Dynamic Self-Healing for Service Flows with Semantic Web Services

Wei Ren[1], Gang Chen[1], Haifeng Shen[4], Zhonghua Yang[1],

[1]School of Electrical and Electronics Engineering, [4]School of Computer Engineering, Nanyang Technological University, Singapore 639798

Jing Bing Zhang[2], Chor Ping Low[1], David Chen[3], Chengzheng Sun[4]

[2]Singapore Institute of Manufacturing Technology (SIMTech), Singapore 638075
[3]School of Computing & Information Technology, Griffith University, Australia 4111

Abstract

With an increasing complexity of business processes, self-healing capability is becoming an important issue in order to support robust service flow execution. In this paper, a dynamic self-healing mechanism is proposed, which can dynamically identify suitable alternatives and replace faulty services such that a service flow can be performed successfully despite of unexpected exceptions. This mechanism explicitly utilizes Semantic Web services for service matching and selection of a composite service in business service flow, and Semantic web services are equipped with rich business rules in a domain-dependent manner. We explore the self-healing mechanism for supporting self-healable service flow execution which is modeled in BPEL4WS. A demo system of self-healing capable Service Flow Execution is built to validate its effectiveness by a concrete scenario, PC manufacturing application.

1. Introduction

Today, increasing number of organizations expose their business functions as Web Services. As the use of Web services grows, more and more organizations are choosing Business Process Execution Language (BPEL) [1] for modeling business processes within the Service Oriented Architecture (SOA). In order to support robust service flow execution, self-healing in service flow is becoming prominent. In general, self-healing includes faults detection and recovery. BPEL provides a set of standard faults detection and recovery mechanisms, such as *Fault handler*, *Compensation handler*, and *Event handler* which are automatically executed by the BPEL execution engine. However, these basic recovery mechanisms are quite simple and scopes-oriented. The recovery is only to compensate all effects of a faulted scope and to continue the service flow execution without any results obtained from this scope, and do not support sophisticated recovery actions [3] such as service replacement. Furthermore, the standard recovery mechanisms provided by BPEL language are static in a sense that they are defined in the service flow design phase.

In this paper, we define the "self-healing service flow" as a service flow which can be locally recovered from the faults detected during service flow execution. Generally, there are two approaches to self-healing of a service flow: static and dynamic. In a static approach, self-healing is addressed at the design time, that is, for every possible faulty in a service flow, a fault handler is constructed at the design time which specifically handles the anticipated fault. If the fault encountered is not expected during the design of the service flow, the service flow is unable to handle it as there is no pre-designed handler available. On the other hand, in a dynamic approach, any fault that occurs at the run time is handled by dynamically constructing, at the run time, a replacement of services which sufficiently matches the faulty one based on some criteria. In this paper, we focus on the dynamic self-healing mechanism for service flows.

In order to support self-healable service flow execution, a self-healing mechanism is proposed to replace the faulty service dynamically. The mechanism has been utilized to build Self-healing Service Flow Generator (SSFG) which addresses the issue at the two different levels. At high level, having semantic support, OWL-S [5] is used in dynamic service discovery and composition. At the concrete level, industry-based BPEL is exploited in service execution. The SSFG bridges the gap between OWL-S and BPEL.

978-0-7695-3496-1/08 $25.00 © 2008 IEEE
DOI 10.1109/WIIAT.2008.111

It embeds the self-healing mechanism into BPEL service flow which achieves dynamic Web service replacement.

Dynamic Web service replacement during self-healing procedure involves service discovery and selection. To improve the effectiveness of service discovery, we exploit the Semantic Web service technology. The inputs and outputs of a service will be marked with domain ontologies. In addition, to facilitate policy based service selection, the description of each atomic Semantic Web service is enhanced with **local business rules** which capture the essential business logic behind the service interface; and each composite service is enhanced with **global business rules** which set the selection criteria for alternative service.

