Knowledge-Based AOP Framework for Business Rule Aspects in Business Process

Chankyu Park, Ho-Jin Choi, Danhyung Lee, Sungwon Kang,
Hyun-Kyu Cho, and Joo-Chan Sohn

In recent years, numerous studies have identified and explored issues related to web-service-oriented business process specifications, such as business process execution language (BPEL). In particular, business rules are an important cross-cutting concern that should be distinguished from business process instances. In this paper, we present a rule-based aspect oriented programming (AOP) framework where business rule aspects contained in business processes can be effectively separated and executed. This is achieved by using a mechanism of the business rule itself at the business rule engine instead of using existing programming language-based AOP technologies. Through some illustrative examples, this work also introduces a method by which business rule aspects, separated through an external rule engine, can be represented and evaluated. We also demonstrate how they can be dynamically woven and executed by providing an implementation example which uses two open-source-based products, the Mandarax rules engine and Bexee BPEL engine.

Keywords: AOP, business rules, business process, BPEL.

Manuscript received May 26, 2006; revised May 16, 2007.

Chankyu Park (phone: + 82 42 860 6708, email: parkck@etri.re.kr), Hyun-Kyu Cho (email: hkcho@etri.re.kr), and Joo-Chan Sohn (email: jcsohn@etri.re.kr) are with Intelligent Robot Research Division, ETRI, Daejeon, S. Korea.

Ho-Jin Choi (email: hjchoi@icu.ac.kr), Danhyung Lee (email: danlee@icu.ac.kr), and Sungwon Kang (email: kangsw@icu.ac.kr) are with the Department of Software Engineering, Information and Communications University, Daejeon, S. Korea.

I. Introduction

The standards of business process execution language (BPEL), Web service choreography interface (WSCI), and business process modeling language (BPML) address common application requirements in an open, portable, and standard manner [1]-[3]. These languages define a business process which determines the logical dependencies between the composed web services. The process specifies the control flow of invocations and methods for data transfer between them.

If an organization is facing the challenge of attaining greater agility in its business processes, ideally it would like to quickly respond to competition and changes in regulations and change the behavior of its processes without modifying or redeploying the business process. It should also consider using business rules as part of its process architecture [4], [5].

One of the drawbacks of process-oriented languages is a lack of adaptability, in that the composition is predefined, static, and does not evolve because there is no support for dynamic process change in their specifications. The only method to accept change is to sequentially stop the currently running process, modify the process definition itself, and then redeploy the changed business process to a business process execution engine [4].

Several previous studies related to these problems have been carried out in two areas, objected-oriented language level and extension of business processes specifications [4], [5]. These works are based on an aspect-oriented programming (AOP) framework. The implementation of business rules tends to cut across several activities of a process definition. Using AOP frameworks provides a means to modularize cross-cutting concerns. They are known to be valuable for modularity and

flexibility [4].

A common concept underlies recent studies. The core composition specification only defines the basic control and data flow between the services to be composed using process-based approaches, whereas business rules which are subject to change are modularized in separated units, such as concrete rules.

To concretize these ideas, concrete business rules are implemented as aspects by using AOP language-based object-oriented languages, such as AspectJ or JAsCo [6], [7]. Some studies investigate how concepts from the business rule world relate to AOP concepts. They describe the ways in which business rules can be implemented by using aspect-oriented extensions of BPEL, such as AO4BPEL (aspect oriented extension for BPEL) [8]. However, they do not sufficiently support the concepts of business rules since their definitions of business rules are implemented using AOP languages instead of business rule languages. Thus, these studies fail to fully provide functionalities of business rules or a proper integration example between a business rule engine (BRE) and a process execution engine which can execute a BPEL process [8].

In order to address these problems, we propose a rule-based AOP framework to employ business rules for actual aspects without using the existing AOP technologies. The framework also outlines a method to integrate rule-based knowledge, accessible through a BRE, with a process orchestration engine compatible with BPEL. To this end, we use a rule join point model (RJPM), which can support AOP concepts in terms of pointcuts and actions, and we extend the process orchestration engine in terms of dynamic weaving and aspect awareness.

The remainder of the paper is organized as follows. Section II provides a short overview of background related to AOP and business rules. Section III discusses problems and limitations of current approaches. Section IV details the proposed approach and describes its implementation. Section V concludes the paper.

II. Background

A business rule has been defined by the Business Rules Group as a statement that defines some aspect of a business [9]. A significant characteristic of business rules is that they have a tendency to change whenever business policies change, which is more often than the core application functionality changes [9], [10]. When business policies are more complex, it is helpful to explicitly extract business rules from them in business processes.

For example, the business rule: "If the number of transactions of a customer is more than 10 in the year 2005, then his/her grade is 'VIP'" and "If the grade of a customer is VIP, then the customer receives a 20% discount coupon" is a kind of logical inference. In order to evaluate whether the grade of a customer is VIP, we need to evaluate the other rules and perform additional computations, for example, by querying customer information from a database. Additionally, if the customer qualifies as a VIP, the next rule should be chained and evaluated. After the two rules are evaluated, the customer can actually receive the discount coupon. The functionalities of the business rules previously discussed are not specified in BPEL. Any activity related to control and decision in BPEL can be a candidate for a business rule and can be separated from core process definition. In real business, these control activities actually have the characteristics of a business rule, and the same patterns or business rules repeatedly occur in a BPEL process. We can regard these rules or patterns as cross-cutting concerns.

III. Problem Statement

In terms of AOP, business rules are typical examples of cross-cutting concerns [5]. Therefore, it is important to separate business rules from the core business processes, and to trace business rules to business policies and decisions. On the other hand, in terms of knowledge base management, business rules are pieces of knowledge about the business. As such, it is not appropriate to bury that knowledge deep in code where no one can identify it [11].

To date, only simple business rules, which do not support inferences and other inherent functionalities of rules, can be represented using basic BPEL elements. However, these can be scattered and tangled if business logics become complex, since the current BPEL specification does not support such AOP concepts [8]. Some studies have found that the modularity and adaptability of process definitions can be enhanced defining a separate rule aspect as a rule [8]. However, such a rule aspect definition as BPEL has a limitation in terms of describing functionalities of business rules. In order to attain the full functionalities of business rules, it is necessary to consider how to integrate a BRE. Currently, there is no standard method to integrate rules with a BRE. Although a BPEL engine can call an invoke activity, which includes business rule services as a web service, the core business process mechanism cannot be directly coupled to the business rule mechanism.

When urgent changes are needed in a running BPEL process, the BPEL engine should be stopped, its definition should be changed, and the changed process should then be restarted. This sequence of actions is necessary because dynamic adaptability related to the change is not supported in the current implementation of the BPEL engine. However, if urgent changes are related to business rules or logics and they are

already separated from the core business process, a core running BPEL process does not have to be stopped.

