A Real-Time Reliability Model for Ontology-Based Dynamic Web Service Composition

Harmeet Chawla and Haiping Xu
Computer and Information Science Department
University of Massachusetts Dartmouth
North Dartmouth, MA 02747, USA
{hchawla, hxu}@umassd.edu

MengChu Zhou
Department of Electrical and Computer Engineering
New Jersey Institute of Technology
Newark, NJ 07102, USA
zhou@njit.edu

***Abstract*—Ontology-based web service composition allows for integration of available web services in real-time to meet desired objectives. In order to evaluate the quality of composite web services at runtime, there is a pressing need to define a feasible real-time web service reliability model. In this paper, we present such a model. We first introduce a dynamic process model that supports the evaluation of web service reliability. Then we provide a hybrid reliability model for atomic web services by considering both software and hardware aspects of the services. In order to calculate efficiently the reliability of ontology-based dynamic composite web services, we present a recursive algorithm that evaluates the reliability of various service composition constructs in real-time. Finally, we use a case study to show how to compute and monitor the reliability of composite web services in real-time, and how our approach supports reliable ontology-based dynamic web service composition.**

Keywords-web service composition; ontology; reliability model; quality of service (QoS); dynamic process model; real-time.

I. INTRODUCTION

Web services are self-contained software components that can be published, discovered and invoked over the Internet. However, in many cases, a standalone web service is not sufficient to provide the needed functionality for certain user requirements [1]. This leads to the idea of composing different web services in order to meet such requirements. The process of web service composition can be either *static* or *dynamic*. In the former, the services are pre-determined during the design phase; while in the latter, only the service template can be initially defined, but the available web services associated with each constitutive component defined in the template must be determined at runtime. In order to discover, invoke, compose and monitor web services with a high degree of automation, we can use the semantic and ontological techniques [2]. In this paper, we adopt the semantic markup language for web services (OWL-S), which is an ontology language, to formally specify the semantics of web services. Such specification makes the definitions of web services machine-understandable. Decisions on adoption of a web service for service composition require matching not only the functional properties of the service, but also the nonfunctional properties such as service reliability. The functional and non-functional properties of a web service can be formally specified using an OWL-S profile. An OWL-S profile can be published and stored in an ontology-based UDDI (Universal Description, Discovery and Integration) such as an OWL-S/UDDI service registry so that when a service client searches it, services with matching profiles can be discovered and the corresponding grounding information can be retrieved. Since there may be more than one matched profiles published in an OWL-S/UDDI, a service client has to select one of them according to certain criteria such as the service reliability. This requires the calculation of service reliability in real-time. To achieve this, we first provide a hybrid reliability model for atomic web services, which considers both software and hardware aspects of the services. Then, to calculate the reliability of composite web services, we design an efficient recursive algorithm that evaluates various service composition constructs in real-time. By employing a real-time reliability model for service composition, our approach not only supports the selection of desirable web services for dynamic web service composition, but also provides an effective way to monitor the reliability of both atomic and composite web services in real-time.

Service reliability has been an important measure of the quality of web services for service composition. There are many different kinds of existing software reliability models for web services. Li *et al.* developed a user-oriented software reliability model for evaluating the reliability of web services [3]. Their approach can be used to evaluate the reliability of atomic web services based on an extended UDDI model, and to predict the overall reliability of a composite web service using a Business Process Execution Language (BPEL)-specified structure. Tsai *et al.* proposed a software reliability model that could dynamically evaluate the reliability of atomic and composite web services [4]. The model first calculates the reliability of atomic services by using group testing and majority voting, and then the overall reliability of a composite service by using an architecture-based model. The above approaches consider only the software aspect of web services and assume the reliabilities of the machines that host the web services are near perfection. Furthermore, they are typically based on BPEL-like architectures that require static binding of available web services with the service components defined in a process model at design time. Thus, they do not support dynamic web service composition. In contrast, our approach considers both software and hardware aspects of service reliability. Furthermore, since our approach supports ontology-based web service composition, it provides a feasible way for maintaining reliable composite web services at runtime.

