Rule-based Dynamic Business Process Modification and Adaptation

Sanghyun Yoo¹, Yo-han Roh², In-Chul Song³, Joo Hyuk Icon⁴, Myoungh Ho Kim⁵, Hak Soo Kim⁶, Jin Hyun Son⁷, Young Sang Paik⁸, Joo Hyun Han⁹, Hyun Ki Jang¹⁰

¹Department of Electrical Engineering and Computer Science, Korea Advanced Institute of Science and Technology, Daejeon, Korea
{shyoo, yhroh, icsong, jhieon, mhkim} @dserver.kaist.ac.kr

²Department of Computer Science and Engineering, Hanyang University, Ansan, Korea
{hsookim, jhson}@hanyang.ac.kr

³Ubiquitous Computing Lab, IBM Korea, Seoul, Korea
{yspaik, hanjh, hkjang} @kr.ibm.com

Abstract— The advances in ubiquitous computing technologies have given rise to the u-Business, where we obtain business-related context information from ubiquitous devices to adapt to the changes of business environments in a timely manner. In these rapidly changing u-Business environments, we need a mechanism to modify business process schemas, which represent a group of business activities arranged in a meaningful way to give values to the u-Business. We also need a way to migrate running process instances to their new business process schemas. In this paper, we propose a rule-driven approach for dynamic business process schema modification and instance adaptation. The proposed approach uses rules to describe schema modification operations to be applied and rule execution conditions. Business environment changes are encoded in these rules, thus timely schema modification is possible. We also propose a system architecture for realizing this rule-driven approach. We have designed the system architecture by utilizing existing technologies so that the system can be easily implemented.

One of the most important types of dynamic business service integration is dynamic business process schema modification and adaptation. Frequent changes in business processes imply frequent modifications of business process schemas. Because stopping and restarting business processes on every schema modification, especially for long-running business processes, can be expensive, adapting running processes to the modified schema is required. Although there has been some work on business process schema modification and adaptation [1][6], no concrete system architecture has been proposed.

Dynamic business process schema modification and adaptation can be achieved by defining a set of rules, which is a common way to adapt to changing business environments. Rules describe interesting business-related events and actions to be taken when the events occur. Here, the conditions for the rules are interesting events, and actions are schema modification operations. By doing so, we can easily modify business schemas and adapt running instances to the modified business schemas, in the face of rapid changes in business environments.

In this paper, we propose a rule-based dynamic schema modification and adaptation method to support dynamic business service integration. This method supports modification of schemas based on a set of user-definable rules and running instances' migration to the modified schemas at run-time.

The remainder of this paper proceeds as follows: We describe our motivation and design goals in Section II and overall system architecture in Section III. In Section IV, we define a language for schema modification, which is called SML4BPEL. After presenting the detailed system architecture for our method in Section V, we give our contributions and analyze the proposed method in various perspectives in Section VI. Finally, we give our concluding remarks in Section VII.
II. MOTIVATION AND DESIGN GOALS

A. Motivation

In rapidly changing business environments, business goals or rules frequently change, resulting in the changes of business process schemas. When a business process schema is modified, one solution to deal with the running instances of the schema is to simply abort them. This solution, however, is not practical for process instances that have been running for a long time because an accumulated amount of work can be lost. Another solution is to let the user decide what should happen. However, this is viable only if the number of process instances to handle is very small. Therefore, a method that can automatically move running instances from old schemas to new ones without any loss of work is required. Such movement is called instance migration.

Several researches have dealt with schema modification and instance migration, but only in a conceptual level, and no concrete system architecture has been proposed. Moreover, to the best of our knowledge, there have been no commercial BPMS or research prototype supporting both schema modification and instance migration.

B. Design Goals

The aim of this paper is to design a system architecture that supports 1) rule-driven dynamic schema modification and 2) dynamic schema adaptation. Rule-driven dynamic schema modification enables automatic business schema modification based on a set of rules that describe interesting business-related events and actions to be taken when faced with those events. Without such approach, the system manager has to manually search and modify business schemas at hand. It mitigates the burden on the schema manager.