Occasionally, BPEL process definition can be opened for the sharing of process definitions or communication between other web-service providers. However, in terms of security, an enterprise will generally avoid revealing its business policies, as they are considered confidential. However, if business policies are not separated in the core process and they are coupled to the business process itself, private and confidential content may be revealed in the public domain.

IV. Our Approach

In order to adapt changes of business rules dynamically in runtime, we present a rule-based AOP approach that describes a method to define business rules that can adapt to changes in the business process as a factor of aspects in the AOP paradigm. We also present a way to integrate a BRE with the BPEL process. The following items comprise the contributions of the proposed framework.

First, in order to fully use the functionalities of business rules, we use a typical BRE. To do so, integration with a BPEL engine should be accomplished. We will describe how the tasks comprising integration are carried out. Business rule aspects contained in business processes can be separated and executed by using a BRE instead of existing programming-language-based AOP technologies or a BPEL engine itself. Furthermore, this separation increases adaptability, because the core process of each separate unit can evolve independently, thus reducing the complexity of the entire business process. When business policies or environments change, users only have to modify the corresponding modules that implement the affected business rules.

Second, in order to represent business rules, we use a basic syntax of business rules such as an if-then statement instead of BPEL standard syntax. An if-then rule statement is more intuitive and easy to read than BPEL standard syntax or other programming languages. In this paper, we use a specific term, process rule aspect (PRA), as a separate business rule aspect. This term is employed because the scope of the BPEL process definition is related to the composition of web services and business rules are mainly confined to a business process.

Third, in order to satisfy AOP concepts, we propose an RJPM capable of supporting AOP concepts without using existing AOP technologies. Also, we present a method to implement an extension of a BPEL execution engine to integrate a BRE in terms of dynamic weaving and aspect awareness to achieve an RJPM. The extended BPEL engine can retrieve the needed information from the BRE and connect business rules to core process events.

In the remainder of this section, we describe our rule-based AOP in detail. Also, several examples of PRAs, a case study of implementation, and an experiment to verify the performance of the proposed concepts are presented [12], [13].

1. Rule-Based AOP in Business Process

One of the key concepts in this paper is that a separated core BPEL process should be executed by a BPEL process engine and also separated PRAs should be evaluated by a business rules engine.

However, the main authority and location of a BRE control belongs to the BPEL execution engine [8]. Figure 1(a) shows a TravelPackageProcess mixed with a business rule and Fig. 1(b) shows a separated process, that is, a core business process and PRA, which plays the role of a business rule.

Commonly, a PRA is used within the business process, such as "if no flight is found for the dates given in the client request, do not search for accommodation" [8]. This rule statement is written in natural language and does not serve as an executable rule which can be managed in a BRE. Figure 1(a), shows that three web services are involved in the PRA: the airline web service, the hotel web service, and the composition itself. Whenever business policies change, the process (a) should be modified. However, in case (b), only a separate PRA has to be modified or new ones are added. This enables the BPEL process to adapt to a change without stopping the runtime

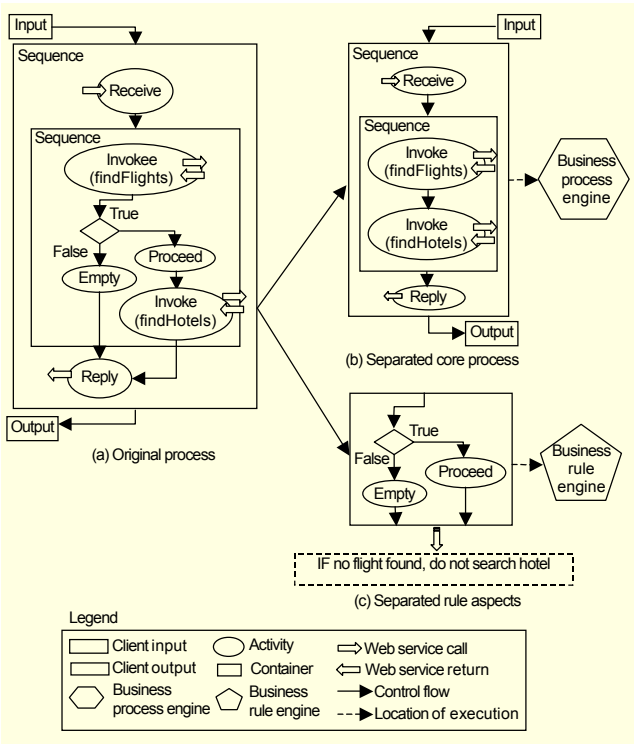


Fig. 1. Concept of separating business rules from an existing business process: TravelPackage.

business process.

Although a <switch> activity which branches to the hotel activity can be added in order to represent the PRA, the implementation of PRAs becomes more difficult if the condition statement of the rule requires some logical inference.

In our rule-based AOP framework (RBAOP), the interaction of an aspect with the core BPEL process is defined by the RJPM. Although we use terms and basic concepts of the RJPM suggested in [14], we newly define a unique RJPM which interacts with a BPEL process. An RJPM has at least three points of similarity with the general AOP concept. First, they are similar regarding the points at which the aspect can apply, often called join points in AOP. The join points in this RJPM are well-defined points along the execution of a business process. They can include basic activities, such as invoke, reply, assign, and so on. Second, they use similar entities to specify multiple join points. These are often called pointcuts in AOP. In this RJPM, pointcuts are a query over all the join points of a business process to select a small subset of join points. Pointcuts are represented using XPath expressions [15] and implemented using a Mandarax rule query mechanism [16]. Third they use similar means to affect behavior at the join points. In AOP, this is often called advice. In our RBAOP, advice depends on connecting PRAs to core application events which depend on run-time properties. Moreover, advice enables a BRE to retrieve needed information and make it available in those events that take place when the rules are applied.

Figure 2 describes the basic architecture of RBAOP, including the RJPM outlined above. First, the BPEL engine sends a query to the BRE when a certain event occurs. The inference result of a BRE is then returned to the action enabler of the BPEL engine. Finally, the action enabler identifies a type of action and executes the action and/or advice.

