−α−β) are the weights of availability, reliability and trust. The assignment of weights can be determined by the users to reflect different emphasis on these attributes. The dependability value range [0,1] is divided into *k* parts. When $d \in [\frac{i-1}{k}, \frac{i}{k})$, we define that the state of the composite service is s_i, marked as $s_i=[\frac{i-1}{k}, \frac{i}{k})$. And the state set of composite service dependability is denoted by $S=\{s_1, ..., s_m\}$. By dividing [0, 1], we can reduce the state space and reduce the complexity of decision-making in service composition.
- Let Δf be the changing amount of dependability factor, We define the adjustment strategy set as $A=\{-\Delta f, 0, +\Delta f\}$, where the strategy $-\Delta f$ and $+\Delta f$ mean the amount of reduction or addition values of dependability factor respectively, and the strategy 0 means no changing to the dependability factor.
- For a composite service *CS*, suppose that *d*, ψ, *w* and η represent the dependability, revenue, service adjustment cost and the importance degree of the *CS*, where η is a positive real number, we denote by $rv = \eta(-\ln(1-d)) + \psi - w$ the reward of composite service *CS*. This represents if composite service properties such as availability, reliability and trust are higher or the maintenance cost of a composite service is smaller, we can get more reward. Then composite service developers will be more satisfied with the composite service, i.e. the value of *v* will be larger. Among them, the income is related to the number of service execution and the prices of services.
- For a composite service CS, at the moment *t* suppose that CS is in the state s_i, we denote by $rv = \eta(-\ln(1-d)) + \psi - w(A_t, s_i)$ the immediate reward of composite service CS after adopting strategy, where $w(A_t, s_i)$ represents the cost of dependability adjustment.

With an MDP problem, if the state transferred probability function and reward function are known, we can compute optimal decision strategies by using dynamic programming methods. However, in the service-oriented software development process, it is difficult to observe all the historical actions of component services. That means that transition probability function and reward function are unknown, therefore we can not use dynamic programming techniques to determine optimal decision strategy. Instead we apply Kalman-Filter based approach to estimate service states, then adjust controller following the estimated value.

Kalman filter(Haykin, 2002) uses the recurrence of the state equation to achieve the optimal estimate of state variables in the linear dynamic system. The Kalman filter is unbiased and has the smallest variance. It is easy to realize with computer programs and is suitable for on-line analysis. Furthermore, the extended Kalman filter (EKF) can be used in nonlinear systems. Therefore, we introduces the extended Kalman filter to estimate the dependability of component services so as to implement the adaptive control maintenance of a composite service's dependability (Guo, Huai, Li, & Deng, 2008).According to the aforementioned analysis, we design the adaptive control algorithm for the dependability maintenance of the composite service. The basic idea is as follows. First we collect and summarize the sampling client's usage data about service execution. Second, we calculate the dependability of composite services and compare with the expected value. Third, we estimate the dependability value of each selected component service by the Kalman filter formula, compute the immediate reward of every action and choose corresponding action following the MDP framework. Finally we execute the service selection algorithm and the strategy executing module to select new services and replace some degenerate service.

KAF adaptive control algorithm

input: constructed composite service, available component service properties measurement value, expected dependable value O_k of composite service.

output: dependability factor, selected services

1. Read the O_k, determine the weight of every attributes;
2. Collect the initial dependable attribute values M_k from SIM;
3. Calculate the dependable attribute values for each component service respectively;
4. Calculate d^*_k of composite service;
5. Collect sampling measure values M_k of component services;
6. Predict the value of the next period by Kalman filter following the measure values;
7. Compute the immediate reward;
8. Determine the action and modify the dependable factor *f*;
9. Calculate updated d_k ;
10. Calculate $\Delta d_k = d_k - d^*_k$; // Δd_k is the C_k
11. Select new service following the HAF algorithm;
12. $d^*_k = d_k$;

goto step 5.

The input of adaptive control algorithm includes constructed composite service, available component services' properties measure value M_k, expected dependable value O_k of composite service. The output includes dependability factor and selected services.

The first part of KAF algorithm (lines 1-4) is initialization, including reading the expectation value O_k of composite services, collecting parameters of component services from SIM, calculating the dependable attribute values and the real dependability of the composite service. Then obtain sampling measure values of component services, and according to Kalman filter calculation formulas estimate the values in the next period. Afterwards in the MDP, compute the immediate reward and determine the action of modifying the dependable factor α (line 5-8).

In the third part (lines 9-13), compute the dependable attribute verity values Δd_k and select new service following the HAF algorithm.(Guo et al., 2007). At last, to measure the component services and start another prediction and control process.

Service Redundancy and Selection

Our KAF scheme relies on certain service redundancy and selection mechanisms to enforce the adaptive dependability adjustment. For example, the HAF algorithm used in KAF algorithm is responsible for selecting appropriate backup services to obtain the desirable dependability. In this section, we are concerned with service redundancy and selection issues. Especially, we take availability as an example of dependability attributes. Replication technique is a well known technique to improve service availability. Generally, there are three different redundancy mechanisms: active redundancy, passive redundancy and hybrid redundancy (Barrett et al., 1990; N.Budhiraja, Marzullo, & Schneider, 1992; Schneider, 1990).

Service Redundancy (SR). Service redundancy, SR is denoted as a tuple <AR, BR>, |AR|>0, |BR|≥0，where *AR* and *BR* are the service subsets that provide the same or similar functions. *AR* is the primary service set and *BR* is the backup service set. Three service redundancy approaches are determined by the number of services in *AR* and *BR*.(Guo et al., 2007).

(1) **Active redundancy.** When |AR|>1, |BR|=0, execution of Service redundancy SR means executing all services in AR; (2) **Passive redundancy.** When |AR|=1, |BR|>0, execution of Service redundancy SR means only executing the primary service in AR, and only if the primary service is failed, one backup service in BR is selected to execute as a primary service;(3) **Hybrid redundancy.** When |AR|>1, |BR|>0, execution of Service redundancy SR means executing all services in the AR, when one or more primary services fails, some backup services in BR are selected to execute as the primary ones.

In practice, with the fixed size of service subsets, larger |AR| and smaller |BR| mean more costs; in contrast, smaller |AR| and larger |BR| can causes worse user experience. Therefore, it is a tradeoff to decide an appropriate service redundancy approach, and we will not discuss this issue in this chapter.

Service availability. Service availability represents the probability that a service is up in a specified period of time under specific conditions.