Dynamic schema adaptation handles running business instances when their business schema is modified. The previous approaches just abort running instances and restart processes according to the modified schema. However, the approaches are not appropriate when business process instances have been running for a long time or their rollback cost is prohibitive. In that case, running instances should be handled without any loss of work, if possible.

We design the system architecture with two things in mind: ease of use and reuse of existing technologies. Ease of use means that there should be a way to easily search and find schemas. Reuse of existing technologies makes it easy to implement our system because existing technologies can be utilized in the development of the system. It also makes the system more robust because existing technologies have already been tested and used in various environments.

III. OVERALL ARCHITECTURE

The role of the system overall is modification and adaptation. At a high level, the system is designed as two stage pipelines, as shown in the Figure 1.

The system process mainly takes two sets of data records. 'A' is a business process schema originally running on the system. 'Add-on' is SML4BPEL (Schema Modification Language for BPEL) script we have newly defined here, describing a set of rules for modifying the original schema based on ECA (event-condition-action) pattern.

The first stage of the pipeline is the modification. Internally modification consists of two phases - join and evaluate. On the join phase, some matching elements on the original schema (A) are joined and transformed according to the 'Add-on' rules. We consider XSLT-based transformation here. For making the 'Add-on' rules pluggable, we consider XPath expressions, where such an expression would specify what constitutes a match. Consequently, the newly transformed schema is evaluated on the evaluate phase.

The second stage of the pipeline is the adaptation. In this step, each of the joined records obtained from the first stage is applied to the execution engine on the fly. Given a proper policy for the migration, the system decides to apply the modified schema in what state.

Based on the description above, high level architecture for heuristic modification and adaption engine is elaborated in Figure 2. Components and features in the picture are more deeply described on the following sections.

IV. SCHEMA MODIFICATION LANGUAGE

To achieve an automated modification of business schemas, we first define a rule language, SML4BPEL. SML4BPEL is an XML rule language that specifies rules for schema modification. A SML4BPEL rule consists of three sections: conditions, actions. The event section specifies the events that trigger SML4BPEL rule execution. The condition section specifies the conditions that should be met before applying schema modification operations. Finally, the action section specifies how to modify schemas.

SML4BPEL is defined by borrowing various parts of its syntax from different languages. For example, it uses the syntax of BPEL's <eventHandlers> element for event description, existing rule languages' condition expressions for condition description, and the syntax of XSLT for schema
modification operations description. The following shows the basic structure of SML4BPEL:

```xml
<xmlbpel name="ruleName">
  <event>
    <condition ruleLanguage="ruleLanguageName">
      <action>
        <xmlbpel>
          ...</xmlbpel>
      </action>
    </condition>
  </event>
</xmlbpel>
```

The `<xmlbpel>` element is the root element of an SML4BPEL rule. The name of the rule is specified by the name attribute. Under the `<xmlbpel>` element, there are three sub-elements, each corresponding to the event, condition, and action section.

The `<event>` element is the same as the `<eventHandlers>` element in BPEL, except that it has a different element name, `event`. As in BPEL, you can specify two kinds of events: timer events and Web service invocation events. A timer event specifies when the rule is triggered, or after how long. The `<onAlarm>` element is used to specify timer events. A Web service invocation event specifies which Web service invocations trigger the rule. The `<onReceive>` element is used to specify these events. The `<onReceive>` element can capture any kind of events that can be expressed as a Web service invocation.

The `<condition>` element specifies the conditions under which the SML4BPEL engine executes the schema modification operations specified in a SML4BPEL rule. We can use any rule language here by specifying its name in the ruleLanguage attribute.

Finally, the `<action>` element specifies schema modification operations to be executed when the rule is triggered and the conditions for rule execution are met. To specify schema modification operations, the schema designer can put under the `<action>` element as many `<modification>` elements as he wants. The structure of the `<modification>` element is as follows:

```xml
<modification schemaID="schemaID">
  <position>"XPath expression"</position>
  <xslt:stylesheet/>
</modification>
```

The schemaID attribute in the `<modification>` element specifies the identifier of a business process schema to which the schema modification operation is applied. The position attribute designates the point of modification by using an XPath expression. The operation attribute indicates the schema modification operation type.