The remaining part of this paper is organized as follows. Section 2 provides a literature review of related research works. Section 3 introduces Service Flow Execution System with self-healing capability. The details of self-healing mechanism are presented in Section 4. The self-healing procedure is illustrated using a PC manufacturing scenario in Section 5. Finally Section 6 concludes the paper.

2. Related work

As systems increase in complexity, self-healing systems are attracting a number of researcher's attention. However, only a few research efforts focus on self-healing in Service Oriented Architecture (SOA) have been reported. Web Services - DIAgnosability, Monitoring and Diagnosis (WS-DIAMOND) [8] is a European research project which has two aims: (1) to develop a framework for self-healing Web Services; (2) to devise guidelines for designing services in such a way that they can be easily diagnosed and recovered during their execution.

Kunal and Amit propose a framework to elevate automatic computing from infrastructure level to process level to create Autonomic Web Processes (AWPs) [9]. AWPs are Web service based processes that support the autonomic computing properties. The behavior of AWPs is controlled by policies defined by users. However, they didn't give technical details about the policy. Our global business rules which determine the alternative plan during the healing process are conceptually similar to the AWPs' policies.

In [3], a self-healing plug-in for BPEL engine is presented to enhance the ability of a standard engine to provide process-based recovery actions. This approach is different from ours in the following aspects: (1) it uses API from activeBPEL engine [2] and therefore it is an implementation dependent solution; (2) its recovery mechanism is annotated in the BPEL service flow which can not be executed directly by any BPEL engine; while the output of our SSFG is a standard BPEL service flow; (3) Semantic Web service discovery technology is exploited in the our self-healing process; (4) our final service selection is based on rule evaluation results.

3. Self-healing capable service flow execution system

Figure 1 shows the architecture of our Service Flow Execution System with self-healing capability. It includes the following components: Service Composer, SSFG, BPEL Execution Engine, Service Repository and Knowledge Base.

The user inputs of Service Composer are the requested service capability which refers primarily to the required outputs of a service given the available inputs. By leveraging Semantic Web service technology, the service composer based on the forward-chaining algorithm will aggressively chain a group of Semantic Web services together in order to satisfy the specific requirements [16]. The output from Service Composer is a list of composite service which consists of one or more atomic services described in OWL-S with *Input*/*Output*/*Precondition*/*Effect* (IOPE).

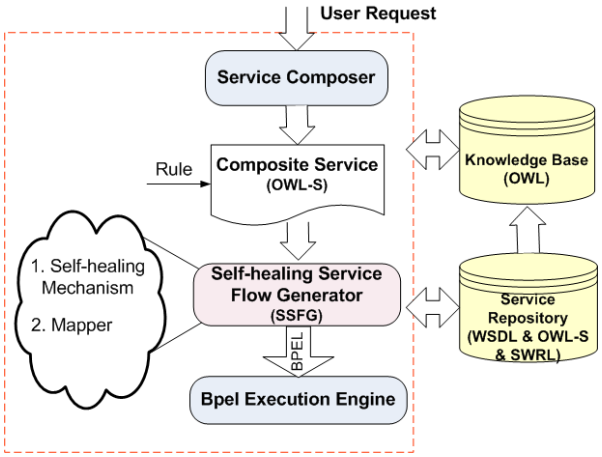


Figure 1. Self-healing capable service flow execution system

The inputs and outputs of a service are marked with domain ontologies to facilitate Semantic Web service discovery. To achieve rule based service selection, the description of each atomic Semantic Web service is enhanced with **local business rules**; and the description of each composite service is enhanced with **global business rules**. The local business rules contain

domain specific information suitable for subsequent service selection, such as completion time and estimated cost. Global business rules define a set of selection criteria (i.e. one may favor the service which has minimum cost, while the other favors minimum completion time) to choose one service among a list of alternatives based on the local business rules evaluation results. Therefore, the Global Business Rule plays a determinant role in service selection. Both local and global business rules are identified by Rule URI which can be referred by their Semantic Web service description.