Service availability can be measured as:

$$A_{s_j^i} = \frac{time_s}{time_a} = \frac{time_s}{time_s + time_u} \quad \text{(Ran, 2003)} \qquad (1)$$

Where services s_j^i means the ith service selected to complete task t_j, $A_{s_j^i}$ denotes the availability of service s_j^i, $time_s$ is the available time of s_j^i, $time_u$ is the unavailable time of s_j^i, $time_a$ represents the total time of s_j^i being measured and $time_a = time_s + time_u$.

The availability of composite based on the redundancy mechanism is influenced by the availability of switch, message replicating and consensus voting component. To simplify the calculation we assume that these components are always available, that means the availability of them is 1. And we get $A_{AR} = A_{PR} = A_{HR} = (1 - \prod_{i=1}^{n}(1 - A_{s_j^i}))$. Where A_{AR}, A_{PR}, A_{HR} represent the availability of a task using the three redundancy approaches respectively.

The availability of composites is gotten based on the availability of tasks and the service composition modes. In the process of service composition, four basic composition modes: sequence (•), parallel (||), choice (+) and iteration (◦) are used. All processes can be modeled using the four modes in principle (W. v. d. Aalst & Hee, 2002). To compute the availability of composite, we get the following formulae based on (Cardoso, 2002).

Table 1. Formulae of composite service availability

$$A_{Sequence} = \prod_{j=1}^{m}(1 - \prod_{i=1}^{n_j}(1 - A_{s_j^i})); \; i = 1,2,...,n_j; \; j = 1,2,...,m \qquad (2)$$

$$A_{Parallel} = \prod_{j=1}^{m}(1 - \prod_{i=1}^{n_j}(1 - A_{s_j^i})); \quad i = 1,2,...,n_j; \; j = 1,2,...,m \qquad (3)$$

$$A_{Choice} = \sum_{j=1}^{m}(1 - \prod_{i=1}^{n_j}(1 - A_{s_j^i})) * p_j; \; \sum_{j=1}^{m} p_j = 1; \; \begin{matrix} i = 1,2,...,n_j; \\ j = 1,2,...,m \end{matrix} \qquad (4)$$

$$A_{Iteration} = (1 - p) * (1 - \prod_{i=1}^{n_j}(1 - A_{s_j^i})) / (1 - p * (1 - \prod_{i=1}^{n_j}(1 - A_{s_j^i}))) \qquad (5)$$

Assume that a process *Proc* includes m tasks t_1, …, t_m, and every task can be completed by several services with the same function. Suppose that after some redundancy services have been selected, the availability of task t_j are $A(t_j)$, and we assume $A(t_1)=\min\{A(t_j)| j=1,\ldots, m\}$. If candidate services s_j^i for every task t_j exist (j=1,…, m), and the availability of every service s_j^i is the same. Let $A(proc)_{t_j}^{s_j^i}$ denote the availability of task t_j after adding service s_j^i, we have the following theorems.

Theorem 1. If tasks t_1, …, t_m in *Proc* only satisfy parallel and sequence modes, then $A(proc)_{t_1}^{s_1^i} = \max\{A(proc)_{t_j}^{s_j^i} | j=1,\ldots, m\}$.

Proof 1. Because $A(proc)_{t_j}^{s_j^i} = A(t_1)\cdot A(t_2)\cdot\ldots\cdot A(t_{j-1})\cdot(1-(1-A(t_j))(1-A_{s_j^i}))\cdot A(t_{j+1})\cdot\ldots\cdot A(t_m), j=1, 2,...,m.$ for any $1<j\le m$, $A(proc)_{t_1}^{s_1^i} - A(proc)_{t_j}^{s_j^i} = ((1-(1-A(t)_1)(1-A_{s_1^i}))\cdot A(t_j) - A(t_1)\cdot(1-(1-A(t_j))(1-A_{s_j^i})))\cdot A(t_2)\cdot\ldots\cdot A(t_{j-1})\cdot A(t_{j+1})\cdot\ldots\cdot A(t_m)$. From the assumption $A_{s_1^i} = A_{s_j^i}$, $A(t_1)=\min\{A(t_j)| j=1,\ldots, m\}$, we

have $(1-(1-A(t_1))(1-A_{s_1^i}))\cdot A(t_j)-A(t_1)\cdot(1-(1-A(t_j))(1-A_{s_j^i}))=(A(t_j)-A(t_1))A_{s_j^i}\geq 0$. So for any $1<j\leq m$, we have $A(proc)_{t_1}^{s_1^i}\geq A(proc)_{t_j}^{s_j^i}$, that is $A(proc)_{t_1}^{s_1^i}=\max\{A(proc)_{t_j}^{s_j^i}|j=1,\ldots,m\}$. □

Theorem 2. If tasks $t_1,\ldots,t_m$ in *Proc* only satisfy the choice mode and the probability of every task execution is the same, then $A(proc)_{t_1}^{s_1^i}=\max\{A(proc)_{t_j}^{s_j^i}|j=1,\ldots,m\}$.

Proof of theorem 2 is similar to the proof of theorem 1, and is omitted.

Using redundancy services can improve the availability of composites. However, the cost of constructing composites will also increase. To improve the availability of overall system, both quality and cost should be taken into consideration at the same time.

The problem of redundancy services chosen can be modeled as a nonlinear mixed integer programming problem. In the travel agent example, we use the services redundancy approach and get the following objective function.

Objective: maximize

$$\{\prod_{j=1}^{5}(1-\prod_{i=1}^{n_j}(1-A_{s_j^i})^{y_j^i})\} \tag{6}$$

where

$$y_j^i=\begin{cases}1, service\ \ s_j^i\ \ is\ \ selected\\ 0, service\ \ s_j^i\ \ is\ \ not\ \ selected\end{cases}\quad i=1,2,...,n_j;\ j=1,2,...,5 \tag{7}$$

The objective will be different according to the actual process of composite service.

Cost Constraint:

$$C_{sum}=(\sum_{j=1}^{5}\sum_{i=1}^{n_j}y_j^i c_{s_j^i})\leq C_{const}\quad i=1,2,...,n_j;\ j=1,2,...,5 \tag{8}$$

Where $c_{s_j^i}$ represents the cost of invoking service s_j^i, C_{sum} is the cost of all selected services and C_{const} is the constraint value of the cost. In this chapter we do not concern the cost of testing and maintenance.

The Integer programming is a NP hard problem. In a service composition process, if there are *m* tasks and *n* physical services to finish every task, the compute complexity in the worst cases is $O(2^{mn})$. If the number of tasks and candidate services are small, we can use the Exhaustive Search Algorithm to find the optimal result. But in fact with the rapid acceptance of Web Service technology, the number of services with the same function will become enormous. At the same time, complicated services that need more partners to work together will become more and more popular. Therefore more efficient algorithm is needed.