The following is an example SML4BPEL rule, which states that an Internet shopping mall website will give a special discount to its VIP members if this new discount policy is accepted through a voting process. This rule becomes effective starting from January 1st, 2008.

```xml
<xmlbpel name="VIPDiscount">
  <event>
    <onAlarm><until>2008-01-01</until></onAlarm>
  </event>
</xmlbpel>
```

In this rule, we use the `<until>` element within the `<onAlarm>` element to specify a timer event which is fired on the first day of 2008. In the condition section of the rule, Oracle’s Rule Language (RL) is used to describe that the schema modification of this rule will be executed only when the new policy is accepted through a voting process. In this rule, we have specified only one schema modification operation which modifies business process “orderProcess.1.2” by adding a VIP discount service right before the confirm activity. The XPath expression in the position attribute designates the location of the confirm activity. The name of the schema modification operation is “add before”. Actual schema modification operation is specified within the `<xslt:stylesheet>` element, which contains a set of rules for transforming business process “orderProcess.1.2” into a new business process which embraces the new discount policy.

V. SYSTEM ARCHITECTURE

A. Use-case View

As shown in Figure 3, the use case diagram for the system has two actors. The Schema Designer writes schema modification rules which reflect business requirements. The External Event triggers the execution of the rules which are related to it. The External Event is not a human but an event that occurs outside the system and the initiator of rule execution.

In the Rule Description use case, the Schema Designer describes business requirements as schema modification rules by using a GUI tool. After writing and saving a schema modification rule, the system converts it into an executable form and stores it in the storage.

The Rule Execution use case, performed by the External Event, includes three sub-use cases: Rule Selection, Schema
Modification, and Instance Migration. In the Rule Selection use case, the system finds out the rules related to the external event and retrieves those rules from the storage. In the Schema Modification use case, the system checks for each rule if execution conditions are met, and if so, it executes the rule and modifies schemas accordingly. Finally, in the Instance Migration use case, the system searches for the running instances of the modified schemas and makes those instances migrate to their new schemas.

B. Components

The proposed system consists of eight packages: SML4BPELProcessor, SchemaValidator, SchemaVersionManager, InstanceMigrator, DataManager, BPELEngine, RuleEngine, and XSLTEngine. In this subsection, we explain only the first five packages because the last three packages are external ones, where we use existing technologies.

1) SML4BPELProcessor. SML4BPELProcessor compiles SML4BPEL rules and executes them. This package consists of three modules: SML4BPELEditor, SML4BPELCompiler, and SML4BPELEngine. SML4BPELEditor is a GUI tool that allows the user to easily describe SML4BPEL rules. This module can be developed by utilizing existing GUI tools for WS-BPEL 2.0 event description, rule language description, XSLT rule description, and so on. SML4BPELCompiler builds schema modification plans by compiling the described rules. The compilation steps include rule parsing, syntax verification, plan creation, and plan optimization. SML4BPELEngine modifies business process schemas by executing schema modification plans. When the specific event occurs, this module finds schema modification plans related to this event and executes them if their execution conditions hold. After modifying schemas, it verifies the modified schemas by using the SchemaValidator package and initiates the migration of running instances with the InstanceMigrator package.

2) SchemaValidator. The SchemaValidator package verifies modified schemas by applying SML4BPEL rules in terms of the syntax and semantics of the WS-BPEL 2.0 specification [15]. This package includes only one module, SchemaValidator. The WS-BPEL 2.0 standard document provides some criteria for the syntax and semantics checking. The SchemaValidator module checks if modified schemas conform to these criteria. Validation checking tools against the WS-BPEL 2.0 standard can be used to develop this module.

3) SchemaVersionManager. The SchemaVersionManager package maintains various versions of modified schemas as well as existing ones in the form of the schema version tree. This package includes one module, SchemaVersionManager. This module creates a version ID of a new (modified) schema and inserts the ID into the schema version tree. A node in the tree represents a schema, and a link between two nodes represents modification operations applied to the source schema. This information will be used to determine instance migration policies. Many of existing technologies such as [6] can be used to manage the schema version tree.