SSFG is comprised of two parts: (1) self-healing mechanism; (2) a mapper which can translate OWL-S process to BPEL process by converting data, data flow and control flow into BPEL counterparts. Existing works on the mapper can be found in [4, 15]. However, to our knowledge, no mapper ever addresses self-healing issue as studied in this paper.

4. Self-healing mechanism

In this paper, we propose a self-healing mechanism to support self-healable service flow execution. It is built in the component SSFG. A sample of script generated by SSFG is shown in Figure 2. As shown in Figure 2, the BPEL *Fault handlers* is used to catch faults and provide exception handling. In order to deal with exceptional situations more locally to the place where they occurred, the *fault handlers* can associate with a *scope*. A *scope* provides the behavior context for each activity which represents an invocation of a Web service operation. When a fault happens within a *scope*, a local fault handler can deal with it before the scope's processing ends. Figure 2 shows the scoped fault handling. We utilize *catchAll* element instead of *catch* to house all the error handling activities. This is due to the matching fault handler can not be found if the operation of a Web service doesn't define the correspondent error messages as its outputs.

Once *Fault hander* catches a fault, the self-healing mechanism will be started. It is comprised of the following major steps as shown in Figure 5:

1. Find a list of alternative services
2. Update knowledge base to record the information on matched services
3. Apply both global and local business rules & update knowledge base & select one service
4. Invoke the selected service

The 4-steps self-healing procedure is realized through four well designed Web services. If an error occurs during a service invocation such as the service is currently unavailable, the self-healing mechanism will discover alternative services first given the semantic description of the faulty service as inputs. And their related information e.g. the URL of a service, the URL of local business rules, etc will be updated in the knowledge base. Thereafter, the local business rules of each discovered service will be evaluated. Based on the evaluation results, the global business rule will determine the final service. Last, the selected service will be invoked. The following sub-sections will give more details on each step of our self-healing mechanism.

```
<process>
        ...
  <scope>
    <sequence>
      <invoke> Operation of Web Service 1 </invoke>
    </sequence>
    <faultHandlers>
      <catch faultName="..."  faultMessageType="..." />
      <catchAll>
          ...<!-- Self-healing activities -->
      </catchAll>
    </faultHandlers>
  </scope>
        …
</process>
```

Figure 2. Scoped fault handling

4.1. Service discovery

In order to find an alternative service to replace the faulty service for self-healing purpose, the Semantic Web service discovery technology is exploited. There are various types of semantic matches. While exact match between services (or between service and requirements) appears to be the best choice, many flavors of relaxed match may also serve the purpose. The characterization of matches requires the semantic description of web services in OWL-S. Two aspects of service behavior described in OWL-S are of particular interests to services matching: (1) the information transformation represented by inputs and outputs, and (2) the state change produced by service execution in the form of preconditions and effects. This together is known as IOPE representing service capability. Inputs and outputs of a service are named and typed using either OWL-S ontologies or data types that XML Schema provides. They together constitute the main interface for the purpose of interacting with the service. In our previous research, we have characterized 14 different matches derived from IOPE [10]. Nevertheless, in this paper we will only highlight several matches that are explicitly utilized by our current implementation of the Service Discovery WS.

Definition 1 (Relax-Input-Match):

$$match_{r-input}(A,B) = A_{inputs} \supseteq B_{inputs}$$

Service A is said Relax-input-matched with service B if and only if inputs of service A are the superset of inputs of service B.

Definition 2 (Subsume-Output-Match):

$$match_{s-output}(A,B) = A_{outputs} \subseteq B_{outputs}$$

Service A is said Subsume-Output-matched with service B if and only if outputs of service A are the subset of outputs of service B.

Definition 3 (Relax-IO-Match):

$$match_{RIO}(A,B) = match_{r-Input}(A,B) \wedge match_{s-Output}(A,B)$$

Service A is said RIO-matched with service B if and only if inputs of service A are the superset of inputs of service B and outputs of service A are the subset of outputs of service B.