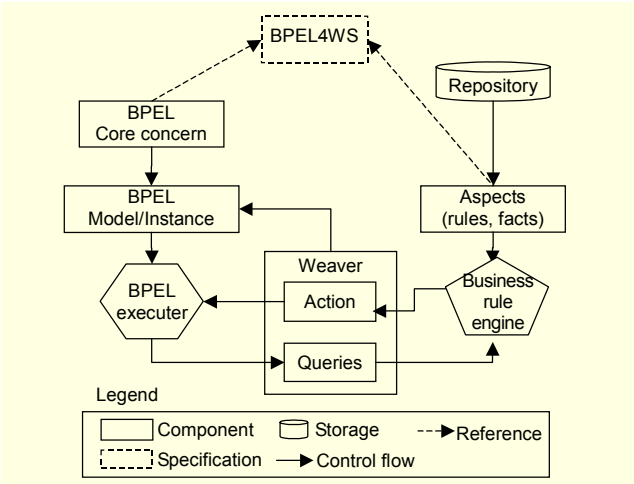


Fig. 2. Basic architecture of rule-based AOP.

In the following sections, we explain how PRAs in BPEL can be represented as business rules used by a BRE, how PRAs abide by AOP concepts, and, finally, how the rule-based AOP mechanism can be realized by integrating a Mandarax rule engine with a Bexee BPEL execution engine [13].

2. Representing PRAs to Mandarax Rules

To satisfy basic AOP characteristics, when a BPEL engine executes a specific action caused by PRAs, the BPEL engine must be aware of an event in which pointcuts are triggered. However, a Mandarax system does not support the sort of event condition action (ECA) mechanism [11], [12] that is used in event-based rule execution. An ECA requires a forward-chaining rule-based system, whereas Mandarax is a backward-chaining engine [16].

We propose that it is possible to implement an intrinsic event-action mechanism making an ActionP predicate, which has a mediator role interconnecting actual BPEL activities to the action in the Mandarax rule without modifying the Mandarax system itself. Figure 3 describes the concept of RJPM in detail.

As shown in Fig. 3, a PRA is composed of two prerequisites and one conclusion. The left side of the two prerequisites stands for actual PRAs in BPEL. This part can also include multiple prerequisites to cover complex business policies. The right side of the two prerequisites stands for how Mandarax recognizes a join point in BPEL and a basic part of RJPM. If both prerequisites must be true, actual PRAs are applied to the join point in BPEL. The conclusion predicate ActionP associates three terms: joinPoint, pointcut, and action. The purpose of this predicate is to execute actions in order, either before or after a pointCut is indicated at a joinPoint in the BPEL process. A query represents a predicate that must exist in rule prerequisites [16]. When the BPEL engine traverses each BPEL activity, it can query whether an action can be executed by substituting a joinPoint for the location information of an actual BPEL activity.

One instance of an RJPM is a PRA, and all constituents of a

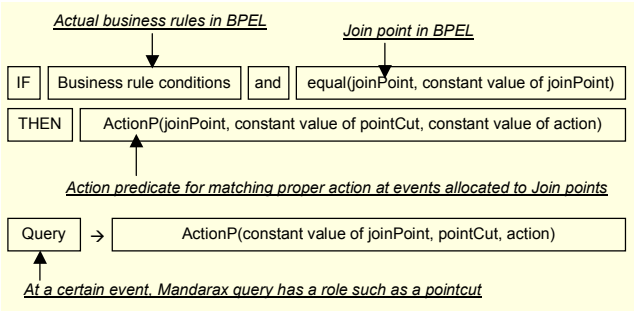


Fig. 3. Structure of the RJPM as Mandarax rules.

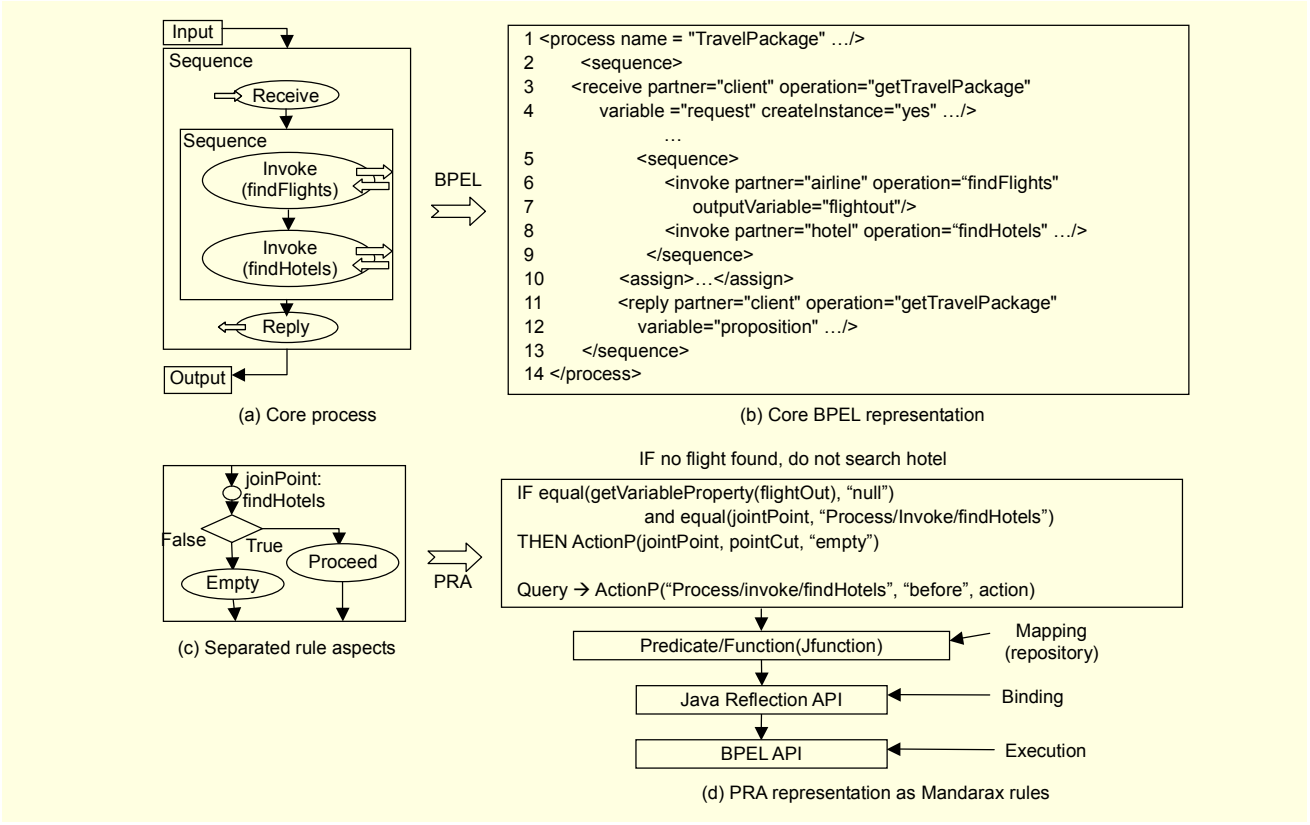


Fig. 4. Revisiting the example given in Fig. 1

PRA should satisfy the syntax and structure of the RJPM. A PRA is managed separately by a BRE. It plays the role of a knowledge base for the Mandarax rule engine, and can be represented by Mandarax rule language.