4) InstanceMigrator. InstanceMigrator is a package that enables running instances to migrate to appropriate schemas. There are two sub-modules in this package: MigrationPolicySelector and MigrationExecutor. MigrationPolicySelector determines migration policies for running instances whose schemas are modified. Since the instance migration is not possible for every instance, it is necessary to check if running instances can migrate to new schemas. If so, they migrate to their new schemas. Otherwise, they should be treated according to some policies such as transfer, rollback, detach, abort, and proceed [1][13]. After the policies are determined, MigrationExecutor coordinates instance migration according to the policies by making a request to the BPEL engine.

5) DataManager: DataManager is a package that stores, manages, and retrieves data used in the system. The RDBMS and the BPEL engine can be data sources.

C. Sequence Diagram

In this subsection, the sequence diagrams are provided to show communication between modules. The modules colored by dark grey mean that they are external modules.

After a rule is designed with SML4BPELEditor, SML4BPELCompiler compiles this rule and generates a rule execution plan. This plan is stored in the storage through DataManager. (Figure 4-(a)) When a business event arrives, SML4BPELEngine requests DataManager to find the relevant rules to the event. (Figure 4-(b))

By using RuleEngine, SML4BPELEngine checks for each rule if the conditions in the rule hold. If so, it retrieves the related business process schema from BPELEngine so that XSLTEngine can transform the schema according to the modification operations written in the action part of the rule. After SchemaValidator checks whether the modified schema is valid, SchemaVersionManager adds it to the version tree.

Fig. 4 Sequence diagram: (a) rule definition and (b) rule selection

Fig. 5 Sequence diagram: schema modification
A. Contribution

Finally, when a business process schema is changed, SML4BPELEngine notifies MigrationExecutor to find the running instances influenced by the schema modification. Then, MigrationExecutor performs instance migration by sending a migration request to BPELEngine according to the policy determined by MigrationPolicySelector. (Figure 6).

VI. Analysis

A. Contribution

The rule-based approach we propose in this paper has many advantages. First, the rule-based approach uses rules in modifying schemas and instance's migrating to the modified schemas. It can easily adapt to the changes in business environments by changing the related rules. Rapidly changing business environments are encoded into a set of rules. This allows business processes to easily adapt to constantly changing business environments and reduces the adaptation cost.

Second, the approach has been designed to utilize existing techniques as much as possible, which reduces development time and makes it possible to develop a reliable system. For example, we utilize the event handler syntax from WS-BPEL 2.0 for event description, existing rule languages for condition description, and XSLT for describing schema modification operations. Thus, we can develop an engine for our system by using available engines for WS-BPEL 2.0, rule languages, or XSLT.

Third, the proposed approach supports migration of running instances after schemas have been changed. This so called the hot-fix technique avoids stopping running instances, which results in the reduction of cost caused by stopping long-running instances.

B. Discussion

The rule-based dynamic schema modification technique makes the best use of existing techniques such as WS-BPEL 2.0, rule languages, and XSLT. The use of existing languages and their engines allows the ease of development. However, the use of XSLT as a schema modification language forces rule designers to know the contents and structures of target schemas. Thus, it is needed to provide a more high-level language so that even business people can describe what they want as rules. We intend to design such language as future work.

VII. Conclusions

In this paper, we have presented a rule-driven approach for dynamic business schema modification and instance adaptation. This approach uses rules that contain schema modification operations and conditions in which those operations are performed, and thus enables rapid schema modification when business environments change. We have also proposed concrete system architecture by making best use of existing technologies and components.

VIII. Acknowledgement

This work was supported by the IT R&D program of MIC/IITA [2005-S-603-02, Core technology development for Ubiquitous Collaboration System]

This work was supported by the Korea Science and Engineering Foundation(KOSEF) grant funded by the Korea government(MOST) (No. R01-2007-000-20135-0)

REFERENCES