HAF algorithm:

1: arrange s^i_j ordered by $A_{s_j^i}$ sorted by t_j;
2: select s^i_j with the maximum $A_{s_j^i}$ // modify y^i_j
3: calculate every $A(t_j)$, A_{sc}, C_{sum},
4: arrange the t_j ordered by $A(t_j)$
5: **for** the task t_j with lowest $A(t_j)$ **do**
6: **for** all unselected s^i_j to complete task t_j **do**
7: select s^i_j with largest $A_{s_j^i}$
8: **if** constraint is violated **then**
9: the last selected service is excluded for further concern
10: run Step 7
11: **end for**

```
12:    if no service available for task t_j then
13:          task t_j is excluded for further concern
14:    run step 3
15: if constraint is precisely satisfied then
16:          run Step 21
17: else
18:          run Step 3
19: end for
20: if all tasks has been excluded then
21:    compute A_SC , output A_SC and y^i_j .
```

To select redundancy service, we design the heuristic algorithms based on Theorem 1. The basic idea is as follows. When choosing multiple physical services for tasks, we select a service for the task that has the lowest availability and choose the service with the best availability. The algorithm is called HAF (High Availability First) algorithm.

In HAF algorithm, the input information includes availability and price of candidate services, the tasks and the relationships among these tasks. The output includes selected services and availability value of the process. The term $A(t_j)$ denotes the availability of task t_j.

The first part of HAF algorithm (lines 1-2) arranges the candidate components ordered by availability sorted by tasks, and services with the best availability are selected. Then for every task, one service is selected. Next, calculate availability of every task, availability of the process and the total cost of selected services. And arrange the tasks ordered by availability value (line 3-4).

In the third part (lines 5-7), for the task with lowest availability, one more candidate service with highest availability is selected. In the fourth part (lines 8-19), the algorithm checks if the constraint function is violated or not. If it is violated ($C_{sum} > C_{const}$), then exclude the last selected service (line 9) and choose another service with best availability in the left services set for this task (line 10). If constraint function is precisely satisfied ($C_{sum} = C_{const}$), calculate the availability of the process (line 16). If constraint function is not violated ($C_{sum} < C_{const}$), then calculate the availability of tasks and repeat the execution of the third part of the algorithm (line 18). In the last part (lines 20-21), compute the value of composite and output the selected services.

Structural Redundancy

Structural redundancy deals with composite service dependability at business process level. After a business process is implemented by a composite service, the runtime data can be collected and analyzed to determine whether the business process is satisfied or not. In terms of dependability, if a composite service cannot deliver dependable services no matter how to select component services, the problem may probably lies in the business process itself. Thus we need an approach to evolve a business process to make it meet dependability requirements. In this section, we are mainly concerned with the *availability* attribute. A method called AvailEvo(Zeng, Sun, Liu, Deng, & Huai, 2010) is introduced to implement structural redundancy. Since structural redundancy is based on the changing of a business process, how to preserve the process correctness becomes an important issue. We start from the soundness of a business process.

Business Process Soundness

For process-aware composite service, the most important correctness criterion is structural soundness (or *soundness* for short, see definition Soundness) (W. v. d. Aalst & Hee, 2002; Rinderle, Reichert, & Dadam, 2004). For a complex business process it may be intractable to verify its soundness. Especially under the online and time-critical scenarios, there is not enough time to do the verification. In a word, it is highly

important and challenging to find an effective method that can satisfy structural redundancy requirements while preserve soundness and functions of original composite services.

In AvailEvo, we adopt WF-net to formally describe a composite service. WF-net is a special Petri net and possesses the advantages of formal semantics definition, graphical nature and analysis techniques.

WF-net(W. v. d. Aalst & Hee, 2002)*: A Petri net WFN=(P,T,F) is a WF-net (Workflow net) if and only if:*

- *There is one source place $i \in P$ such that $\bullet i = \Phi$;*
- *There is one sink place $o \in P$ such that $o\bullet = \Phi$; and*
- *Every node $x \in P \cup T$ is on a path from i to o;*

It should be noted that a WF-net specifies the dynamic behavior of a composite service. In WF-net, tasks (component services) are represented by transitions, conditions are represented by places, and flow relationships are used to specify the partial ordering of tasks. In addition, the *state* of a WF-net is indicated by the distribution of tokens amongst its places. We use is a $|P|$ dimension vector M to present the state of a live procedure in WF-net, where every element means the number of tokens in a place. We denote the number of tokens in place p in state M by M_p. Specifically, we use M_0 (M_{end}) to denote initial state (final state) which has only token in source (sink) place. A transition $t \in T$ is enabled in state M if and only if $M_p>0$ for any place p *such that* $(p,t) \in F$. If t is enabled, t can fire leading to a new state M' such that $M'_p=M_p-1$ and $M'_{p'}= M_{p'}+1$ for each $(p,t) \in F$ and $(t,p') \in F$, which is denoted by $M\,[t>M'$.

Soundness (W. v. d. Aalst & Hee, 2002)*: A procedure modeled by a WF-net WFN=(P, T, F, i, o, M_0) is sound if and only if:*

- *For every state M reachable from initial state M_0, there exists a firing sequence leading from state M to final state M_{end}. Formally:*

$$\forall M (M_0 \xrightarrow{*} M) \Rightarrow (M \xrightarrow{*} M_{end})$$

- *State M_{end} is the only state reachable from state M_0 with at least one token in place o. Formally:*

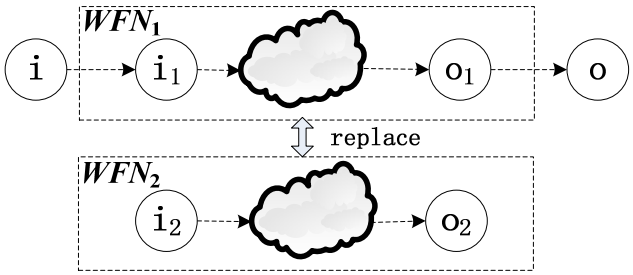
$$\forall M (M_0 \xrightarrow{*} M \wedge M \geq M_{end}) \Rightarrow (M = M_{end})$$

- *There are no dead transitions in WFN. Formally:*

$$\forall t \in T \; \exists M,\, M' \text{ s.t. } M_0 \xrightarrow{*} M \xrightarrow{t} M'$$

Soundness is an important correctness criterion that guarantees proper termination of composite service. The first requirement of the definition Soundness means that starting from the initial state M_0, it is always possible to reach the state only with one token in place o. The second requirement states that when a token is put in place o, all the other places should be empty. The last requirement states that there are no dead transitions in the initial state M_0. Generally speaking, for a complex WF-net it may be intractable to decide soundness(Cheng, Esparza, & Palsberg, 1993). In this chapter we do not verify soundness of an evolved composite service while we focus on preserve soundness using a basic change operation set. In addition, all of the discussed composite services are modeled by WF-net are acyclic structures and a component service (transition) appears only once in a composite service (WFN). We do not consider data flow and data constraints issues in this work.