Figure 4 shows that the PRA introduced in Fig. 2 can be converted into an example of a PRA. The joinPoint is expressed using an XPath expression for querying the pointCut, and getVariableProperty(flightOut) is a BPEL API function which retrieves the values of BPEL variables and makes them available to those events that take place when the PRAs are applied [1], [13].

The proprietary BPEL API can be used by wrapping it with a JFunction, which enables a Java function to execute in a PRA because the Mandarax system has a mechanism of execution using a Java reflection API. For instance, if the evaluation result of this example PRA is true, an invoke activity called findHotels is not executed and the process is ended.

3. Other Examples

In this section, we introduce two other examples to support the rule-based AOP framework. There are typically four kinds of business rules [9]. However, we focus on the three kinds of business rules most closely related to dynamic behavior.

The basic rule of a rule-based AOP is an action enabler rule which checks conditions and, upon finding them true, initiates the appropriate action. The rule explained in the previous section is a good example of an action enabler. Also, two examples introduced in this section contain many action enabler rules. The following sub-section explains how a computation rule and an inference rule are applied to PRAs in the BPEL process.

A. Example of a Computation Rule

In the process shown in Fig. 5, more action enabler rules are used than in that shown in Fig. 4. Additionally, a computation rule which checks a condition and, when the result is true, provides an algorithm to calculate the value of a term, is used in the process in Fig. 5. This example shows that PRAs enable various business constraints to be separated.

In comparison with the example in Fig. 4, R1, R2, R4, Query1, and Query4 are added to complement business policies: whether a customer is valid, whether flights are available, whether hotels are available, and, finally, computation of the returned prices. These PRAs can be evaluated through a BPEL engine which issues a query to a Mandarax inference engine when a BPEL activity which has arrived at the joinPoint is assigned to a Query. If the result of the inference is false at each query, this process

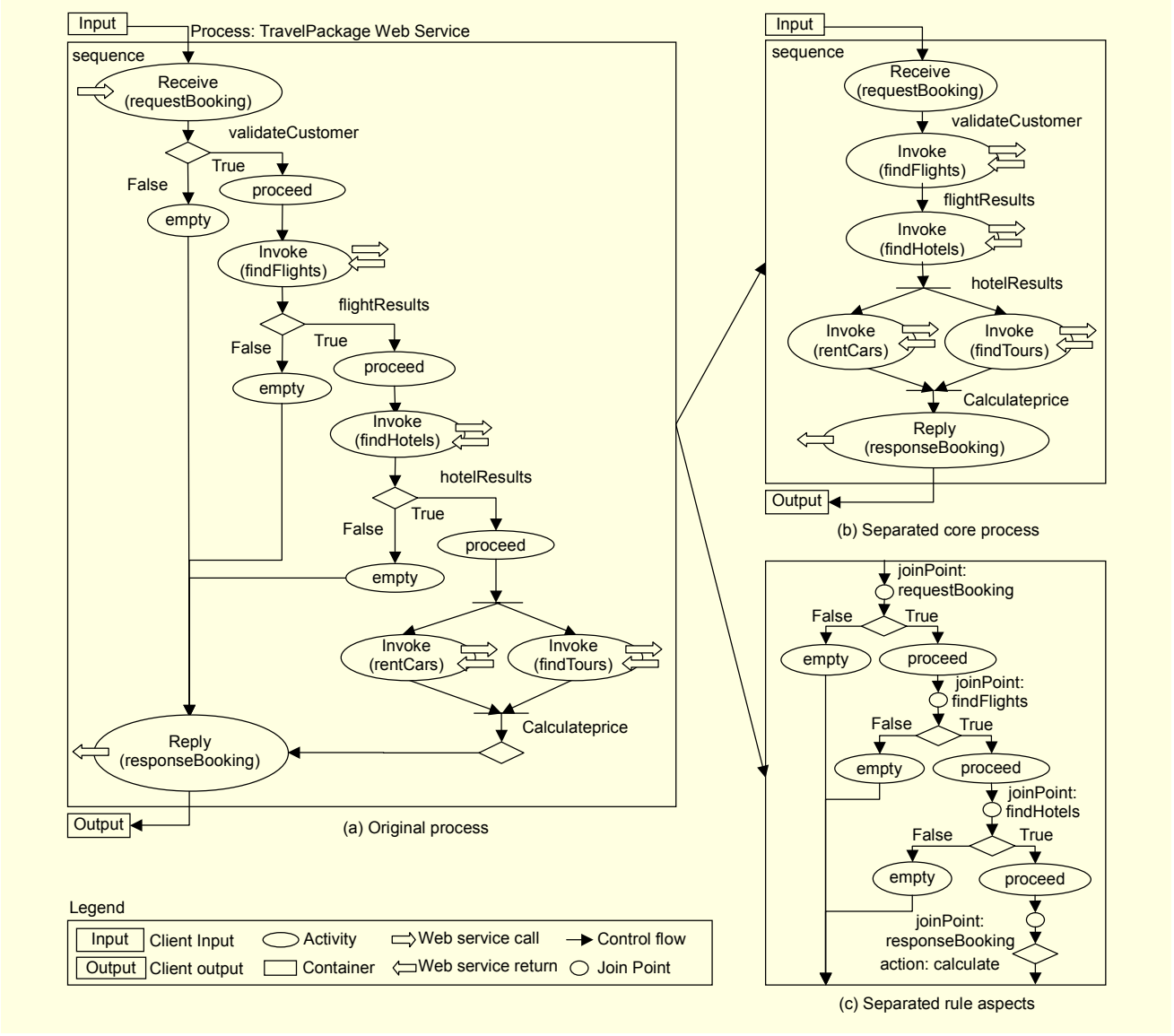


Fig. 5. Example of a computation rule.

stops all subsequent processes and sends a failure result to the client program.

As shown in Figs. 5 and 6, "empty" and "proceed" activities can each occur three times in the original process. We can consider empty or proceed activities as a kind of security check service which plays the role of a typical cross-cutting concern. Typical AOP technologies define the security check aspect and the three join points separately, whereas the present RBAOP approach does not separately define aspects and join points. A PRA structure includes both aspects and join points.

B. Example of an Inference Rule

The example in Fig. 7 appears very complex and contains all three kinds of business rules as well as a fact. The most important among them are inference rules, which test conditions and, upon finding them true, establish a new fact. In a Mandarax system, a collection of these rules, facts, and queries is called a knowledge base. This knowledge base, as a PRA, shows that discount policies depend on a customer's grade in the runtime process. The PRAs from R4 to R11 and Fact1 are added to the existing example in Fig. 5(c) to complement the discount policies. It is possible to express action enabler rules in the current BPEL process definition; however, additional inference rules cannot be explicitly created in the definition because these rules are needed to induce new facts. Thus, these implicit inference rules have to be managed by an independent inference system.

