Abstract—Self-healing in workflow is becoming prominent because of the increasing software complexity. In order to support robust workflow execution, in this paper, we propose a self-healing mechanism to replace the faulty service dynamically. It is based on our previous research work—Workflow Execution System (WES). The OWLS2BPEL Mapper in WES is further enhanced to embed the self-healing mechanism into BPEL workflow by exploiting Semantic Web service technology. In order to realize service substitution, business rules is utilized to model the essential business logic behind the service interface (local business rules) and the service selection policy of a composite service (global business rules) in a domain-dependent and semantic-rich manner. A concrete scenario in PC manufacturing Collaborative Virtual Enterprise is presented to test our self-healing capable workflow execution system.

Keywords—self-healing; workflow execution; semantic web service

I. INTRODUCTION

In recent years, the increasing software complexity becomes the most important challenge facing the IT industry [1]. Management of such complexity has been far more beyond the human's capability. As a consequence, the ultimate and possibly only solution is autonomic computing, which is a new idea introduced by IBM. Inspired by the autonomic nervous system of the human body [1], an autonomic computing system would manage the functioning of computer applications and systems by themselves with limited direct human intervention in case of failure, malicious attack, etc. Self-management is the essence of an autonomic computing system which includes following properties: self-configuration, self-optimization, self-healing and self-protection [2].

Although most of the autonomic computing concepts are still premature, several leading IT vendors such as IBM, HP, SUN and Microsoft are carrying out research on autonomic systems. Self-healing concepts are typically part of them. A self-healing system must be able to perceive that the system is not operating correctly and make the necessary corrections to restore itself to normal operation without human intervention. It denotes the system has the ability to detect, diagnose and react to system malfunctions to prevent disruptions. To achieve self-healing, a system must have knowledge about itself and its environment, such as current status of its components, available resources and connections with other systems [1].

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) [3] for modeling business processes within the Service Oriented Architecture (SOA). In order to support robust workflow execution, self-healing in workflow 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 do not support sophisticated recovery actions [4], e.g. service replacement which is a natural way to resort to when a service is not available at run-time, given a large amount of services accessible in service-oriented environment. Thus, we can view the standard recovery mechanisms provided by BPEL language as static self-healing strategies which are defined in the workflow design phase.

In this paper, we define the "self-healing workflow" as a workflow which can be locally recovered from the faults detected during workflow execution by dynamic Web Service replacement. Thus, the static BPEL self-healing strategy defined in design phase is not enough. To this end, we propose a self-healing enhanced OWLS2BPEL Mapper to replace the faulty service dynamically to support robust workflow execution. It embeds the self-healing mechanism into BPEL workflow based on our previous research work [5]—Workflow Execution System (WES).

Dynamic Web service replacement during self-healing procedure relies on service discovery and selection. To provide information for service discovery, we exploit the Semantic Web service technology. The inputs and outputs of a service will be marked with domain ontologies. 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. In addition, each composite service is enhanced with **global business rules** which set the selection policy for alternative service.

We have designed the system architecture which is described in section III. Currently, we are in the process of implementing the prototyping system to examine the effectiveness of our WES with self-healing capability. Detailed experiment results will be reported later.

The remaining part of this paper is organized as follows. Section II provides a literature review of related research works. Section III introduces WES with self-healing capability. The details of self-healing mechanism are presented in Section IV. The self-healing procedure in a PC manufacturing scenario is illustrated in Section V. Finally Section VI concludes the paper.

II. RELATED WORK

As systems increase in complexity, self-healing systems are attracting a number of researcher's attention. In order to give researchers a common basis for defining the scope of self-healing systems, Philip Koopman proposes taxonomy for describing the problem space for self-healing systems. The taxonomy includes four general categories: fault model, system response, system completeness and design context [8]. Debanjan Ghosh presents a literature review in the field of self-healing systems and attempts a classification based on similarities or relationship. He believes the key self-healing process includes maintenance of the system health, detection of system failure and system recovery process [9].

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) [10] is a European research project which aims at developing a framework for self-healing Web Services and devising 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) [11]. 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 general, there are two approaches to achieve self-healing in BPEL workflow. One is standard fault detection and recovery mechanisms defined in the static design phase. The other is realized by dynamic service replacement during workflow execution.

In [4], a self-healing plug-in for Ws-BPEL engine is presented to enhance the ability of a standard engine to provide process-based recovery actions. In addition to the standard Ws-BPEL recovery mechanism, the plug-in also supports pre-processing based Ws-BPEL recovery mechanisms and extended recovery mechanisms. The extended recovery mechanism allows the substitution of selected service which quite similar to ours. However, the execution of this mechanism is hidden to BPEL engine and is managed by a specific management model. Our self-healing mechanism is embedded into BPEL workflow. It is the output of OWLS2BPEL Mapper and its execution still relies on BPEL engine. Furthermore, Semantic Web service technology is exploited in the self-healing process. The alternative service selection is based on user specified policy.

