PerCAS: An Approach to Enabling Dynamic and Personalized Adaptation for Context-Aware Services

Jian Yu[1], Jun Han[1], Quan Z. Sheng[2], and Steven O. Gunarso[1]

[1] Faculty of Information and Communication Technologies,
Swinburne University of Technology,
Hawthorn, 3122, Melbourne, Victoria, Australia
{jianyu,jhan}@swin.edu.au, 7253702@student.swin.edu.au
[2] School of Computer Science,
The University of Adelaide, SA 5005, Australia
qsheng@cs.adelaide.edu.au

Abstract. Context-aware services often need to adapt their behaviors according to physical situations and user preferences. However, most of the existing approaches to developing context-aware services can only do adaptation based on globally defined adaptation logic without considering the personalized context-aware adaptation needs of a specific user. In this paper, we propose a novel model-driven approach called PerCAS to developing and executing *personalized context-aware services* that are able to adapt to a specific user's adaptation needs at runtime. To enable dynamic and personalized context-aware adaptation, user-specific adaptation logic is encoded as rules, which are then weaved into a base process with an aspect-oriented mechanism. At runtime, the active user-specific rule set will be switched depending on who is using/invoking the service. A model-driven platform has been implemented to support the development and maintenance of personalized context-aware services from specification, design, to deployment and execution. Initial in-lab performance experiments have been conducted to demonstrate the efficiency of our approach.

Keywords: Context-aware services, web services, personalized adaptation, model-driven development, aspect-oriented methodology, business rules.

1 Introduction

Context awareness refers to the system capability of both sensing and reacting to situational changes, which is one of the most exciting trends in computing today that holds the potential to make our daily life more productive, convenient, and enjoyable [7,14,10]. Recently, with the rapid development of service-oriented computing paradigm, Web services have become a major technology for building distributed software systems and applications over the Internet [21]. Through the use of context, a new generation of *smart* Web services is currently emerging as an important technology for building innovative context-aware applications [27,29].

To date, how to build context-aware Web services (CASs in short) that are able to dynamically change its adaptation logic is still a major challenge [13]. Although CASs are meant to be aware of and adaptive to context change, in most existing approaches context-awareness logic is usually tightly coupled with the main functionality of a service and thus not able to change at runtime [26,23]. Another issue that hinders the usability of CASs is that existing context-aware systems and services usually define the context-awareness logic based on a specific *context* instead of a specific *user*, which may lead to system behavior that is not in accord with this user's preference. For example, in a context-aware travel booking system, one of the context-awareness features is that if the weather forecast in the destination is `rainy` when the customer arrives, then a pickup service will be arranged. Such context-awareness logic may be suitable for Alice, but may not be suitable for Bob, who wants to have a rent-car service instead. Because user's long tail of needs can never be exhausted [1], CASs that are able to do context-aware adaptation according to user's personalized needs are highly desirable.

To tackle the above-mentioned challenges, in this paper, we present a novel approach called PerCAS to developing and executing a type of dynamically adaptive CASs that are able to automatically switch context-awareness logic at runtime according to a user's unique adaptation needs. We call such CASs the *personalized CASs*. We have designed and implemented a model-driven development approach to facilitate the modeling of personalized context-aware adaptation logic and the automatic generation of executable service code. A high-level business rule language is proposed to facilitate the specification of a user's personalized context-aware adaptation logic, and an aspect-oriented mechanism is used to integrate the adaptation logic into the main functionality. A runtime environment that integrates both a rule engine and a Web service process engine is implemented to support the dynamic switching between personalized adaptation logic and the execution of personalized context-aware services.

The rest of the paper is organized as follows. Section 2 introduces a motivating scenario that will be referred to throughout the paper. Section 4 describes the PerCAS approach in detail. Section 4 introduce the PerCAS model-driven development platform. Section 5 discusses the execution environment architecture and demonstrates how dynamic and personalized adaptation is achieved in this architecture. In this section, we also discuss an initial performance evaluation of the execution environment. Section 6 is related work discussion and we conclude our paper in Section 7.