For example, when Query3 is issued to the Mandarax engine, chaining occurs in backward order from Query3→ R10→

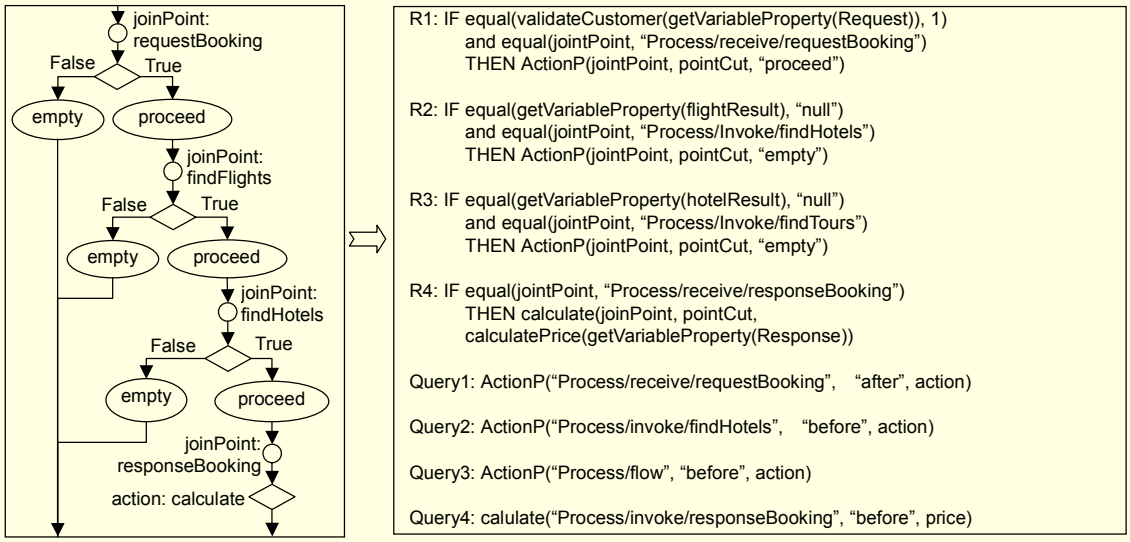


Fig. 6. Rule representation of Fig. 5(c).

```
R1: IF equal(validateCustomer(getVariableProperty(Request)), 1)
    and equal(jointPoint, “Process/receive/requestBooking”)
    THEN Action(jointPoint, “after”, “proceed”)
R2: IF equal(getVariableProperty(flightResult), “null”)
    and equal(jointPoint, “Process/Invoke/findHotels”)
    THEN Action(jointPoint, “before”, “empty”)
R3: IF equal(getVariableProperty(hotelResult), “null”)
    and equal(jointPoint, “Process/Invoke/findHotels”)
    THEN Action(jointPoint, “after”, “empty”)
R4: IF transaction(getVariableProperty(customer), 2005) > 10
    THEN grade(getVariableProperty(customer), “VIP”)
R5: IF grade(getVariableProperty(customer), “VIP”)
    THEN coupon(getVariableProperty(customer), “20%”)
R6: IF TotalAmount(getVariableProperty(customer), 2005) < 100
    THEN grade(getVariableProperty(customer), “Gold”)
R7: IF grade(getVariableProperty(customer), “Gold”)
    THEN coupon(getVariableProperty(customer), “10%”)
R8: IF grade(getVariableProperty(customer), “Standard”)
    THEN coupon(getVariableProperty(customer), “0%”)
Fact1: grade(getVariableProperty(customer), “Standard”)
R9: IF coupon(getVariableProperty(customer), “20%”)
    and equal(jointPoint, “Process/receive/responseBooking”)
    THEN calcuate(joinPoint, “before”, calculatePrice(getVariableProperty
    (Response), “20%”)
R10: IF coupon(getVariableProperty(customer), “10%”)
     and equal(jointPoint, “Process/receive/responseBooking”)
     THEN calcuate(joinPoint, “before”, calculatePrice(getVariableProperty
    (Response), “10%”)
R11: IF coupon(getVariableProperty(customer), “0%”)
     and equal(jointPoint, “Process/receive/responseBooking”)
     THEN calcuate(joinPoint, “before”, calculatePrice(getVariableProperty
    (Response), “0%”)
Query1: ActionP(“Process/receive/requestBooking”, “after”, action)
Query2: ActionP(“Process/invoke/findHotels”, “before”, action)
Query3: ActionP(“Process/invoke/findHotels”, “after”, action)
Query4: calulate(“Process/invoke/responseBooking”, “before”, price)
```

Fig. 7. PRA example of an inference rule.

R7→ R6[16]. Finally, the customer receives a discount of 10% on travel packages. If discount policies change because of a promotion program, the user can simply modify the PRAs without modifying the whole process or stopping operation.

4. Implementation of a Rule-Based AOP

Integrating a rule-based system into a web service environment is a complex task because both systems have their own paradigms [8]. However, it is reasonable to integrate both systems using a more standardized API, or an open-source-based API related to them, because rule-based systems play a significant role in our rule-based AOP.

To evaluate whether our rule-based AOP framework is feasible, we focus on an action enabler, which identifies proper actions, and a dynamic weaver which executes advice at a joinPoint. Because a standardized API does not contain specifications for access to a BPEL engine, most commercial vendors have either no interface, or a proprietary interface. This can make connecting a BRE with a BPEL engine difficult.

However, the BPEL engine, Bexee, an open source project, provides a better architecture than other products, accommodating an extension of itself and interoperability to different systems [17]. Consequently, the Bexee engine can communicate directly with the BRE through its proprietary software. The PRAs can access Bexee APIs in their statements because Mandarax provides an access mechanism using Java reflection [16]. In addition, user-defined objects and methods that perform specific tasks or computations can be used in their statements.