When the *Fault handlers* catch a faulty service, its service capability will be served as input of Service Discovery WS. The requested service (the faulty service) will be matched against all the advertisements stored in the service repository. Based on the concept subsumption relations in the corresponding ontology, our Service Discovery WS can determine a list of services that semantically match with the given faulty service. The matching algorithm based on the above definition is shown in Figure 3. When there is no exact matched service, the relax matched service can also work.

```
For each advertised service in the service repository
  if (  match ( adv.inputs, req.inputs ) is Relax-Input-match ) and
     ( match ( adv.outputs, req.outputs ) is Subsume-Output-Match )
      match-list.add ( adv );

 return match-list;
```

Figure 3. Matching algorithm

4.2. Knowledge base update

Knowledge is presented by ontology which is a specification of formally described, machine-readable collection of concepts and their relationships within a domain. The concept of ontology enables description of explicit semantics. In order to record the discovered services for further service selection purpose, the ontology for self-healing procedure should be built in addition to the domain ontology. During service flow execution, the knowledge base serves as a semantics container to store all the dynamic execution information. A web standard-based way for representing ontology is the use of Web Ontology Language (OWL) [11].

Figure 4 shows the ontology for self-healing. The two OWL classes, namely, **CompositeWS** and **AtomicWS** are subclass of the root class **Thing**. An instance of **CompositeWS** will be created for each of the outputs of Service Composer. Global business rules will be associated to each instance of **CompositeWS**. The faulty service and discovered services are instances of **AtomicWS**. **CompositeWS** has one object property **hasFaultyService** which has a range class **AtomicWS**. **AtomicWS** has one object property **hasReplaceableSerivce** whose domain and range classes are itself. Each class has their correspondent datatype properties, such as service Name, service URL, etc. During the knowledge base updating phase, an instance of the faulty service will be created first followed by each matched service in the matching list. The relationship between the **CompositeWS** and the faulty **AtomicWS**, the faulty **AtomicWS** and matched services are also established accordingly.

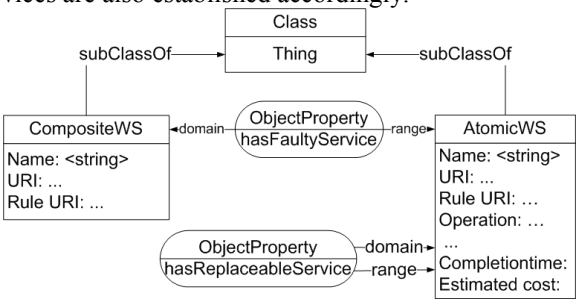
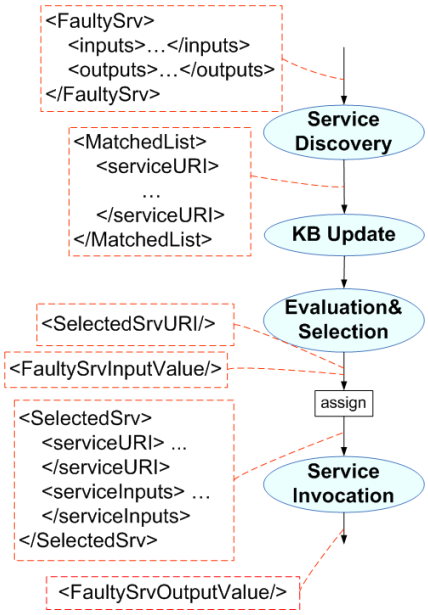


Figure 4. Ontology for self-healing

4.3. Rule evaluation and rule driven service selection

In this research work, Semantic Query-Enhanced Web Rule Language (SQWRL; pronounced squirrel) [12] is utilized for both local and global business rules. SQWRL is a SWRL [6] based query language that can be used to query OWL ontologies. SQWRL provides SQL-like operations to format knowledge retrieved from an OWL ontology. The SQWRLQueryAPI offered by Protégé is used to retrieve the result of SQWRL queries. It provides a JDBC-like Java interface. An SQWRL description can be found in the next section.