Figure 2. Replacement



A set of change operations preserving soundness is defined including replacement, addition, deletion and process structural adjustments(Zeng, Huai, Sun, Deng, & Li, 2009). We illustrate that a sound WF-net evolved with the set can preserve soundness all the same. *Replacement* operation means replacing a sound sub WF-net in a sound WF-net by another sound WF-net.

Replacement Operation. *Let $WFN_1=(P_1,T_1,F_1, i_1,o_1,M_{01})$ be a sound sub net of a sound WF-net $WFN=(P,T,F,i,o,M_0)$ and $WFN_2=(P_2,T_2,F_2,i_2,o_2, M_{02})$ be a sound WF-net such that $P_1\cap P_2=\Phi$, $T_1\cap T_2=\Phi$, $F_1\cap F_2=\Phi$, then $WFN'=(P', T', F', i', o', M_0')$ is the WF-net obtained by replacing WFN_1 by WFN_2, such that $P'=(P\backslash P_1) \cup P_2$, $T'=(T\backslash T_1) \cup T_2$, $F'=(F\backslash F_1) \cup F_2 \cup F''$, where $F''=\{(x,i_2) \in P\times T_2 | (x, i_1) \in F_1 \} \cup \{(o_2, y) \in T_2\times P \mid (o_1, y) \in F_1 \}$, and initial state M_0' is a $|P'|$ dimension vector, where $S\backslash S'$ denotes the difference between set S and S', i.e. the set of elements which is in S and not in S'. (Figure 2)*

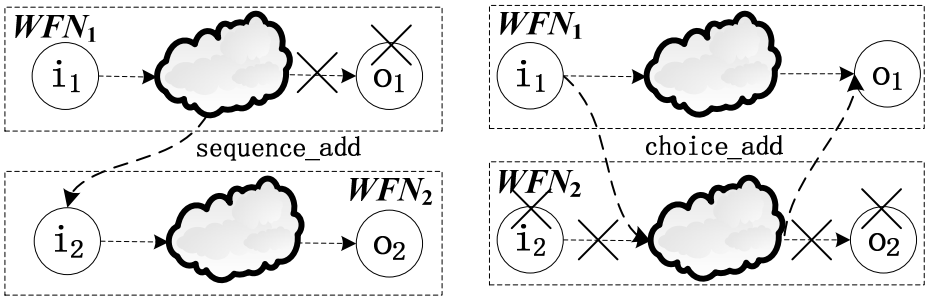
We know if a transition in a sound WF-net is replaced by a other sound WF-net, then the resulting WF-net is also sound (Theorem 3 in literature (W. M. P. v. d. Aalst, 2000)). Our replacement operation is an extension to the result mentioned above, because a sound WF-net behaves like a transition. Obviously, we have the conclusion:

Proposition 1. Replacement operation can preserve soundness of a sound WF-net.

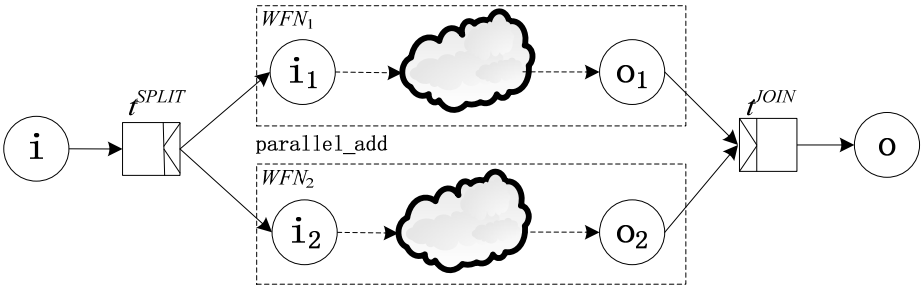
Addition Operations. *Let $WFN_1=(P_1,T_1,F_1, i_1, o_1, M_{01})$ and $WFN_2=(P_2,T_2,F_2,i_2,o_2,M_{02})$ be two sound WF-nets, such that $P_1\cap P_2=\Phi$, $T_1\cap T_2=\Phi$, $F_1\cap F_2=\Phi$.*

- ***Sequence_add ($WFN_1 \rightarrow WFN_2$)****: WFN=(P, T, F, i_1, o_2, M_0) is the WF-net obtained by sequence_adding WFN_1 with WFN_2, where $P=(P_1\backslash\{o_1\}) \cup P_2$, $T=T_1 \cup T_2$, $F=\{(x,y) \in F_1 | y\neq o_1\} \cup \{(x, i_2) \in T_2\times P_1 | (x, o_1) \in F_1\} \cup F_2$ and M_0 is a $|P|$ dimension vector (where first element is 1 and rest ones are 0). (Figure 3a)*
- ***Choice_add (WFN_1+WFN_2)****: WFN=(P, T, F, i_1, o_1, M_0) is the WF-net obtained by choice_adding WFN_1 with WFN_2, where $P=P_1 \cup (P_2\backslash\{i_2, o_2\})$, $T=T_1 \cup T_2$, $F=F_1 \cup \{(x, y) \in F_2 | x\neq i_2 \wedge y\neq o_2 \} \cup \{(i_1, y) \in P_1\times T_2 | (i_2, y) \in F_2\} \cup \{(x, o_1) \in T_2\times P_1 | (x, o_2) \in F_2\}$ and M_0 is a $|P|$ dimension vector (where first element is 1 and rest ones are 0). (Figure 3b)*
- ***Parallel_add (WFN_1||WFN_2)****: WFN=(P, T, F, i, o, M_0) is the WF-net obtained by parallel_adding WFN_1 with WFN_2, where $P= P_1 \cup P_2 \cup \{i, o\}$, $T= T_1 \cup T_2 \cup \{t^{SPLIT}, t^{JOIN}\}$, $F=F_1 \cup F_2 \cup \{< i , t^{SPLIT} >, < t^{SPLIT} , i_1>, < t^{SPLIT} , i_2>, < o_1, t^{JOIN} >, <o_2, t^{JOIN} >, < t^{JOIN} , o>\}$ and M_0 is a $|P|$ dimension vector (where first element is 1 and rest ones are 0). (Figure 3c)*

Figure 3. Addition



a. Sequence_add **b.** Choice_add



c. Parallel_add

Addition operations show the method of combining two sound WF-nets together. Proposed *Addition* operations are sequential, parallel and choice adding a sound WF-net to another sound WF-net, obviously so obtained new WF-net is sound. Hence we have the conclusion:

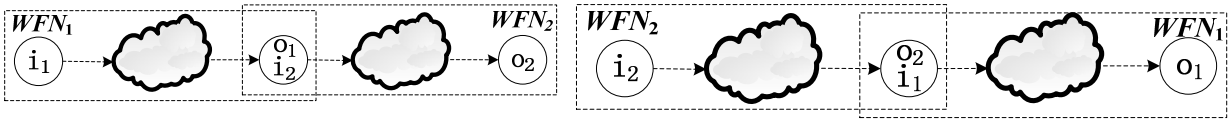
Proposition 2. Addition operations can preserve soundness of a sound WF-net.