There are also many previous efforts on ontology-based web service composition and formal modeling of dynamic service composition. Ma *et al.* introduced an ontology-based model for web service composition, called OMWSC [5]. They presented a goal-driven and ontology-based architecture that could support automatic composition of web services. Xiong *et al.* presented a service functional configuration net based on Petri nets for automatic service composition [6]. They described the configuration specification for component services through the structure of disassembly Petri nets, and obtained the optimal one using linear programming. Tan *et al.* introduced a formal method to derive possible web service composition candidates based on a service portfolio [7]. They first generated a service net (SN) containing all needed operations, and then used Petri net decomposition techniques to derive a subnet of SN that meets the business requirements. Although the above approaches support dynamic web service composition, most of them consider only the functional requirements for service composition, and none of them attempted to use service reliability as a major criterion for service selection. Different from the above approaches, we developed a real-time service reliability model for ontology-based web service composition. Thus, our approach supports dynamic composition of reliable web services at runtime.

II. Ontology-Based Web Service Composition

A. Ontology-Based Web Service Composition

Specifying service related information semantically is the key to effective dynamic service discovery and service composition. Web service description language (WSDL) can be used to describe the syntax of a web service such as its input and output parameters, as well as related information such as the service provider and the service endpoint address; however, it does not support specifying the semantics of a web service. Therefore, WSDL has its limitations in supporting the dynamic service discovery, execution, composition and interoperation of web services. On the other hand, ontology-based techniques can not only be used to describe the syntax but also the semantics of web services. In ontology-based semantic modeling, the terms used in the concerned domain can be precisely defined, thus services can be matched based on their semantics rather than their syntax or keywords. In our approach, the process model is defined as a template, called a *Process Model Template* (PMT). In order to instantiate a PMT into an *Instantiated Process Model* (IPM), we need to search for available web services for its constitutive service components. For this purpose, each service component, which is also called a *simple component*, is associated with an OWL-S profile template. An OWL-S profile template is essentially an incomplete OWL-S profile with semantically defined input, output, preconditions and effects so that it can be matched with existing OWL-S profiles published in an OWL-S/UDDI. Based on the matched OWL-S profiles, a simple component can be bound to either an atomic or a composite web service. When a matched web service is a composite one, its process model must also be defined using a PMT, which can be instantiated further in the same manner. This procedure repeats until all matched web services become atomic. In this case, the process of instantiating the PMT is completed.

Since a profile template can be matched with more than one published OWL-S profiles, the most desirable one must be selected for execution. The criteria for service selection can be based on multiple features, such as provider, reliability, and price; however, for simplicity, in this paper, we only consider the reliability as the sole criterion for service selection. Since service reliability is a dynamic property, it requires the system be able to calculate it in real-time. This serves two purposes, namely the selection of web services for dynamic service composition and the real-time monitoring of web service reliabilities. With the monitoring function, a reliable service-oriented system can be maintained continuously – when the reliability of some web services drop to an unacceptable level, they can be replaced by other reliable ones at runtime.

B. Dynamic Process Model

A PMT is defined as a dynamic process model that consists of structural components, such as a *sequence* component and a *parallel* one. Each structural component contains simple components and possibly other structural ones. At runtime, a simple one can be bound to either an atomic web service or a composite one. The PMT is formally defined using Backus-Naur form (BNF) as in Fig. 1. As shown in the definition of PMT, the reliability requirements for a process model are described by using two parameters, namely `<desired reliability>` and `<marginal reliability>`. The former defines the desired reliability of an instantiated process model (i.e., a composite web service). A composite web service with at least the desired reliability is considered to be reliable. On the other hand, a composite web service with at least the `<marginal reliability>` but less than the `<desired reliability>` is considered as not reliable but is acceptable for a temporary usage. In this case, the application must try to search for more reliable ones in order to meet the reliability requirements of the composite web service. If it fails, a warning message must be sent to the user of the application. Furthermore, when the reliability of a composite web service becomes less than the marginal reliability, the composite web service becomes unacceptable and must stop its execution.

```
<PMT> ::= <pmt><desired reliability>
    <marginal reliability><process model></pmt>
<desired reliability> ::= <float>
<marginal reliability>::= <float>
<process model> ::= <process><start>
    <structural component><finish></process>
<structural component> ::= <sequence component>|
    <parallel component>|<loop component>|
    <choice component>
<sequence component> ::= <sequence><component>
    <component>{<component>}</sequence>
<parallel component> ::= <parallel>component>
    <component>{<component>}</parallel>
<loop component> ::= <loop><condition><component>
    </loop>
<condition> ::= <Boolean expression>
<choice component> ::= <choice><component>
    <component>{<component>}</choice>
<component> ::= <simple component>|
    <structural component>
<simple component> ::= <simple><component id>
    <owl-s profile template></simple>
<component id> ::= <string>
```