III. SELF-HEALING CAPABLE WORKFLOW EXECUTION SYSTEM

Globalization leads to an efficient new business paradigm generally known as Collaborative Virtual Enterprise (CVE) which demands flexible service orchestration and robust workflow execution. As two major service orchestration strategies, OWL-S [6] and BPEL have their own strength and deficiencies. To meet the challenge in CVE, in our previous research, we proposed a WES that takes advantage of the complementary strengths of these two technologies. On the one hand, having semantic support, OWL-S is used in dynamic service discovery and composition at high level. On the other hand, at the concrete level, industry-based BPEL is exploited in service execution. The description of each Semantic Web Service is enhanced with business rules that model the essential business logic behind the service interface. In order to realize interoperability in OWL-S and BPEL without loss of semantic information, we further proposed an OWLS2BPEL Mapper to facilitate the workflow robustness and support rule evaluation to increase responsiveness to customers.

In this research work, we will propose a self-healing enhanced OWLS2BPEL Mapper to support robust workflow execution. Figure 1 shows the architecture of our WES with self-healing capability. It includes the following components: Service Composer, Global Rule Editor, Self-healing Enhanced OWLS2BPEL Mapper, BPEL Execution Engine, Service Repository and Knowledge Base. We will focus on introduction of self-healing enhanced OWLS2BPEL Mapper.
the details about other components in the architecture can be found in [5].

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. 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).

The inputs and outputs of a service are marked with domain ontologies to facilitate service discovery. To achieve policy 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 Global Rule Editor will provide users an interface to specify the selection policy (global business rules), e.g. one may favor the service which has minimum cost, while the other favors minimum completion time. Therefore, the Global Business Rule plays a determinant role in service selection.

Fault handlers in BPEL are generated by the self-healing enhanced OWLS2BPEL Mapper to catch faults and provide exception handling for the error conditions, along with a catchAll element to house default error handling activities. 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. 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.

After Fault handler catches a fault, the self-healing mechanism will be started. The self-healing mechanism can be divided into the following major steps as shown in Figure 1:

1. Repeat until alternative services are discovered
2. Update knowledge base to record the information on alternative services
3. Apply both global and local business rules & update knowledge base & select one service
4. Invoke the selected service

Definition 1 (Exact-IO-Match):

\[
match_{EO}(A, B) = A_{out} \Leftrightarrow B_{in}
\]

Service A is said EIO-matched with service B if and only if each input of service B has corresponding matched partner in service A’s outputs.

Definition 2 (Input-Match):

\[
match_{I}(A, B) = A_{in} \Leftrightarrow B_{in}
\]

Service A is said I-matched with service B if and only if each input of service B has corresponding matched partner in service A’s inputs.

Definition 3 (Output-Match):

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 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 alternative 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 section will give more details on each step of self-healing mechanism.

IV. SELF-HEALING MECHANISM

A. Service Discovery

There are several service discovery mechanisms. The Universal Description Discovery and Integration (UDDI) [12] is an emerging standard registry for Web services. It defines a way to publish and discover information about Web services. However, it supports only a keyword based search (or exact match) and does not allow capability-based discovery of Web services [13].

Semantic Web service discovery makes up this weakness. 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 [14]. 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 (Exact-IO-Match):

\[
match_{EO}(A, B) = A_{out} \Leftrightarrow B_{in}
\]

Definition 2 (Input-Match):

\[
match_{I}(A, B) = A_{in} \Leftrightarrow B_{in}
\]

Definition 3 (Output-Match):
Service A is said O-matched with service B if and only if each output of service B has corresponding matched partner in service A’s outputs.

Each type of matches as shown above can be further classified in terms of four matching degrees: Exact, Subsume, Relaxed, and Fail which is similar in [15]. 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 repeatUntil activity will be used to repeat searching until at least one alternative service is discovered.

B. Knowledge Base Update

During workflow execution, the knowledge base serves as a semantics container to store all the dynamic semantic information. In order to further select alternative service, the dynamic information about the discovered services needs to be stored in the knowledge base. Therefore the ontology for self-healing procedure should be built in addition to the domain ontology. Ontology is a specification of formally described, machine-readable collection of concepts within a domain and their relationships. The concept of ontologies enables description of explicit semantics. A web standard-based way for representing ontology is the use of Web Ontology Language (OWL) [16]. Built on top of XML, XML Schema, RDF, and RDF schema, OWL adds more vocabulary for describing properties and classes, relations between classes, cardinality, equality, richer typing of properties, characteristics of properties, and enumerated classes. Benefited from many years of research in description logic, OWL has emerged as a web standard language for ontology.

Figure 3 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 after user inputs the global business rules. CompositeWS has one object property hasFaultyService which has a range class AtomicWS. AtomicWS has one object property hasReplaceableService 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 alternative service in the list. The relationship between the CompositeWS and the faulty AtomicWS, the faulty AtomicWS and alternative services are also established accordingly.