Delete operations are the reverse of addition operations; we do not discuss a series of *delete* operations due to the limitation of chapter space. To process evolution, besides above *replacement*, *addition* and *deletion* operations, sometimes we need deal with process structure adjustments. In this chapter, five basic *structural adjustment* operations are presented to allow modifying some sound sub nets in a sound WF-net.

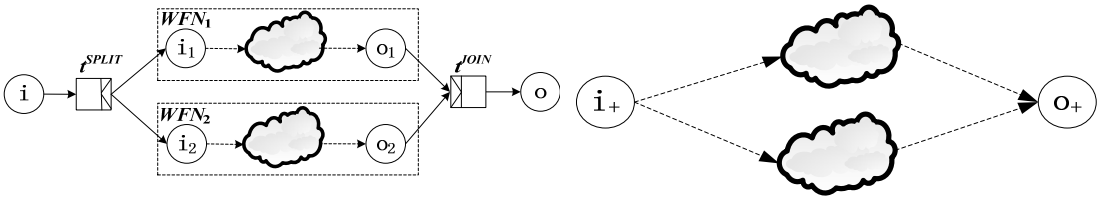
Sequence Structure Adjustments. *Let $WFN= (P, T, F, i, o, M_0)$ be a sound WF-net consisting of $WFN_1=(P_1,T_1,F_1,i_1,o_1, M_{01})$ and $WFN_2=(P_2,T_2,F_2,i_2,o_2,M_{02})$ in sequence, such that $P=P_1 \cup P_2$, $T=T_1 \cup T_2$, $F=F_1 \cup F_2$, $i=i_1$, $o_1=i_2$, $o=o_2$ (Figure 4a).*

- ***Reversal_sequence adjustment****: $WFN' =(P, T , F, i', o' , M_0)$ is the WF-net obtained by reversal sequence adjustment between WFN_1 and WFN_2, where $i'=i_2$, $o'=o_1$, $o_2=i_1$; (Figure 4b)*
- ***Sequence-to-parallel adjustment****: $WFN_{||}=(P_{||}, T_{||}, F_{||}, i_{||},o_{||},M_{0||})$ is the WF-net obtained by sequence_to_parallel adjustment between WFN_1 and WFN_2, where $P_{||}= P \cup\{i_{||}, o_{||}\}$, $T_{||}=T \cup\{ t^{SPLIT}, t^{JOIN} \}$, $F_{||}=F \cup\{< i_{||}, t^{SPLIT} >, < t^{SPLIT} , i_1>, < t^{SPLIT} , i_2>, <o_1, t^{JOIN} >, <o_2, t^{JOIN} >, < t^{JOIN}, o_{||}>\}$ and $M_{0||}$ is a $|P_{||}|$ dimension vector (where its first element is 1 and rest ones are 0). (Figure 4c)*
- ***Sequence_to_choice adjustment****: $WFN_+=(P_+,T_+,F_+ i_+, o_+,M_{0+})$ is the WF-net obtained by sequence_to_choice adjustment between WFN_1 and WFN_2, where $P_+=(P\setminus\{i_x, i_y, o_x, o_y\})\cup\{i_+, o_+\}$, $T_+=T$, $F_+=\{(p, q) \in F|p\neq i_1, i_2 \wedge q\neq o_1, o_2\} \cup\{(i_+, q)|(i_1, q) \in F_1 \vee (i_2, q) \in F_1 \} \cup\{(p, o_+)|(p, o_1) \in F_1 \vee (p, o_2) \in F_2\}$ and M_{0+} is a $|P_+|$ dimension vector (where its first element is 1 and rest ones are 0). (Figure 4d)*

Figure 4. Sequence Structure Adjustment



a. Original sequence structure of two WF-net **b.** Reversal sequence adjustment



c. Sequence_to_parallel adjustment **d.** Sequence_to_choice adjustment

Structure Adjustment operations indicate the reassembly method of two sound sub WF-nets in a sound WF-net. Similarly the "*parallel_to_sequence adjustment*" and "*choice_to_sequence adjustment*" could be defined. We omit them due to the limitation of chapter space. Because proposed *structural adjustments* act on two sound sequential sub WF-net, clearly obtained new WF-net is sound. Hence we have the conclusion:

Proposition 3. Structural adjustment operations can preserve soundness of a sound WF-net.

AvailEvo: A Structural Redundancy Method

The AvailEvo method is an extension and optimization of conventional dependability guarantee methods based on planning in service composition. The main procedure is as follows. First, by analyzing execution logs of component services, the sets of component services with low availability and the execution paths with highest frequency are found out. Thus, a basic execution sequence can be acquired. Certain component services in the basic execution sequence affect whole availability of composite services seriously. Second, by analyzing the structural relationships between component services, a redundant path is obtained after expanding the basic execution sequence, and then the original composite service are evolved, that is adding the redundant path into the composite service process structure and obtaining a new process structure. Finally, classical planning methods (e.g. HTN planning) are used to select newly component services and to produce a new and executable composite service with high dependability. For introducing our AvailEvo in detail, definitions of Transition Path and Transition Relationships are given.