Figure 1. Definiton of PMT in BNF

A structural component can be one of the four major composition constructs, namely *sequence*, *parallel*, *loop*, and *choice* [4]. We now give a description of the major constructs defined in a PMT as follows.

Sequence In a sequence structural component, the constitutive components are executed in series. Fig. 2(a) shows an example with two simple components *A* and *B*, defined in a PMT. The directed arrow between *A* and *B* indicates the order of execution. When one of the constitutive components in a sequence construct is not functioning, the entire sequence structural component is not.

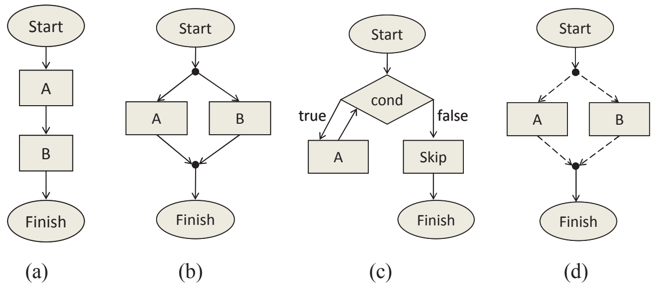


Figure 2. Examples: (a) sequence (b) parallel (c) loop (d) choice constructs

Parallel In a parallel structural component, two or more constitutive components can execute concurrently. The structural component terminates when all of its components have finished their execution. Fig. 2(b) shows a parallel structural component with two simple components *A* and *B*.

Loop A loop structural component refers to the repetitive execution of a simple or structural component. As shown in Fig. 2(c), when the condition *cond* is evaluated to be *true*, simple component *A* is executed repetitively. Only when it becomes *false*, the loop construct terminates. Note that the "skip" component is an empty component that is used to separate the loop construct from other components.

Choice In a choice structural component, only one of the constitutive components can be selected for execution. Fig. 2(d) shows an example of a simplified choice structural component with no guards (conditions). When guards are not defined, the component (*A* or *B*) to be selected for execution is determined manually by user inputs.

Fig. 3 shows an example of PMT with multiple structural components. The process model is defined as a parallel component with two sequence components defined as its constitutive ones, which can execute concurrently. The first sequence component consists of two components, namely simple component *A* and a choice structural component with two simple ones *B* and *C*. The second sequence component consists of two simple ones *D* and *E*.

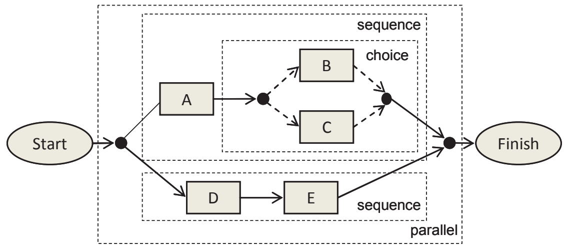


Figure 3. An example of PMT with mutiple structural components

C. *Instantiation of PMT into IPM*

When a PMT is instantiated into an IPM, each simple component defined in the PMT needs to be bound to either an atomic web service described by a WSDL file or a composite one specified by another PMT. Since such information must be recorded in the IPM when a matched web service is selected, we define an IPM as an extended version of a PMT with a set of placeholders for the mapping information that details how a simple component can be bound to a selected web service. At runtime, the placeholders are filled up with such detailed service information. The IPM can be formally defined using BNF as in Fig. 4.

```
<IPM> ::= <ipm><PMT><simple component mapping>
    {<simple component mapping>}</ipm>
<simple component mapping> ::= <simple mapping>
    <component id><placeholders for matched
    service></simple mapping>
<component id> ::= <string>
<placeholders for matched service> ::= <ph>
    <real-time reliability><service id>
    <service type>(<atomic service grounding
    info>|<composite service grounding info>)</ph>
<real-time reliability> ::= <float>
<service id> ::= <string>
<service type> ::= ”atomic” | ”composite”
<atomic service grounding info> ::= <wsdl file>
<composite service grounding info> ::= <ipm file>
```