Bexee consists of four major parts: a core engine, factory, dispatcher, and process controller. As shown in Fig. 8, the core engine comprises all the business logic pertaining to how to

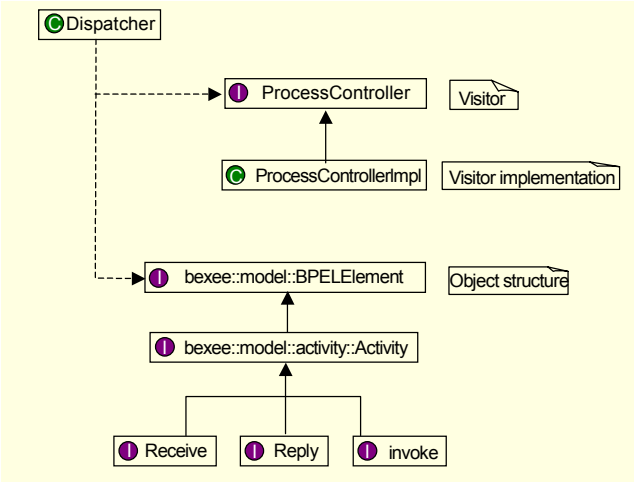


Fig. 8. ProcessController with a visitor pattern.

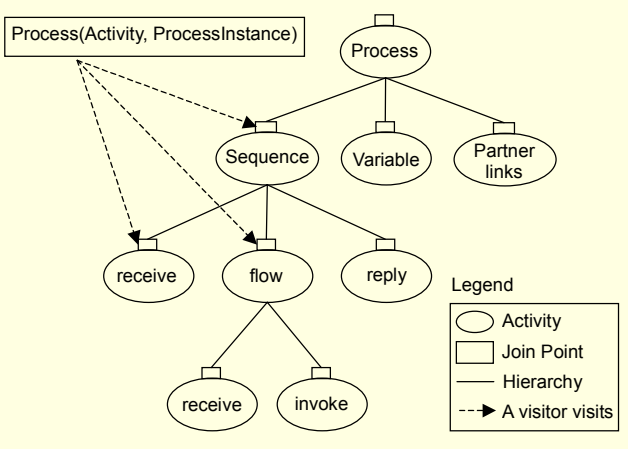


Fig. 9. Joinpoint access in process hierarchy.

process a given BPEL document. This includes BPELProcessFactory to create a BPEL process given a BPEL document, ProcessController to process a BPEL process, and a dispatcher to look up deployed BPEL processes and execute instances, as well as a number of other components.

The main task of ProcessController is to receive messages for business processes and to execute them against a business process. Each activity has its own assigned method for processing. Enhancing Bexee with capabilities for processing new activities corresponds to implementing the right method within ProcessController. The processing of activities which contain other activities consists of iterating over the contained activities and calling their accept() method. For the proposed framework, it is necessary to enhance ProcessController to support rule-based AOP concepts. Its detail implementations are described as a form of pseudo Java code.

A. Aspect Awareness in Bexee

Every different BPEL activity of the process is treated uniformly as an activity. This uniform manner of processing activities allows the process model to be used by other tools, such as the AOP paradigm [17].

Figure 9 shows the points at which custom treatments can be called during processing. When process (Activity, ProcessInstance) is executed at the entry point, the point can be a joinPoint. At this point, a Mandarax query can be executed by an inference engine. According to the inference result, the proper action can be processed by Bexee.

Figure 10 shows a list of generateKnowledgeBase() and illustrates how the knowledge base can be loaded to ProcessController. Thus far, all PRAs illustrated in this paper can be serialized into XML format, such as the XKB Mandarax format [16]. Line 21 in Fig. 10 is an invocation of

```
01   private KnowledgeBasePlus generatedKnowledgeBase(String name) {
02         XKBDriver_2_1 xkbMgr = new XKBDriver_2_1();

03         String fileName = getFileNameFromProjectName(name);
04         File file = new File(fileName);
05         InputStream in;
         Thread.currentThread().setContextClassLoader(getClass().getClass
         Loader());

06         try {
07               in = new FileInputStream(file);
08               KnowledgeBasePlus result = xkbMgr.importKB(in);
09               in.close();
10               return result;
11         }catch (XKBException e) {
12               System.err.println("Cannot import knowledge base!");
13               e.printStackTrace();
14         }
15         return null;
16   }

     // process the Process
17   public void process(Process process, ProcessInstance instance)
18               throws Exception {

19         log.info("Processing a Process");

           // process all child elements
20       .....
           // Knowledgebase is loaded into memory
21         knowledgeBase = generateKnowledgeBase("TravelPackage");

22       ....
23   }
```

Fig. 10. Pseudo codes loading and initializing knowledge from XKB file.

this method when ProcessController visits a root node process in BPEL. The knowledge base of the process should be initialized, but ProcessController can share the same knowledge base in multiple process instances because ProcessController has a context that can have multiple process instances.

As shown in Fig. 11, the dynamic weaver injects the advice presented in PRAs into the specified join-points associated with each advice listed as pseudo codes. Line 7 assigns the current joinPoint value as an XPath expression to match it with

```
    // one of examples implementing dynamic weaving in invoke activity
01  public String process(Invoke invoke, ProcessInstance instance)
02          throws Exception {
03      log(invoke);
04      String pointCut, action;
        // initialize objects
05      ProcessContext ctx = instance.getContext();
06      BPELProcess process = instance.getProcess();
        // current position is converted to joinPoint XPathExpression
07      String jointPoint = XPathConvert(invoke);
        LogicFactorySupport lfs = new LogicFactorySupport();
08      InferenceEngine ie = new ResolutionInferenceEngine();
09      // prepareing query and then execute query. Finally execute action
        // accroding to the result of inference
10      for (Iterator iter = KnowledgeBase.queries();iter.hasNext();) {
11          Query query = (Query)iter.next();
12          Fact[] fact = query.getFacts();
13          String joinPointFromQuery[] = fact[0].getPredicate().getSlotNames();
14          if(joinPointFromQuery[0].equals(joinPoint)) {
                // In case that pointcut identifier is "before"
15              if(joinPointFromQuery[1].equals("before") {
16                  ResultSet result = ie.query(query, knowledgeBase, ie.ONE,
                                        ie.TRY_NEXT);
17                  while(result.next()) {
18                      if(qeury.getName().equals("ActionP") {
19                          Activity action = (Activity)result.getResult(Activity.class,
                                "action");
20                          if (action instanceof empty) return "empty";
21                          elseif (action instanceof proceed) processInvoke(invoke,
                                    instance);
22                          else
23                              return "empty";
24                      }
                        // if query is not "ActionP" but a general fact based query
25                      else {
26                          String arbitary = (String)result.getResult(String.class,
                                joinPointFromQuery[2]);
27                          String pointCut = (String)result.getResult(String.class,
                                    "pointCut");
28                          ctx.setVariablePart(joinPointFromQuery[2], arbitrary);
29                      }
30                  }
31              }
                // In case that pointcut identifier is "before"
32              elseif(joinPointFromQuery[1].equals("after") {
33                  processInvoke(invoke, instance);
34                  ResultSet result = ie.query(query, knowledgeBase, ie.ONE,
                                        ie.TRY_NEXT);
35                  while(result.next()) {
36                      if(query.getNames().equals("ActionP") {
37                          Activity action = (Activity)result.getResult(Activity.class,
                                "action");
38                          if(action instanceof empty) return "empty";
39                          elseif (action instanceof proceed) return "proceed";
40                          else
41                              return "empty";
42                      }
43                      else {
44                          String arbitrary = (String)result.getResult(String.class,
                                joinPointFromQuery[2]);
45                          String pointCut = (String)result.getResult(String.class,
                                "pointCut");
46                          ctx.setVariablePart(joinPointFromQuery[2], arbitrary);
47                      }
48                  }
49              }
50          }
51      }
52      return null;
53  }
```

Fig. 11. Pseudo codes: dynamic weaver and action enabler.

the value of the query's joinPoint, which the user defines. At line 8, it initializes the Mandarax inference engine to prepare query tasks.