Transition Path. *WFN=(P, T, F, i, o, M_0) is a sound WF-net, a transition sequence (firing sequence) from initial state M_0 to final state M_{end} is named transition path (denoted by tp), and ep $\in T^*$; in addition, a set containing all of the execution paths is called Log over the WF-net (denoted by L).*

Transition Relationships. *Let WFN= (P, T, F, i, o, M_0) be a sound WF-net and L be the Log over WFN, where a and b are transitions of WFN, i.e. a, b $\in T$:*

- *a* is a predecessor of *b* on *tp* (or *b* is a successor of *a* on *tp*): there is a $tp=t_1t_2\ldots\ldots t_n \in L$, such that $t_i=a$, $t_j=b$, $j \geq i+1$, denoted $a >_{tp} b$;
- *a* is a predecessor of *b* on *L* (or *b* is a successor of *a* on *L*): there is each $tp \in L$, if $a \in tp$ and $b \in tp$, then $a >_{tp} b$, denoted $a \rightarrow_L b$;
- *a* and *b* are parallel on *L*: there is *tp*, $tp' \in L$, such that $a >_{tp} b$ and $b >_{tp'} a$, denoted $a ||_L b$;
- *a* and *b* are choice on *L*: there is each $tp \in L$, if $a \in tp$ and $b \in tp$, then $a \not\succ_{tp} b$ and $b \not\succ_{tp} a$, denoted $a \#_L b$.

AvailEvo algorithm provides an implementation for the composite service method with availability guarantee. The input of the algorithm consists of a sound WF-net WFN^O describing the business process structure of a composite service, a transition path *tp* with highest execution frequency gained by analyzing execution log and a component service set *S* not achieving expected availability target. The output of the algorithm is a new evolved composite service WFN^N with redundancy paths. First, a basic transition sequence *ts* is easily obtained, which is generated by a intersection of transition sets contained in *tp* and set

S. However, besides transitions in *S*, a whole transition sequence *ts* need to contain necessary transitions in *tp*(see line 1,2). Here, some special situations with empty *ts* or only one transition in *ts* are not considered, i.e. $|ts|\geq 2$. Thus, that availability of a part of components in *ts* has not reached expected target results in low availability of whole composite service. As the sub WF-net composed of transitions in *ts* is not always sound, the evolution as redundancy path with *ts* will destroy execution semantic of original composite service. Therefore, the emphasis of the proposed algorithm is that a minimum transition sequence *ts'* is found out on the basis of *ts*, and in WF-net *WFN,* transitions contained in *ts'* compose a sound sub WF-net *WFN'*. In fact, the process of finding *ts'* is the process of extending *ts*. By comparing the relations between two near transition, *ts* is extended. For it is a directed sequence, *ts* must first be extended in the forward direction. Transitions(Set_1) in *ts* have parallel relation with t_i and transitions(Set_2) in *ts* have parallel relation with t_{i+1} are found. When transition sets Set_1 and Set_2 are not equal, it is regarded that t_i and t_{i+1} belong to two different parallel structures separately, so all transitions in two parallel structures need to be added into *ts* (here start transition t^{SPLIT} and join transition t^{JOIN} need to be added). Transitions(*Set3*) in *ts* have choice relation with t_i and transitions(*Set4*) in *ts* have choice relation with t_{i+1} are computed. When transition sets *Set3* and *Set4* are not equal, it is regarded that t_i and t_{i+1} belong to different choice structures separately, so these transitions in same choice structures need to be added into *ts*(see line 3 to 14). In a similar way, backward extension is carried out(see line 15 to 23). Finally, transition sequence $ts_{FORWARD}$ obtained by forward extension and transition sequence $ts_{backWARD}$ obtained by forward extension are merged to gain *ts'*. At the same time, a sound sub WF-net *WFN'* composed of transitions in *ts'* is found out in WFN^O. then *WFN'* is treated with for redundancy by using *ts'* (see line 24 to 27) using operation *Choice_add* in change operations preserving soundness.

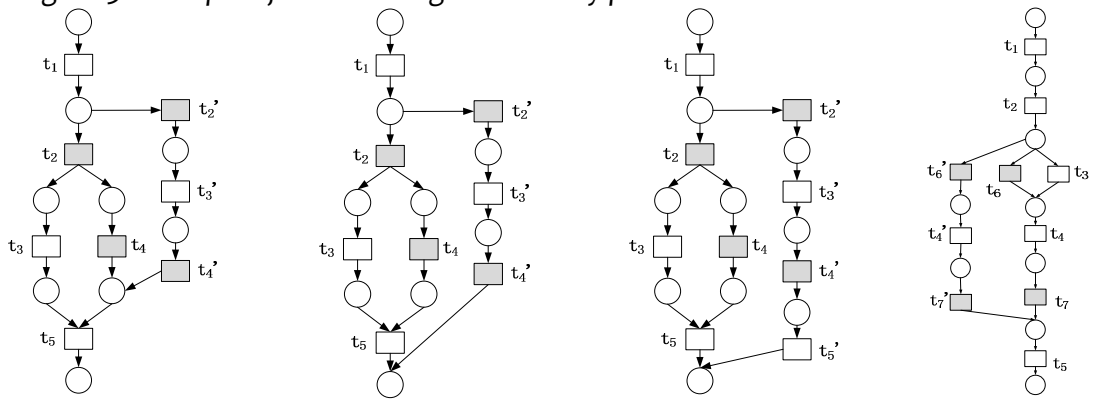
The key of AvailEvo algorithm is finding redundancy paths according to the different relations between transitions. Preconditions of determining the relation between transitions must be the known the log the WF-net. It is difficult to compute a *Log* of a sound WF-net directly, so we convert it into a *reachability graph*(Xinming Ye, Zhou, & Song, 2003). Then the *Log* can be found out by means of Breadth-First-Search based on the *reachability graph.* Here, a necessary explanation is that the size of *reachability graph* of WF-net is exponential on the size of WF-net in the worst-case time, so the worst-case complexity of AvailEvo Algorithm is exponential. In addition, the main part of AvailEvo algorithm is forward and backward extension operations for basic transition sequence *ts,* so the algorithm is able to terminate obviously. In course of working, the algorithm virtually includes sequence, parallel and choice structures contained in *ts.* According to literature(W. v. d. Aalst & Hee, 2002), sub WF-net *WFN'* corresponding with redundancy path *ts'* must be sound.