Figure 4. Definition of IPM in BNF

As shown in the definition, the placeholders for a matched and selected web service are enclosed by the `<ph>…</ph>` tags. The item `<real-time reliability>` is the reliability of the web service that is bound to a simple component, which is calculated at runtime. The two parameters `<service id>` and `<service type>` are the identification and the type of the selected web service, respectively, where the service type can be either "atomic" or "composite." If the selected web service is an atomic one, the placeholder `<atomic service grounding info>` must be filled with the address of its WSDL file. On the other hand, if the selected one is a composite one, the placeholder `<composite service grounding info>` needs to be filled with the location of the IPM file that specifies the composite web service. The procedure for instantiating a PMT into a set of IPMs is defined in Algorithm 1. Note that it is defined recursively because a simple component can be mapped to a composite web service specified by another PMT file. In this case, that IPM file must also be instantiated by invoking the method *Instantiate-PMTintoIPM* recursively. As a result, the output of the algorithm is a set of IPM files organized in a tree-like structure. Furthermore, the instantiation process requires calculating web service reliabilities in real-time. This is because when more than one matched web services are discovered, their reliabilities need to be calculated in real-time, such that the most reliable one can be selected for execution. In order to calculate the reliability of a composite web service, an external method *CalculateReliability* (to be discussed in Section III-B) must be invoked. Note that calculating the reliability of a composite web service requires its IPM file as an input (line 17 of the algorithm), which has been generated after the recursive method call *InstantiatePMTintoIPM (ws.pmt)* in line 16.

Algorithm 1: Instantiate PMT into IPMs

Input: a *pmt* file to be converted
Output: a set of *ipm* files arranged in a tree structure.

```
1. InstantiatePMTintoIPM (File fname.pmt)
2.   copy fname.pmt to fname.ipm & create placeholders in fname.ipm
3.   create a PMT object pmt_obj from file fname.pmt
4.   foreach simple component sc in pmt_obj.process_model
5.     initialize sc.realtime_reliability to 0
6.     query OWL-S/UDDI using sc.profile_template
7.     foreach matched web service ws
8.       if (ws is atomic)
9.         extract reliability parameters from OWL-S profile
10.        calculate ws.reliability for atomic web service ws
11.        if (ws.reliability > sc.realtime_reliability)
12           sc.realtime_reliability = ws.reliability
13.          sc.service_id = ws.id
14.          sc.service_type = “atomic”
15.      else if (ws is composite)
16.        InstantiatePMTintoIPM (ws.pmt)   // create ws.ipm file
17.        ws.reliability = CalculateReliability (ws.ipm, true, null)
18.        if (ws.reliability > sc.realtime_reliability)
19.          sc.realtime_reliability = ws.reliability
20.          sc.service_id = ws.id
21.          sc.service_type = “composite”
22.    if (sc.service_type == “atomic”)
23.      extract wsdl address from owl-s profile of sc.id
24.      set sc.service_type = “atomic”
25.      set sc.wsdl_file to the wsdl file of sc.service_id
26.    else
27.      set sc.service_type = “composite”
28.      set sc.ipm to the ipm file of sc.service_id
29.  save the service info of sc into its placeholders in file fname.ipm
```

III. Real-Time Service Reliability Evaluation

Service reliability represents an important attribute for the QoS of a web service deployed on a certain machine. In this paper, we take into account both hardware and software aspects of service reliability as both are needed to determine the reliability of a deployed atomic web service. Then based on the reliability of the participating atomic web services, we can calculate the reliability of a composite web service according to its dynamic process model.