To our knowledge, the actual use of SWRL in OWL-S specification is to use SWRL-based expression to model *precondition* and *effect*. It does not support full rule specification. In this research, both local and global business rules are used to purely support service selection. Due to conceptually different from *precondition* and *effect* according to OWL-S specification, we put their URL into *ServiceParameter* of OWL-S *profile*.




```
<partnerLinks>
   <partnerLink name="SrvDiscover" />
   <partnerLink name="KBUpdate" />
   <partnerLink name="Evaluate&Select" />
   <partnerLink name="SrvInvoke" />
</partnerLinks>

<variables>
   <variable name="FaultySrv" />
   <variable name="MatchedList" />
   <variable name="SelectedSrvURL" />
   <variable name="SelectedSrv" />
   <variable name="FaultySrvInputValue" />
   <variable name="FaultySrvOutputValue" />
</variables>

<sequence>
   <invoke partnerLink="SrvDiscover"
            inputVariable="FaultySrv"
              outputVariable="MatchedList" />
   <invoke partnerLink="KBUpdate"
            inputVariable="MatchedList" />
   <invoke partnerLink="Evaluate&Select"
            outputVariable="SelectedSrvURI" />
   <assign> <copy><from>
          $SelectedSrvURL and $FaultySrvInputValue
       <to>$SelectedSrv
   </assign>
   <invoke partnerLink="SrvInvoke"
             inputVariable="SelectedSrv"
             outputVariable="FaultySrvOutputValue" />
</sequence>
```

Figure 5. Generated self-healing service flow

Two open-source projects, Protégé [13] and Jess Rules Engine [14] are exploited in the rule evaluation. Using SQWRL Query Engine bundled with the Protégé distribution, both local and global business rules can be processed by the backend Jess Rules Engine. Based on these tools, we develop the Rule Evaluation WS [7].

4.4. Service invocation

Based on the rules evaluation results and selection criteria, the final service will be selected. Given the selected service URL and the input variable of the faulty service as inputs, the Service Invocation WS will invoke the selected service. The output of the Service Invocation WS is the output of the faulty service.

The generated self-healing service flow is depicted in Figure 5. The input and output ontology of the faulty service will be fed into the Service Discovery WS. The Service Discovery WS will perform the semantic matching and return a list of matched service URL. Upon knowledge base update and business rules evaluation, the substitution service will be selected. The selected service URL and the inputs of original faulty service together will be assigned to a new variable which will be subsequently consumed by the Service Invocation WS. The output will be assigned to the original faulty service output variable which will be served as input of next Web service in the service flow. In such a way, the self-healing procedure merged with the original service flow seamlessly.

5. PC manufacturing prototyping system

In order to respond to the dynamic market demand for computers, business service flows for PC manufacturing need to be dynamically formulated and executed to share their functionalities and resources in a collaborative manner. Figure 6 shows a sub-service flow formulated in a Collaborative PC manufacturing Virtual Enterprise. After processing customer order, a monitor purchase order is issued to the *MonitorSupply* Web service which produces a delivery order. For computer assembly purpose, *HardwareShipping* Web service has to ship monitors to the company which can assemble the computers. Our self-healing capable service flow execution system will be evaluated based on this scenario.

Figure 6. Sub service flow in PC manufacturing CVE

5.1. Service discovery

In this section, we will illustrate the self-healing procedure once a fault is caught by the *Fault handler* due to the *MonitorSupply* service is currently not available. Suppose the input of *MonitorSupply* service is *LCDMonitorPurchaseOrder* and its output is *MonitorDeliveryOrder*. The input and output represent the service capability which will be exploited in semantic service discovery. While exact match between services appears to be the best choice, very often, this kind of match can not be found. In such a case, many flavors of relaxed match may also serve the purpose. For example, if we have an ontology that defines "*LCDMonitorPurchaseOrder* is-subclass-of *MonitorPurchaseOrder*", and "*LCDMonitorDeliveryOrder* is-subclass-of *MonitorDeliveryOrder*", we may find a list of Web services that serve our needs.