AvailEvo Algorithm
input: $WFN^O=(P^O, T^O, F^O, i^O, o^O, M_0{}^O)$, *tp, S*
output: $WFN^N=(P^N, T^N, F^N, i^N, o^N, M_0{}^N)$

```
1   begin
2   ts=SubPath(S,tp)
3   k=|ts|
4   ts_FORWARD=ts
5   ts_BACKWARD=ts
6   for(i=1; i≤k; i++)
7   {  t_{i+1}= NEXT(t_i)
8      if t_{i+1}≠NULL
9     {  if (Set_1= {x ∈T | t_i||_L x })≠(Set_2={x ∈T | t_{i+1}||_L x })
10        {  ts_FORWARD=ADD_FORWARD(Set_1 ∪ {x ∈T | {y ∈T | x||_L y }==Set_1 })
11           ts_FORWARD=ADD_FORWARD(Set_2 ∪ {x ∈T | {y ∈T | x||_L y }==Set_2 })  }
12        if (Set_3= {x ∈T | t_i#_L x })≠(Set_4={x ∈T | t_{i+1}#_L x })
13        {  ts_FORWARD=ADD_FORWARD({x ∈T | {y ∈T | x#_L y }==Set_3 })
14           ts_FORWARD=ADD_FORWARD({x ∈T | {y ∈T | x#_L y }==Set_4 })  }  }  }
```

```
15  for(i=k; i≤k; i--)
16  {  t_{i-1}= PREVIOUS(t_i)
17    if t_{i-1}≠NULL
18    {  if (Set_1= {x ∈T | t_i||_L x })≠(Set_2={x ∈T | t_{i-1}||_L x })
19       {  ts_BACKWARD=ADD_BACKWARD(Set_1 ∪ {x ∈T | {y ∈T | x||_L y }==Set_1 })
20          ts_BACKWARD=ADD_BACKWARD(Set_2 ∪ {x ∈T | {y ∈T | x||_L y }==Set_2 })  }
21       if (Set_3= {x ∈T | t_i#_L x })≠(Set_4={x ∈T | t_{i+1}#_L x })
22       {  ts_BACKWARD=ADD_BACKWARD({x ∈T | {y ∈T | z#_L y }==Set_3 })
23          ts_BACKWARD=ADD_BACKWARD({x ∈T | {y ∈T | x#_L y }==Set_4 })  }  }  }
24  ts'= SYNCRETIZE(ts_FORWARD, ts_BACKWARD)
25  WFN'=GENERATION(ts')
26  WFN^N= Choice_add (ts'+ WFN')
27  end
```

Figure 5. Example of constructing redundancy path



a Unsound process structure b Breach the original function c Deal with parallel structure d Deal with choice structure

Figure 5 illustrates a case of dealling with parallel structure and choice structure when AvailEvo algorithm constructs redundancy paths. Supposing that in composite service shown in Figure 5a, the transition path of the highest execution frequency is $t_1t_2t_3t_4t_5$ and availability of t_2 and t_4 is poor, then basic transition sequence *ts* is $t_2t_3t_4$. Because transitions in *ts* can not correspond with a sound sub WF-net ,obviously *ts* is not a redundancy path. If here *ts* is regarded as a redundancy path, a unsound evolved WF-net will resulted in. As shown in Figure 5a, if redundancy path $t_2't_3't_4'$ is executed, the WF-net will be unable to end normally, because the new composite servcie after evolution is unsound. As shown in Figure 5b, $t_2't_3't_4'$ is still regarded as a redundancy path, and a operation *Choice_add* are executed with a sound sub net in the WF-net. In such a case, when the redundancy path is executed, transition t_5 will be ignored. Obviously, functions of the original composite servcie are destroyed. According to processing method of parallel structure in AvailEvo algorithm, "join" transition t_5 with parallel structure is added. Final the redundancy path become $t_2t_3t_4t_5$. The redundancy path can correspond with a sound sub net, and functions of the new composite service are consistent with functions before evolution when the redundancy path is executed(see Figure 5c). To deal with choice structure is simple correspondingly. Supposing that in the composite service shown in Figure 2d, the transition path of the highest execution frequency is $t_1t_2t_6t_4t_7t_5$, availability of t_6 and t_7 is poor, then basic transition sequence *ts* is $t_6t_4t_7$,that also is redundancy path. Because, in this case, no other transitions are consistent with t_6 in choice relation (i.e. be choice relation with t_3), if there are such transitions, they need to be added into the redundancy path (see Figure 5d).

CONCLUSION AND FUTURE WORK

In this chapter, we are concerned with the dependability issue in service-oriented software applications. Service composition is widely considered as an effective method to implement a service oriented application, thus the dependability of a composited service becomes the core problem to achieve dependable service oriented applications. We analyze the challenges to achieve a dependable composite service. Unlike traditional software applications, the dependability of service oriented software is largely affected by the uncertainty of component services and runtime environments. Moreover, as component services are provided by third-party providers, it is almost not possible to modify an undependable component service. We find out that component services and business process structure are two important factors to impact the corresponding composite service. Redundancy is a typical and effective method to deal with the dependability issues in distributed systems. We propose a two-level redundancy framework for dependable composite services. At component service level, we study three redundancy mechanisms including active, passive and hybrid redundancy. Especially we focus on how to determine the number of backup component services so as to comply with the cost constraints. Meanwhile we propose an adaptive control mechanism using Kafman Filter and Markov model to guarantee consistent dependability of a composite service. At business process level, we propose to use structural redundancy to deal with the possible fragile business process structure. In practice, one important problem is how to integrate the two redundancy mechanisms to improve composite service dependability. In fact, the two mechanisms can complement with each other to form a complete redundancy solution. In KAF, one assumption is that there are abundant component services to select. That is not true in practical service oriented systems. Given an expected dependability value O_k, it is possible that a real system cannot meet the dependability requirement. In such situation, an alternative approach is to change the business process structure, i.e. to adopt structural redundancy. When a composite service is added a redundant execution path, more component services can be used to improve the composite service dependability.

Our present work provides a general framework and some relevant ideas to address the dependability issue in service-oriented applications. As a matter of fact, there are still a few open problems to research along this direction. For component service redundancy, more comprehensive and finer grained QoS models for composite services are needed to be further investigated, e.g., to define parameters in more flexible way. On the other hand, richer composition models other than supporting the general mode of sequence, parallel, choice and iteration, e.g., more complex relationships, such as compensation and transaction, should be considered. While for structural redundancy, our present work requires that a business process segment should have the same structure with its backups. When service resources become more abundant, there can be heterogeneous business process segment satisfying the same function. Therefore, how to discover and match these heterogeneous business process segments is an open problem for structural redundancy method.

REFERENCES

Aalst, W. M. P. v. d. (2000). Workflow Verification: Finding Control-Flow Errors Using Petri-Net-Based Techniques. In *Business Process Management: Models, Techniques, and Empirical Studies* (Vol. 1806, pp. 161-183): Berlin: Springer-Verlag.

Aalst, W. v. d., & Hee, K. v. (2002). *Workflow Management Models, Methods, and Systems*. Massachusetts London, England: The MIT Press Cambridge.

Avizienis, A., Randell, B., & Landwehr, C. (2004). Basic Concepts and Taxonomy of Dependable and Secure Computing. *IEEE TRANSACTIONS ON DEPENDABLE AND SECURE COMPUTING, 1*(1), 11-33.