A. A Hybrid Reliability Model for Atomic Web Services

Software reliability growth models (SRGM) are based on the assumption that the number of faults of a software system can be continuously reduced, which results in growth of its software reliability [8]. Although SRGM has been considered as one of the most successful techniques in software reliability engineering, it is most suitable for measuring and predicting the improvement of software reliability through the testing process [9]. In this paper, we assume that there are no features added and no faults removed once an atomic web service is deployed. In this case, the failure intensity of the software component (i.e., the atomic web service) will be constant. According to [10], the number of failures of the service in a given time follows a Poisson distribution. The corresponding formula to calculate software reliability can be defined as in (1).

$$R_{software}(\tau) = \exp(-\lambda\tau) \qquad (1)$$

where λ is the constant failure intensity and τ is the execution time of the web service. When a web service is computation-intensive and executed continuously, τ can be further replaced by the elapsed time since the service is deployed.

On the other hand, hardware reliability can be represented by the two-parameter Weibull distribution [11]. The corresponding formula to calculate the reliability of a hardware component can be defined as in (2).

$$R_{hardware}(t) = \exp[-(\beta t)^{\alpha}] \qquad (2)$$

where α is the shape parameter ($\alpha > 0$), and β is the scale parameter ($\beta > 0$). Note that the hardware reliability decreases with time. This is because after a certain age, the product enters its wear-out phase and the failure rate starts to increase.

Bowles tried to derive a combined hardware and software reliability model for networks [12]. According to [12], “The probability of successful operation of a device is the probability that the hardware does not fail and the probability that the software does not fail.” Inspired by this idea, in this paper, we calculate web service reliability by considering both the reliability of the web service and the reliability of the machine where the web service is deployed. For simplicity, we assume that a web service is initially deployed on a new machine with the perfect reliability of 1. Thus, time t can be defined as $t = t_{current} - t_0$, where $t_{current}$ is the time when the reliability is calculated and t_0 is the time when the web service is deployed. Based on (1) and (2), the hybrid reliability model for atomic web service can be defined as in (3).

$$\begin{aligned} R_{service}(t) &= R_{software}(t) * R_{hardware}(t) \\ &= \exp(-\lambda t) * \exp[-(\beta t)^{\alpha}] \end{aligned} \qquad (3)$$

where $R_{software}(t)$ is the reliability function of the atomic web service and $R_{hardware}(t)$ is the reliability function of the machine that hosts the atomic web service.

The needed parameters for calculating the reliability of a deployed atomic web service can be stored in an OWL-S profile as follows.

```
<profile: Reliability>
  <FailureIntensity datatype=”float”> 0.0001
  </FailureIntensity>
  <Shape datatype=”float”>2</Shape>
  <Scale datatype=”float”>0.00002</Scale>
  <Date datatype=”date”>07/16/2010</Date>
  <Time datatype=”time”>14:30</Time>
</profile: Reliability>
```

Note that in the reliability portion of an OWL-S profile, we also include the parameters of the deployment date and time of the atomic web service because they are needed for calculating the web service reliability. Since the reliability of an atomic web service is time-dependent, the above parameters must be retrieved at runtime such that the reliability of the atomic web service can be calculated in real-time.

B. Reliability Model for Composite Web Services

The overall reliability of a composite web service depends on its structure, the degree of independence between service components and the availability of its constitutive web services. In order to calculate the reliability of a composite web service, we first need to consider the reliability of the major structural components defined in Section II-B. Based on previous work [3, 4], we now define the reliability model for each structural component as follows.

Sequence The reliability of a composite web service composed of services in sequence can be calculated as in (4).

$$R_{sequence}(t) = \prod_{i=1}^{n} R_{service\_i}(t) \tag{4}$$

where the constitutive web services *service_i* ($1 \leq i \leq n$) are all independent of each other (i.e., the failure of one service does not lead to the failure of the others), and $R_{service\_i}(t)$ is the reliability function of each constitutive atomic or composite web service.

Parallel The reliability of a composite web service composed of services in parallel can be calculated as in (5).

$$R_{parallel}(t) = \prod_{i=1}^{n} R_{service\_i}(t) \tag{5}$$

where the constitutive web services *service_i* ($1 \leq i \leq n$) are all independent of each other. Note that the failure of any constitutive service results in the failure of the composite one because the successful termination of the latter requires the successful termination of all of its constitutive web services.