5.2. Knowledge base update

Next, the searched results will be updated in knowledge base. Suppose we discover two *MonitorSupply* services from company C1 and company C2 respectively which are matched with the faulty service. Based on the ontology designed in Figure 4, the services instances are created correspondently in knowledge base as shown in Figure 7. The *PCmanufacturing* composite service has a faulty service *MonitorSupplyWS* which has two replaceable services *MonitorSupply_C1* and *MonitorSupply_C2*.

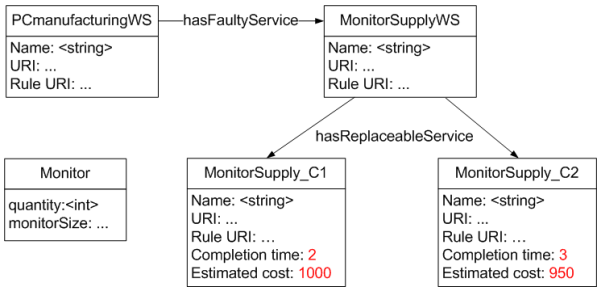


Figure 7. Created instances in knowledge base

5.3. Rule evaluation and rule driven service selection

Each of these replaceable services is associated with local business rules. Suppose the following business rule models the *estimated cost* of *MonitorSupply_C1* service. Using natural English, this local business rule can be described with rule R1 below.

R1. IF (c1) the monitor size is 17 inch; and (c2) the quantity of monitors to be purchased (?q) is less than 500; THEN the estimated service cost incurred is (?qX250) dollars.

The rule R1 contains two conditions c1 and c2. Assume that a client issues a request to this service for purchasing four 17-inch monitors. This request satisfies both conditions c1 and c2. According to R1, if the service call can be finished successfully, the estimated cost will be (4X250) dollars. This result is reflected on the *estimated cost* property of *MonitorSupply_C1* service instance contained in the OWL knowledge base as shown in Figure 7.

One prerequisite for specifying R1 using SWRL is to design a proper group of OWL ontologies. In the context of R1, two ontology concepts can be identified, namely, *Monitor* and *MonitorSupply_C1*. These two concepts are implemented via OWL classes as shown in Figure 7. For the *Monitor* class, it contains two properties: *monitorSize* and *quantity*. Meanwhile, we can create another property, namely *estimated cost*, for the *MonitorSupply_C1* class.

After local business rules evaluation for these two replaceable services *MonitorSupply_C1* and *MonitorSupply_C2*, the results such as the *completion time* and *estimated cost* will be updated in the knowledge base as shown in red in Figure 7.

In order to select one service among the several alternatives, SQWRL is utilized for specifying the selection rules (the global business rules) for alternative service. Using natural English, this global business rule can be described with rule R2 below.

R2. Select the substitution service with minimum completion time for *MonitorSupply* Web service.

Based on the identified OWL classes and properties, Figure 8 shows the SQWRL description for R2. Seven atomic conditions are involved in defining the antecedent of R2. The first three atomic conditions identify the faulty service *MonitorSupplyWS*. The fourth atomic condition selects all the replaceable services for *MonitorSupplyWS*. The fifth, sixth and seventh atomic conditions together identify the properties of the replaceable services. The SQWRL query in the consequent of R2 will return a service with minimum completion time based on the local business rules evaluation results. In this scenario, the *MonitorSupply_C1* service from company C1 will be selected because it takes 2 days to get the monitors which is faster than the service from the company C2.

5.4. Service invocation

Finally, the selected *MonitorSupply_C1* service URL and its input will be fed into *Service Invocation WS* and be invoked to replace the faulty *MonitorSupply* service. The output will be assigned to the output variable of the faulty service which subsequently serves as the input variable of the following Web service.

```
CompositeWS (?cs) ∧ hasFaultyService (?cs, ?fs)
∧ name (?fs, "MonitorSupplyWS")
∧  hasReplaceableService (?fs, ?rs )
∧  completionTime (?rs, ?time)
∧ estimatedCost (?rs, ?cost) ∧ uri (?rs, ?u)
→ sqwrl:select (?rs, ?u, ?time, ?cost )
   ∧ sqwrl:min (?time)
```