From line 10, the matching task, with respect to whether the query's user-defined joinPoint is equal to the current activity position, is shown; if the matching result is true, then a queryingtask using the matched query starts. Finally, actual PRAs are evaluated at the visited method's position according to the "before" or "after" identifier in the query.

We simplify the process of separating general cross-cutting concerns. Some activities, such as empty and proceed only are illustrated. Thus, we focus on how a rule-based AOP can be implemented by integrating Mandarax with Bexee because PRAs used to adapt an application's behavior dynamically are more important than the separation of general cross-cutting concerns.

Inference rules as well as action enabler rules can be evaluated using general predicate-based queries, as shown in lines 25 and 45. If a query retrieved from a knowledge base does not include a predicate of ActionP, then only general backward chaining inference is performed. In this case, the inference result value is transferred to the corresponding BPEL variable.

B. Experiment Results

To test the dynamic characteristics of the proposed approach, we consider three implementation types. The first type is implemented by using Java instead of a BPEL engine (namely, Bexee). A business process for this type is manually coded in Java codes and does not use a BPEL definition. However, the Mandarax engine is used to integrate and process PRAs. The second type is implemented using only the Bexee engine. Its business process naturally includes BPEL activities and elements corresponding to PRAs as well as the core business process. The third type is implemented using Bexee and Mandarax in order to justify the proposed RBAOP framework. Business processes of all types have the same logics and semantics.

To compare these three types fairly, we adopted the travel package process shown in Fig. 5 (an extension of the business process shown in Fig. 4) because Bexee cannot process PRAs and the second implementation type only uses a Bexee engine. The travel package process shown in Fig. 5 is based on some PRAs and Web service interactions which simulate the business activity of an online supplier of hotels, flights, and cars.

We simulated multiple concurrent Web service clients, each of which invokes deployed services multiple times. Three business process types were deployed with the Bexee BPEL

engine or Java logics to orchestrate processes, Mandarax to provide business rule capabilities, and axis to provide the travel package web services. They were deployed on a dual P4 Xeon 2.8 GHz 2 GB RAM server running Windows XP, Tomcat 5.5, and Axis 2. Clients were installed on a Windows XP laptop with P4 1.7 GHz and 2 GB RAM. We used POSDATA's demonstration database as a customer database for the simulation test.

The experimental step involves two scenarios. In the first scenario, PRAs are not changed and all invocations are performed without any interruption. In the second scenario, business elements or activities that serve as business rules are changed.

To estimate the impact of our approach on dynamic adaptability and integration performance we use two performance metrics: net time and gross time. Net time is related to the first scenario, as mentioned above. It is defined as the average period from the time a client sends a request to the time when it successfully receives a full reply from its final activity without any interruption. It includes the xml parsing time of the BPEL definition and web services invocation time; we excluded extra time consumed by the application server, Web server, database server, and network delays for effective approximation. Gross time is related to the second scenario. It is defined as the sum of net time and the average period, which includes code medication time, when the programmer modifies business process codes; re-deployment time, when the business process is modified again; and re-start time when a new business process instance is started. Table 1 shows the net time results when the first scenario was applied to three implementation types.

In the case of RBAOP, there is overhead due to XML parsing time analyzing the BPEL and PRA definition. Additional integration effort is needed between Bexee and Mandarax compared to other implementation types. The first implementation type, Java+Mandarax, has the fastest time with respect to executing the first scenario and has less XML parsing-time overhead, because the BPEL definition code is hard-coded using Java. However, it lacks flexibility regarding modification of the business process when the process's policies changes.

Table 2 presents the gross time results and the impact on configuration changes of each of the three implementations when the second scenario was applied. We assume that the price policy of the travel package products is changed according to the business process of Fig. 5. The R4 rule in Fig. 6 is changed with respect to rule sets or BPEL definition. In this experiment, the calculatePrice() function is replaced by a new calculatePriceCoupon() function as a result of a new promotion policy in the R4 rule. Additionally, we assume the R4 rule is changed when the travel package is normally running in each implementation type, as this can serve to reveal dynamic adaptability features.

Table 1. Net time results for the first scenario.

Time vs. type	JSP + Mandrax	BPEL only (Bexee)	RBAOP (Bexee + Mandarax)
XML Parsing time (a)	10 ms	30 ms	450 ms
Web services call (b)	10 ms	10 ms	10 ms
Net time (c) = a + b	20 ms	40 ms	550 ms

Table 2. Gross time results for second scenario.

Time vs. type	JSP + Mandrax	BPEL only (Bexee)	RBAOP (Bexee + Mandarax)
Code modification (d)	5 min	4 min	1 min
Re-deploy time (e)	50 ms	100 ms	20 ms
Re-start time (f)	20 ms (=net time)	40 ms (=net time)	0
Gross time = net time+d+e+f	5 min 130 ms	4 min 220 ms	1 min 130 ms
Code or spec. change	Modify JSP codes & rules	Modify BPEL spec.	Modify rules
Deployment	Re-deploy Java code &Update rules	Re-deploy BPEL engine	Update rules
Code complexity	8	17	8

In terms of gross time, as seen in Table 2, RBAOP is faster than other types since it modifies only PRAs, not the BPEL definition; moreover, it does not require re-start time. From the viewpoint of implementation architecture, the first and second implementation types must stop the business process instance, modify the business process definition, re-deploy the modified process, and start a new process because neither type supports dynamic change of business rules. However, RBAOP can support uninterrupted process execution since only modified PRAs are updated in the knowledge base of Mandarax. The original business process in Bexee is suspended for a short time while PRAs are updated. When the updating of rules is completed, the suspended process resumes its tasks.

We used a metric suggested in [18] to analyze the control-flow complexity (CFC) of the BPEL processes. According to the CFC metric, only the CFC values of core business are calculated, considering basic BPEL activities, such as, invoke, reply, switch, flow, and so on. The CFC values of the first and third types in Table 2 are smaller than the second type, because

the second type includes all BPEL elements of PRAs as a monolithic definition. The RBAOP approach enables dynamic adaptation when business rules change by decreasing code complexity in terms of integration performance.