Loop The reliability of a composite web service composed of web services in a loop can be calculated as in (6).

$$R_{loop}(t) = \min_{0 \leq i \leq n}(R_{service\_loopbody}(t + i * \Delta t)) \tag{6}$$

where $R_{service\_loopbody}(t)$ is the reliability function of the loop body that can be an atomic web service or composite one; n is the number of iterations; and Δt is the execution time of each iteration. When $n*\Delta t$ is not a large value, the reliability of the loop structural component would be approximately the same as when it was executed the first time.

Algorithm 2: Calculate composite web service reliability

Input: an *ipm* file for a composite web service
Output: the real-time reliability of the composite web service

```
1.  CalculateReliability (File fname.ipm, Boolean initialization,
2.                        StructuralComponent structcom)
3.    initialize reliability to 1
4.    if (initialization == true)                    // step 1
5.      create an IPM object ipm_obj from file fname.ipm
6.      foreach simple component sc in ipm_obj
7.        if (sc.service_type == "atomic")
8.          calculate sc.realtime_reliability
9.        else sc.realtime_reliability =            // sc is "composite"
10.         CalculateReliability(sc.ipm, true, null)
11.   strc = ipm_obj.process_model                   // step 2
12.   if (strc is squenceComponent)
13.     foreach component com in strc
14.       if (com is simpleComponent)
15.         reliability *= com.realtime_reliability
16.       else reliability *= CalculateReliability(null, false, com)
17.   else if (strc is parallelComponent)
18.     foreach component com in strc
19.       if (com is simpleComponent)
20.         reliability *= com.realtime_reliability
21.       else reliability *= CalculateReliability(null, false, com)
22.   else if (…)      // other cases: redundant component,
23.   …                // loop component, and choice component
24.   return reliability
```

Choice The reliability of a composite web service composed of web services in choice can be calculated as in (7).

$$R_{choice}(t) = \min_{1 \leq i \leq n}(R_{service\_i}(t)) \tag{7}$$

Since we consider the worst-case scenario, the reliability of a choice structural component equals the minimal reliability of the constitutive web services.

The algorithm for calculating the reliability of a composite web service is defined recursively as in Algorithm 2. Algorithm 2 involves two steps when calculating the reliability of a composite web service. In its first step, the reliabilities of all simple components are calculated. In a case when a simple component is bound to a composite web service, the method *CalculateReliability* must be invoked recursively with the parameter *initialization* being *true*. In the second step, the reliabilities of the structural components are calculated. Similarly, when a structural component contains another structural component as its constitutive component, the method *CalculateReliability* is also invoked recursively, but this time, the parameter *initialization* is set to *false* indicating that the algorithm is now processing its second step.

IV. Case Study

To demonstrate the effectiveness of our approach, we utilize a case study of financial services, which involves investment in mutual funds and stocks. We define a process model for financial investment as a composite web service with a choice structure. As shown in Fig. 5, the choice is between buying mutual funds or stocks. The upper structural component represents mutual fund investment wherein the investor has a choice of investing in three types of mutual funds. They are *equity* that involves high risk, high gain funds, *debt* that represents low risk low gain funds, and *balanced fund* that gives almost steady gain with medium risk. In order to use the financial services, the investor needs to provide information about the type of fund, investment amount and personal information. For example, if the user wants to invest in equity funds, the *Equity* web service is selected and invoked, which provides a list of equity mutual funds along with their returns. Then the *SelectMutualFund* service is invoked to automatically choose the service with best returns for the user. Finally, the *BuyMutualFund* service will be invoked to buy the selected mutual funds. Similarly, the web services *Debt* and *Balanced* can be invoked to provide a list of debt funds and a list of balanced funds with returns, respectively. On the other hand, if buying stock is chosen, the process model has the *BuyStocks* service, where the input to this service is the stock to be bought, the number of stocks, and the rate at which the user wants to buy. When the specified stock is available at the preferred rate, the *BuyStocks* service is automatically invoked to buy the specified number of stocks.