Baresi, L., & Guinea, S. (2006). *Towards Dynamic Web Services.* Paper presented at the Proceedings of the 28th International Conference on Software Engineering (ICSE).

Barrett, P. A., Hilborne, A. M., Veríssimo, P., Rodrigues, L., Bond, P. G., Seaton, D. T., et al. (1990). *The Delta-4 Extra Performance Architecture (XPA)*. Paper presented at the The 20th International Symposium on Fault-Tolerant Computing (FTCS).

Birman, K., Renesse, R. v., & Vogels, W. (2004). *Adding High Availability and Autonomic Behavior to Web Services*. Paper presented at the International Conference on Software Engineering (ICSE 2004).

Cardoso, J. (2002). *Quality of Service and Semantic Composition of Workflows.* Unpublished PHD thesis, University of Georgia.

Casati, F., Ilnicki, S., Jin, L.-J., Krishnamoorthy, V., & Shan, M.-C. (2000). *eFlow: A Platform for Developing and Managing Composite e-Services*. Paper presented at the Academia/Industry Working Conference on Research Challenges (AIWORC'00).

Cheng, A., Esparza, J., & Palsberg, J. (1993). Complexity Results for 1-safe Nets. *Foundations of Software Technology and Theoretical computer Science, 761*, 326-337.

Goyeva-Popstojanova, K., Mathur, A. P., & Trivedi, K. S. (2001). *Comparison of Architecture-Based Software Reliability Models*. Paper presented at the 12th International Symposium on Software Reliability Engineering.

Guinea, S. (2005). *Self-healing web service compositions*. Paper presented at the ICSE2005.

Guo, H., Huai, J., Li, H., Deng, T., Li, Y., & Du, Z. (2007). *ANGEL: Optimal Configuration for High Available Service Composition.* Paper presented at the 2007 IEEE International Conference on Web Services (ICWS),.

Guo, H., Huai, J., Li, Y., & Deng, T. (2008). *KAF: Kalman Filter Based Adaptive Maintenance for Dependability of Composite Service*. Paper presented at the International Conference on Advanced Information Systems Engineering (CAiSE).

Hamadi., R., & Benatallah., B. (2003). *A Petri Net-based Model for Web Service Composition*. Paper presented at the Fourteenth Australasian Database Conference (ADC2003).

Harney, J., & Doshi, P. (2006). *Adaptive web processes using value of changed information*. Paper presented at the International Conference on Service Oriented Computing (ICSOC).

Harney, J., & Doshi, P. (2007). *Speeding up adaptation of web service compositions using expiration times*. Paper presented at the International World Wide Web Conference (WWW).

Haykin, S. (2002). *Adaptive Filter Theory (4th Ed.)*: Pearson Education.

Kai-yuan, C. (1995). *Base of Software Reliability Engineering*: Beijing:Tsinghua University Press.

Liu, Y., Ngu, A. H., & Zeng, L. Z. (2004). *QoS computation and policing in dynamic web service selection*. Paper presented at the International World Wide Web Conference (WWW).

Mennie, D., & Pagurek, B. (2000). *An Architecture to Support Dynamic Composition of Service Components*. Paper presented at the Workshop on Component-Oriented Programming (WCOP)

N.Budhiraja, Marzullo, K., & Schneider, F. B. (1992). *Primary-backup protocols: Lower bounds and optimal implementations.* Paper presented at the of the Third IFIP Conference on Dependable Computing for Critical Applications.

Puterman, M. L. (1994). *Markov Decision Processes: Discrete Stochastic Dynamic Programming*: Wiley-Interscience

Ramamoorthy, C. V., & Bastani, F. B. (1982). Software Reliability - Status and Perspectives. *IEEE Transaction on Software Engineering, SE-8, No 4*, 354 -371.

Ran, S. (2003). A model for web services discovery with QoS *ACM SIGecom Exchanges, 4*.

Rinderle, S., Reichert, M., & Dadam, P. (2004). Correctness criteria for dynamic changes in workflow systems--a survey. *Data & Knowledge Engineering, 50*, 9-34.

Salas, J., Pérez-Sorrosal, F., Patiño-Martínez, M., & Jiménez-Peris, R. (2006). *WS-Replication: A Framework for Highly Available Web Services*. Paper presented at the the 15th International Conference on the World Wide Web (WWW'06).

Schneider, F. B. (1990). Implementing Fault-Tolerant Services Using the State Machine Approach: A Tutorial. *ACM Computing Surveys, 22*, 299-319.

Solanki, M., Cau, A., & H.Zedan. (2006). *ASDL: A Wide Spectrum Language For Designing Web Services*. Paper presented at the the 15th International Conference on the World Wide Web (WWW'06).

Sun, H., Wang, X., Zhou, B., & Zou, P. (2003). *Research and Implementation of Dynamic Web Services Composition* Paper presented at the Advanced Parallel Processing Technologies (APPT).

Tartanoglu, F., Issarny, V., Romanovsky, A., & Levy, N. (2003). *Coordinated forward error recovery for composite Web services*. Paper presented at the IEEE Symposium on Reliable Distributed Systems (SRDS).

Tsai, W. T., Zhang, D., Chen, Y., H. Huang, Paul, R., & Liao, N. (2004). *A Software Reliability Model for Web Services*. Paper presented at the 8th IASTED International Conference on Software Engineering and Applications.

Verma, K., Sivashanmugam, K., Sheth, A., Patil, A., Oundhakar, S., & Miller, J. (2005). METEOR-S WSDI: A Scalable P2P Infrastructure of Registries for Semantic Publication and Discovery of Web Services *Information Technology and Management, 6*(1), 17-39.

Ye, X., & Shen, Y. (2005). *A Middleware for Replicated Web Services.* Paper presented at the Proceedings of the IEEE International Conference on Web Services (ICWS ' 05).

Ye, X., Zhou, J., & Song, X. (2003). On reachability graphs of Petri nets. *Computers & Electrical Engineering, 29*(2), 263-272.

Zeng, J., Huai, J., Sun, H., Deng, T., & Li, X. (2009). *LiveMig: An Approach to Live Instance Migration in Composite Service Evolution*. Paper presented at the 2009 IEEE International Conference on Web Services (ICWS).

Zeng, J., Sun, H., Liu, X., Deng, T., & Huai, J. (2010). Dynamic Evolution Mechanism for Trustworthy Software Based on Service Composition. *Journal of Software, 21*(2), 261-276 (in Chinese).

Zhang, L.-J., Zhang, J., & Cai, H. (2007). *Services Computing*. Beijing: Tsinghua University Press.