V. Conclusion

In this paper, we presented an AOP framework which uses ruled-based knowledge to employ business rules for actual business rule aspects without using existing AOP techniques. We demonstrated a method to integrate rule-based knowledge, accessible through a BRE, with a process orchestration engine compatible with BPEL. Using examples, we demonstrated how PRAs separated through an external rule engine can be represented and evaluated. We further demonstrated how they can be dynamically woven and executed.

The proposed rule-based AOP approach enables business processes to adapt to dynamic changes without stopping runtime processes. It also enables better management by using business rule technology with high level representation methods. Evaluation of three frameworks, including the proposed approach, reveals that the proposed RBAOP method is more useful in terms of fulfilling business rule aspects than other frameworks.

References

[1] F. Curbera, Y. Goland, J. Klein, F. Leymann, D. Roller, S. Thatte, and S. Weerawarana, "Business Process Execution Language for Web Services BPEL, version 1.1, http://www.106.ibm.com/developerworks/webservices/library/ws-bpel/, July 2002.

[2] Business Process Modeling Initiative, "Business Process Modeling Language," http://www.bpmi.org, June 2003.

[3] "Web Service Choreography Interface WSCI 1.0,",http://www.w3.org/TR/ wsci/, Aug. 2002.

[4] A. Charfi and M. Mezini, "Aspect Oriented Web Service Composition with AO4BPEL," *Proc. of the European Conference on Web Services ECOWS 2004*, LNCS 3250.

[5] M.A. Cibrán and B. Verheecke, "Dynamic Business Rules for Web Service Composition," *Proc. of the 2nd Dynamic Aspects Workshop DAW05*, March 2005.

[6] M.A. Cibrán, M. D'Hondt, and V. Jonckers, "Aspect-Oriented Programming for Connecting Business Rules," *Proceedings BIS*, 2003.

[7] AspectJ, http://eclipse.org/aspectj/

[8] A. Charfi and M. Mezini, "Hybrid Web Service Composition: Business Processes Meet Business Rules," *2nd International Conference on Service Oriented Computing*, New York City, USA, Nov. 2004.

[9] The Business Rules Group, "Defining Business Rules: What Are They Really?" http://www.businessrulesgroup.org/, July 2000.

[10] C. Date, "What, not How: The Business Rules Approach to Application Development," Addison-Wesley, 2000.

[11] G. Wagner, "How to Design a General Rule Markup Language," Invited Talk, Workshop XML Technologies for the Semantic Web (XSW 2002), June 2002.

[12] Mandarax business rule system, http://www.mandarax.org/

[13] bexee BPEL execution engine, http://sourceforge.net/ projects/ bexee/

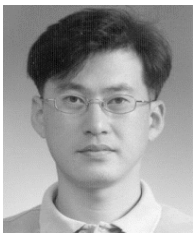
[14] M. D'Hondt, "Hybrid Aspects for Integrating Rule-Based Knowledge and Object-Oriented Functionality," Phd Thesis, Vrije Universiteit Brussels, May 2004.

[15] XML Path Language 1.0, http://www.w3.org/TR/xpath/

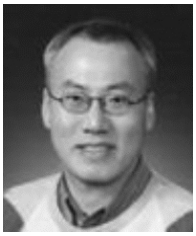
[16] Jens Dietrich, "The Mandarax Manual," http://www.mandarax.org, Dec. 2003.

[17] RuleML, http://www.ruleml.org/

[18] Jorge Cardoso, "Complexity Analysis of BPEL Web Processes," www.interscience.wiley.com, Oct. 2006.



Chankyu Park received the MS degree in electronics engineering from Kyungpook National University in 1997, and the MS degree in software engineering from Carnegie-Mellon University in 2005. He joined Electronics and Telecommunications Research Institute (ETRI) in 1997 and has been a Senior Member of Engineering Staff in the Intelligent Robot Research Division since 2004. His research interests are in semantic web-services, software engineering, and computer vision.

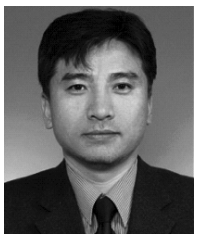


Ho-Jin Choi received the BS degree in computer engineering from Seoul National University in 1982, the MSc degree in computing software and systems design from University of Newcastle upon Tyne in 1985, and the PhD degree in artificial intelligence from Imperial College of Science and Technology in 1995. He worked for DACOM Inc. as a research engineer from 1982 to 1989, for Imperial College as a post-doctoral researcher from 1995 to 1996, and for Hankuk Aviation University as a faculty member from 1997 to 2002. Since 2002, he has been in the faculty of the School of Engineering at Information and Communications University. His main research interests include artificial intelligence, software engineering, and cognitive robotics.

Danhyung Lee is a professor of computer science and engineering in the School of Engineering at Information and Communications University (ICU). He graduated from the Engineering College of Seoul National University, received his Master's degree in Management Science from Arthur D. Little, and received his Ph.D. degree in Information Systems from Virginia Commonwealth University. He served as a principal researcher and Vice-President of SERI (Systems Engineering Research Institute) from 1973 to 1998. His research interests include methods and tools for requirements engineering, software process improvement, and software product lines. Since 1995, he has served as a convener and editor of ISO/IEC JTC1/SC7 WG4.

Sungwon Kang received his BA from Seoul National University, Korea, in 1982, and he received his MS and PhD in computer science from the University of Iowa, USA, in 1989 and 1992, respectively. From 1993, he was a principal researcher with Korea Telecom R & D Group until October 2001, when he joined Information and Communications University. Since 2003, he has been an adjunct faculty member of Carnegie-Mellon University, USA, for the Master of Software Engineering Program. His current research areas include software architecture, software modeling and analysis, software testing, and formal methods.

Hyun-Kyu Cho is a Principle Member of Engineering Staff in the Intelligent Robot Research Division at ETRI. He received the PhD degree in Management Information Systems from Hannam University in 1997 and the MA degree in MIS from Korea University in 1990. He has conducted R&D about computer integrated manufacturing systems electronic commerce systems, and intelligent e-business systems for the past 15 years. Currently, his main research interests are URC proactive service technology and web robot technology.



Joo-Chan Sohn received the MS degree in management information systems from Hankook University of Foreign Studies in 1990. Since joining ETRI, he has been involved with electronic commerce systems and intelligent e-business systems. Currently, he is a Senior Member of Engineering Staff in the Intelligent Robot Research Division. His research interests are intelligent service infrastructure for robots and artificial emotion systems.