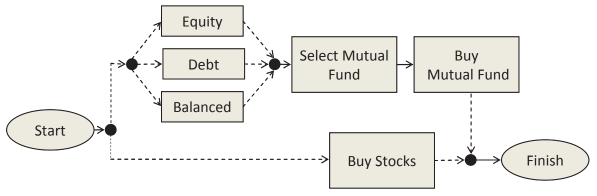
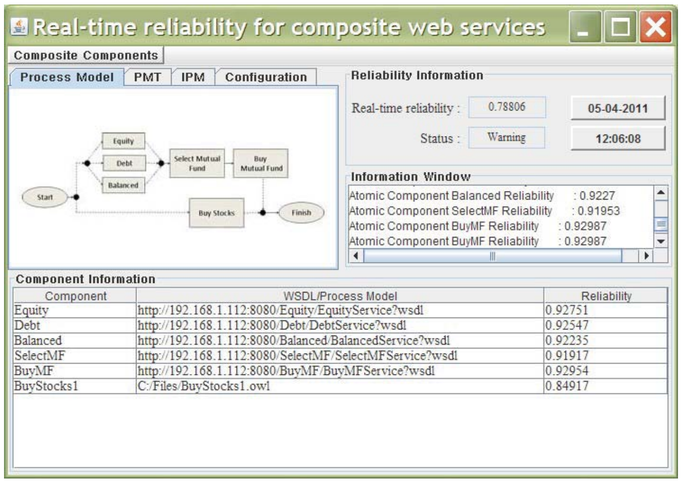


Figure 5. A process model for financial investment

In order to ensure the desired reliability of the financial services for investment, we develop a prototype service reliability monitoring tool. Fig. 6 shows the interface for monitoring the reliability of the financial investment service in real-time. Under the *ProcessModel* tab, it presents the predefined process model (as shown in Fig. 5) at the left hand side, and displays the real-time reliability of the composite web service at the right hand side. The interface also shows the status of the composite web service, which could be *normal*, *warning* or *unacceptable*. A *normal* status indicates that the reliability of the composite web service equals to or more than the desired reliability; a *warning* status indicates that the reliability is below the desired reliability but no lower than the marginal reliability. When the reliability falls below the marginal reliability, the status becomes *unacceptable*. Note that in this example, the *desired* and *marginal* reliabilities are set to 0.85 and 0.75, respectively. They can be easily re-configured by clicking on the *Configuration* tab. In addition, the interface also displays the current date and time as well as execution information of the composite web service. At the bottom part of the interface, all services bound to the simple components are listed. If the service is atomic (e.g., *EquityService*), the address of its WSDL and its real-time reliability are displayed (e.g., *http://192.168.1.112:8080/equity/EquityService?wsdl* and 0.92751); otherwise, if the service is composite (e.g., *BuyStocks1*), the address of the corresponding IPM file and its real-time reliability are displayed (e.g., *C://Files/Buy-Stocks1.owl* and 0.84917). The detailed information about a composite web service (e.g., *BuyStocks1*) can be retrieved by clicking on the *CompositeComponents* menu at the top of the interface. Note that when the real-time reliability of a composite service falls below the desired reliability, some of its constitutive web services with low reliabilities must be replaced by others in order to improve its QoS.



Reliability Information

Real-time reliability : 0.78806 05-04-2011

Status : Warning 12:06:08

Information Window

```
Atomic Component Balanced Reliability    : 0.9227
Atomic Component SelectMF Reliability    : 0.91953
Atomic Component BuyMF Reliability       : 0.92987
Atomic Component BuyMF Reliability       : 0.92987
```

Component Information

Component	WSDL/Process Model	Reliability
Equity	http://192.168.1.112:8080/Equity/EquityService?wsdl	0.92751
Debt	http://192.168.1.112:8080/Debt/DebtService?wsdl	0.92547
Balanced	http://192.168.1.112:8080/Balanced/BalancedService?wsdl	0.92235
SelectMF	http://192.168.1.112:8080/SelectMF/SelectMFService?wsdl	0.91917
BuyMF	http://192.168.1.112:8080/BuyMF/BuyMFService?wsdl	0.92954
BuyStocks1	C:/Files/BuyStocks1.owl	0.84917