2 A Motivating Scenario

In this section, we present a context-aware travel booking service as a motivating scenario. It is worth noting that part of this scenario is borrowed from the case study used in [13].

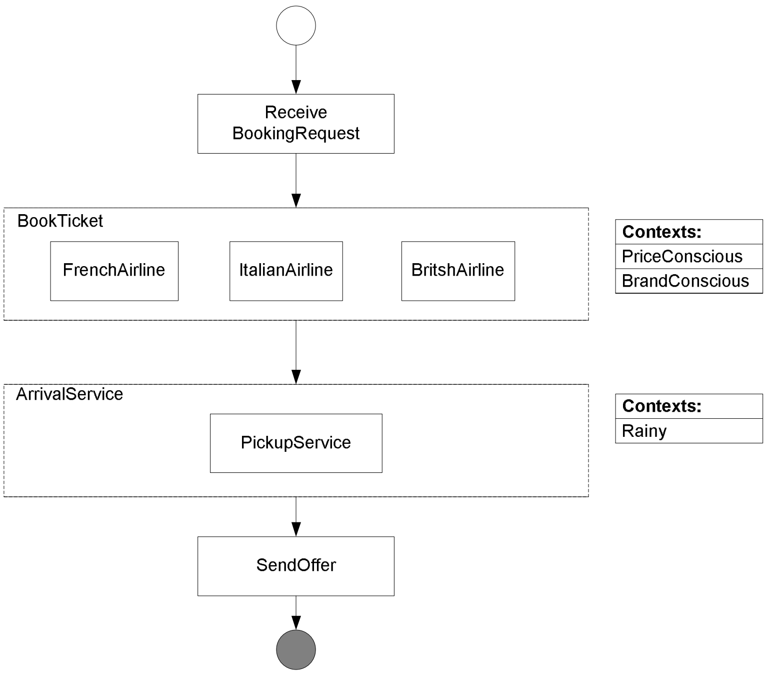


Fig. 1. The travel booking scenario

Figure 1 is the main process of the travel booking service. As we can see, the travel booking service is started when a customer issues a booking request. When the request is received, the service invokes three ticket booking services provided by three different airlines. After that, according to the weather forecast, if it is rainy at the destination when the customer arrives, a pickup service is invoked. Finally the offer is sent to the customer.

The context-awareness features of the travel booking service are as following: 1) if the customer is `PriceConscious`, then the lowest quote from the three airlines will be used; 2) if the customer is `BrandConscious`, then only one airline service needs to be invoked instead of three; 3) if it is forecasted to be `rainy` at the destination, a pickup service will be invoked.

At runtime, it is highly desirable that this context-aware travel booking service can dynamically change its context-awareness logic to suit the personalized needs of customers. For example, if a customer Alice happens to be `RegionConscious`, which means she prefers to fly airlines from a certain region because of her food preference, how can we introduce this new context and its associated context-awareness logic to the service? Furthermore, the other customer Bob may want to use a rent car service instead of the pickup service if it rains, while Alice still wants to use the pickup service. How can we solve this conflict by providing personalized context-awareness logic unique to individual customers?

3 The PerCAS Approach

3.1 Overview

In this section, we briefly introduce the PerCAS approach and discuss several key principles used in the design of this approach.