Figure 8. SQWRL description for Global Business Rule R2

6. Conclusion

In this paper, we focused on self-healing capability in robust service flow execution. The self-healing mechanism has been proposed to dynamically replace the faulty service with suitable alternatives such that the service flow can be performed smoothly. In order to realize the service substitution, our self-healing mechanism can be divided into four major steps: (1) find a list of alternative services; (2) update knowledge base to record the information on matched services; (3) apply both global and local business rules & update knowledge base & select one service; (4) invoke the selected service. The mechanism has been specifically explored in this paper to support self-healable service flow execution. Its effectiveness has also been demonstrated by a concrete scenario (PC manufacturing application) using a self-healing capable Service Flow Execution System.

7. Acknowledgement

This work was supported in part by the Agency for Science, Technology, and Research (A*STAR) of Singapore SERC TSRP on IMSS Grants 0521160078.

8. References

[1] A. Alves, et al. "Web Services Business Process Execution Language Version 2.0", 2006 Available from: http://www.oasis-open.org/apps/org/workgroup/wsbpel/.

[2] ActiveBPEL3, http://www.activebpel.org, 2004

[3] S. Modafferi, M. Enrico, and P. Barbara, "SH-BPEL: a self-healing plug-in for WS-BPEL engines", in Proceedings of the 1st workshop on Middleware for Service Oriented Computing. 2006, ACM: Melbourne, Australia.

[4] W. Ren, et al. "Semantic Enhanced Rule Driven Workflow Execution in Collaborative Virtual Enterprise", in The 10th International Conference on Control, Automation, Robotics and Vision. Special Session on Integrated Manufacturing & Service Systems (IMSS). ICARCV, December 17-20, 2008. Hanoi, Vietnam.

[5] OWL-S Coalition, "OWL-S: Semantic Markup for Web Services", W3C Member Submission 2004 Available from: http://www.w3.org/Submission/OWL-S/.

[6] I. Horrocks, et al. "SWRL: A Semantic Web Rule Language Combining OWL and RuleML", 2004 Available from: http://www.w3.org/Submission/SWRL/.

[7] G. Chen, et al. "Dynamic Virtual Enterprise Integration via Business Rule Enhanced Semantic Service Composition Framework", 3rd IEEE Conference on Industrial Electronics and Applications (ICIEA 2008), June 3-5, 2008, Singapore.

[8] WS-Diamond. "Web Services - DIAgnosability, Monitoring and Diagnosis". 2005 Available from: http://wsdiamond.di.unito.it/.

[9] K. Verma and A.P. Sheth, "Autonomic Web Processes", in Proceedings of the Third International Conference on Service-Oriented Computing. 2005 Springer-Verlag.

[10] Z. Yang, J. B. Zhang, and C.P. Low. "Towards Dynamic Integration of Collaborative Virtual Enterprise using Semantic Web Services", in The 4th International IEEE Conference on Industrial Informatics. 2006. Singapore.

[11] D. L. McGuinness, F.V.H., "OWL Web Ontology Language Overview", W3C Recommendation 2004 Available from: http://www.w3.org/TR/owl-features/.

[12]Yuhana. "SQWRL", 2007 Available from: http://protege.cim3.net/cgi-bin/wiki.pl?SQWRL.

[13] "Protege Ontology editor and knowledge-base framework", Available from: http://protege.stanford.edu/.

[14] E.F. Hill. "Jess the Rule Engine for the Java Platform", Available from: http://www.jessrules.com/.

[15] S. Liu, R. Khalaf, and F. Curbera. "From DAML-S Processes to BPEL4WS", in Proceedings of the 14th International workshop on Research Issues on Data Engineering: Web Services for E-Commerce and E-Government Applications (RIDE'04). 2004.

[16] G. Chen, et al. "Collaborative Virtual Enterprise Integration via Semantic Web Service Composition", in The 2nd IEEE Conference on Industrial Electronics and Applications. 2007. Harbin, China.