Figure 6. Interface for monitoring real-time web service reliability

V. Conclusions and Future Work

In this paper, we define a dynamic process model that consists of various constructs for web service composition. The dynamic process model is initially defined as a process model template, called PMT, where the constitutive components are not bound to any specific web services. At runtime, the PMT is instantiated into a set of instantiated process models or IPMs. During the instantiation process, web services that are matched with the simple components in a PMT are discovered and selected based on their real-time reliability values. In order to calculate the reliabilities of composite web services, we first present a hybrid reliability model for atomic web services. Then we define a real-time reliability model for a composite web service that aggregates the reliabilities of its constitutive components according to the definition of its dynamic process model. Our approach not only supports service selection for dynamic web service composition, but also supports maintaining a reliable composite web service at runtime by monitoring the service reliabilities in real-time. For future work, we will study existing software reliability models and propose the most suitable ones for atomic web services by considering additional factors such as software aging due to performance degradation or a sudden software crash. We will demonstrate how to automatically switch a web service to a reliable one when its reliability becomes unacceptable. By utilizing artificial intelligence techniques, we may further improve the service reliability model for automatic adjustment of its parameters at runtime. In addition, integrating the proposed reliability index into some existing approaches [5, 6] can also be considered as a worthy future direction.

References

[1] J. Rao and X. Su, "A survey of automated web service composition methods," in *Proc. of the First Int. Workshop on Semantic Web Services and Web Process Composition (SWSWPC 2004)*, San Diego, CA, USA, pp. 43-54, Jul. 2004.

[2] D. Martin, M. Paolucci, S. McIlraith, M. Burstein, D. McDermott, D. McGuinness, *et al.*, "Bringing semantics to web services: the OWL-S approach," in *Proc. of the First Int. Workshop on Semantic Web Services and Web Process Composition (SWSWPC 2004)*, San Diego, CA, USA, pp. 26-42, Jul. 2004.

[3] B. Li, Z. Su, Y. Zhou, and X. Gong, "A user-oriented web service reliability model," in *Proc. of the IEEE Int. Conf. on Systems, Man and Cybernetics*, Singapore, pp. 3612-3617, Oct. 2008.

[4] W. T. Tsai, D. Zhang, Y. Chen, H. Huang, R. Paul, and N. Liao, "A software reliability model for web services," in *Proc. of the 8th IASTED Int. Conf. on Software Engineering and Applications (SEA 2004)*, Cambridge, MA, pp. 144-149, Nov. 2004,

[5] J. Ma, Y. Zhang, and M. Li, "OMWSC - an ontology-based model for web services composition," in *Proc. of the 5th Int. Conf. on Quality Software (QSIC 2005)*, Melbourne, Australia, pp. 464-469, Sept. 2005.

[6] P. C. Xiong, Y. Fan, and M. C. Zhou, "Web service configuration under multiple quality-of-service attributes," *IEEE Trans. on Automation Science and Engineering*, vol. 6, no. 2, pp. 311-321, Apr. 2009.

[7] W. Tan, Y. Fan, M. C. Zhou, and Z. Tian, "Data-driven service composition in building SOA solutions: a Petri net approach," *IEEE Trans. on Automation Science and Engineering*, vol. 7, no. 3, pp. 686-694, Jul. 2010.

[8] M. R. Lyu, "Software reliability engineering: a roadmap," in *Proc. of the Workshop on Future of Software Engineering (FOSE'07)*, Int. Conf. on Software Engineering, Washington, DC, pp.153-170, 2007.

[9] H. Pham, *System Software Reliability*, Springer-Verlag, London, pp. 152-177, 2006.

[10] J. Musa, *Software Reliability Engineering*, McGraw-Hill, New York, pp. 310-311, 1999.

[11] A. Hoyland and M. Rausand, *System Reliability Theory: Models and Statistical Models*, John Wiley & Sons, New York, pp. 37-40, 1994.

[12] J. B. Bowles, "A model for assessing computer network reliability," in *Proc. of the IEEE Conf. on Energy and Information Technologies in the Southeast (SoutheastCon)*, Columbia, SC, USA, pp. 603-608, Apr. 1989.