C. Rule Evaluation and Policy based Service Selection

This part will present the technical details about how to utilize business rules as a general vehicle to drive the service selection. We will first introduce our business rule enhanced service descriptions. The basic idea is to utilize business rules to model the essential business logic behind every Web Service interface.

To achieve policy based service selection, the description of each atomic Web service is enhanced with local business rules; and the description of each composite service is enhanced with global business rules. Both local and global business rules are identified by Rule URI which is referred by their Semantic Web service description. Each local business rules in the discovered services list will be evaluated first to get their completion time and estimated cost. Global business rules define a set of selection policy to choose one service among a list of alternatives based on the local business rules evaluation results.

In this research work, the Semantic Web Rule Language (SWRL) [7], one of the recommended rule languages in OWL-S specification, is utilized for rule description. SWRL is an emerging standard rooted in the OWL. Through extending the set of OWL axioms, Horn-like SWRL rules can be designed to manipulate OWL classes and properties, and to infer new knowledge from existing OWL knowledge bases. SWRL also provides a range of build-ins to support various mathematical and logical calculations. We have developed a business rule evaluation Web service in our previous work. For more details, please refer to [5].

Semantic Query-Enhanced Web Rule Language (SQWRL; pronounced squirrel) [17] is utilized for the global business rules. They are configurable through the Rule Editor shown in Figure 1. SQWRL is a SWRL-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 SQWRLQuery API 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.

Two open-source projects, Protégé [18] and Jess Rules Engine [19] are exploited in the self-healing process. Upon given an OWL-S service description, APIs offered by Protégé will be first applied to parse the service description in order to identify the corresponding service inputs, service outputs, preconditions, and effects (IOPE). Using SWRL Rule Engine Bridge and SQWRL Query Engine bundled with the Protégé distribution, SWRL rules including both local and global business rules will be further converted to Lisp-like rules suitable for processing and query by the Jess Rules Engine. Evaluating these rules may result in modifications of an OWL knowledge base that serves as the local execution context of the service.

![Figure 3. Ontology for self-healing](image)
D. 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.

V. PC Manufacturing Prototyping System

With new material and technology emerging, computer products upgrade frequently. In order to respond to the dynamic market demand, business workflows for PC manufacturing need to be dynamically formulated and executed. In addition, most of computer manufacturers increasingly concentrate on their core competencies and outsource other functions to third parties, such as product design function and adaptor manufacturing, etc. To fulfill market demand for computers, a timely alliance of enterprises will be formed to share their functionalities and resources in a collaborative manner. As shown in Figure 4, company A, B, C and many others join together to formulate a Collaborative PC manufacturing Virtual Enterprise. Our self-healing capable workflow execution system will be evaluated based on this scenario.

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 MonitorPurchaseOrder 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", we may find a list of Web services that serve our needs.

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 Figure3, the services instances are created correspondently in knowledge base as shown in Figure 5. The PCmanufacturing composite service has a faulty service MonitorSupplyWS which has two replaceable services MonitorSupply_C1 and MonitorSupply_C2.

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 effect is reflected on the estimated cost of that service instance contained in the OWL knowledge base as shown in Figure5. One prerequisite for specifying R1 using SWRL is to design a proper group of OWL ontologies.
in Figure 5. In order to select one service among the several alternatives, SQWRL is utilized for specifying the selection policy (the global business rules) for alternative service. For example, user may favor the service with minimum completion time. Based on the identified OWL classes and properties, Figure 6 shows the SQWRL description for the global business rule in this scenario. Seven atomic conditions are involved in defining the antecedent of the global business rule. They are linked together using the conjunction relation \( \land \). The SQWRL query will return a service with minimum completion time. In this scenario, the MonitorSupply service from company C1 will be selected because it needs only 2 days faster than the service from the company C2.

Finally, the selected MonitorSupply_C1 service will be invoked to replace the faulty MonitorSupply service.

| CompositeWS(cs) \( \land \) hasFaultyService(cs, ?fs) \( \land \) name(?fs, "MonitorSupplyWS") \( \land \) hasReplaceableService(?fs, ?rs) \( \land \) completionTime(?rs, ?time) \( \land \) estimatedCost(?rs, ?cost) \( \land \) uri(?rs, ?u) -> sqwrl:select (?rs, ?u, ?time, ?cost) \( \land \) sqwrl:min (?time) |

Figure 6. SQWRL description for Global Business Rules

VI. CONCLUSION

Self-healing in workflow is becoming prominent because of the increasing software complexity. In order to support robust workflow execution, in this paper, we propose a self-healing mechanism to replace the faulty service dynamically. It is based on our previous research work – Workflow Execution System (WES). The OWLS2BPEL Mapper in WES is further enhanced to embed the self-healing mechanism into BPEL workflow by exploiting Semantic Web service technology. In order to realize service substitution, business rules is utilized to model the essential business logic behind the service interface (local business rules) and the service selection policy of a composite service (global business rules) in a domain-dependent and semantic-rich manner. A concrete scenario in PC manufacturing Collaborative Virtual Enterprise is presented to test our self-healing capable workflow execution system.

REFERENCES


513