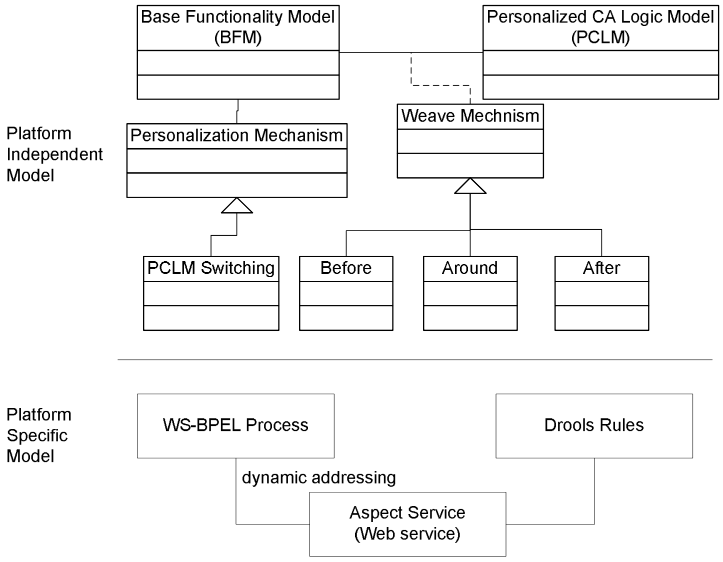

Base Functionality Model (BFM)
Personalized CA Logic Model (PCLM)
Platform Independent Model
Personalization Mechanism
Weave Mechnism
PCLM Switching
Before
Around
After
Platform Specific Model
WS-BPEL Process
Drools Rules
dynamic addressing
Aspect Service (Web service)

Fig. 2. Overview of the PerCAS approach

As illustrated in Figure 2, PerCAS is a model-driven approach that contains a Platform Independent Model (PIM) and a Platform Specific Model (PSM). In the PIM, the main components include a *Base Functionality Model* (BFM in short) that represents the core functionality of the service, a *Personalized Context-Awareness Logic Model* (PCLM in short) that represents the personalized context-awareness logic of the user, a *Weave Mechanism* that integrate the above two models using an aspect-oriented technique, and a *Personalization Mechanism* for switching context-awareness logic between individual users. Because dynamically adaptive systems are generally difficult to specify due to its high complexity and variability [30,20], we adopt the *separation of concerns* principle [15] to manage complexity and variability: the context-awareness logic that needs to be changed is separated from the main functionality. The BFM represents the relatively stable processing procedure of a service; while the PCLM represents the variable context-awareness logic. A process language adapted from BPMN (Business Process Modeling Notation)[1] has been designed as the modeling language for the BFM. As to the PCLM, we have designed a natural language-like high-level rule language as its modeling language. We adopt

[1] `http://www.bpmn.org/`

a rule language to specify the PCLM because: i) Business-level rules are easier to be used by technically non-experienced users because of their atomic and declarative nature [25,5]. In our case, the user may use our rule language to define his/her personalized context-awareness rules. ii) Context-awareness rules as a type of business rules are one of the core components in specifying requirements. Keeping rules in the design model instead of translating them into complex conditional branches of a process not only prevents the paradigm shift from declarative rules to procedural processes but also maintains the modularity and traceability of rules. We adopt an aspect-oriented approach to integrate the BFM and the PCLM using the Weave Mechanism. This approach ensures the modularity of the BFM and the PCLM so that they can evolve independently. If we directly translate rules into process structures and insert them into a process, both modularity and traceability of the rules are lost. Based on the aspect-oriented methodology [8], context-awareness rules can be applied *before*, *after* a service, or *around* it to replace this service. Finally, a personalization mechanism is applied to the BFM for switching between personalized context-awareness rules that are encapsulated in PCLM.

In the PSM, WS-BPEL (BPEL in short) [9], the de facto industry standard for service composition, is used as the process execution language, and Drools[2] is used as the rule execution language. Accordingly, the BFM is transformed to a BPEL process and the PCLM is translated to Drools rules. An aspect service that encapsulates the invocation logic to Drools rules is used as the communication bridge between the BPEL engine and the Drools rule engine. At runtime, the aspect service takes the unique URL to the Drools rule file (corresponding to a unique PCLM, or user) to switch aspects containing personalized context-awareness rules.

3.2 The Base Functionality Model

The BFM captures the main processing logic of a service, excluding all the context-awareness logic. Mainly we reuse the language constructs defined in BPMN for its popularity. To make the BFM and PCLM semantically interoperable, we have extended the `Business Activity` element of BPMN with semantic annotations.

As illustrated in Figure 3, the BFM modeling language in general has two types of elements: the *Flow Object* and the *Connecting Object*, where flow objects are processing entities and connecting objects specify the flow relations between flow objects. There are three types of flow objects: the *Business Activity*, the *Event*, and the *Parallel Gateway*. Business activities represent the main processing unit of a service. Events have the common meaning as defined in BPMN: they are happenings that affect the execution of a process, for example start events and exceptions. Gateways also have the common meaning as defined in BPMN: they determines forking and merging of paths. It is worth noting that although context-awareness logic usually can be specified as static

[2] `http://www.jboss.org/drools/`

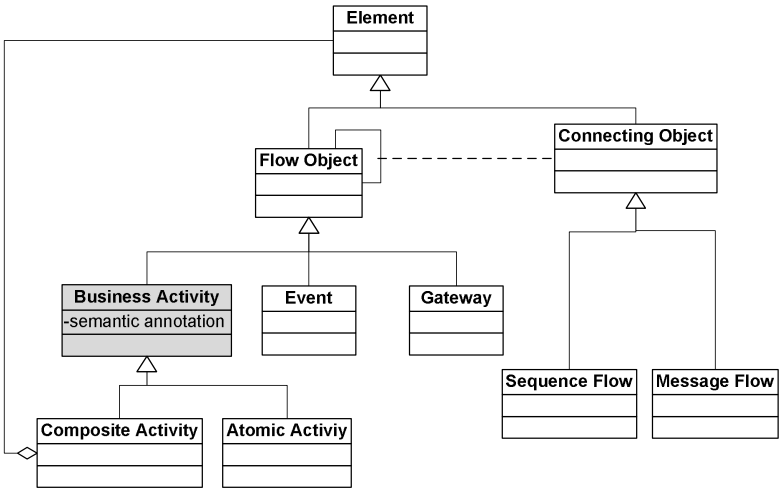


Fig. 3. The BFM language structure

gateway structures, but in our approach, such logic must not be specified in BFM, instead, they should be specified in PCLM.

The detailed definition of the Business Activity is given as follows. A business activity is a tuple of *name*, *inputs*, and *outputs*: $t = <name: \mathcal{N}ame, \mathcal{I}: \mathcal{N}ame \times \mathcal{C}, \mathcal{O}: \mathcal{N}ame \times \mathcal{C}>$, where $\mathcal{N}ame$ is a finite set of names; $\mathcal{C}$ is a finite set of types, and every input or output of a business activity has a name and a type. The type of an input or output parameter is a concept or property defined in an ontology. As we know, an ontology provides a shared vocabulary, which can be used to model a domain—that is, the type of objects and concepts that exist, and their properties and relations [2]. The purpose that we associate an I/O parameter with an ontology concept or property is twofold: first, the ontology serves as the common ground between the BFM and the PCLM and thus makes these two models semantically interoperable; second, the semantics attached to business activities later can be used to semantically discover services that implement business activities. For example, suppose the `BookTicket` activity needs to use the customer information as an input parameter, then we may use an ontology concept `Customer` that has properties such as `firstName`, `lastName`, and `passportNumber`, to give a semantic meaning to this input parameter.

Figure 4 shows the BFM of the motivating example. It only contains two business activities: `BookTicket` and `SendOffer`. We do not include the arrival service in it because the arrival service is part of the context-awareness logic, and such logic needs to be defined in the PCLM instead.

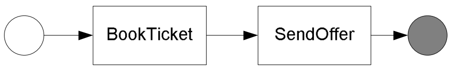


Fig. 4. The BFM of the motivating scenario

3.3 The Personalized Context-Awareness Logic Model

The PCLM captures the context-awareness logic. Usually there are more than one PCLM defined for one BFM. PCLMs can be defined either at design time or runtime, and a specific user can choose one of the PCLM or dynamically defines a new PCLM as his/her PCLM.

A PCLM is composed of a set of rules, and each rule r is defined as a 3-tuple: $r = < type, condition, action >$. Type is defined based on the context related to a PCLM. For example, two contexts `PriceConscious` and `BrandConscious` have the same context type `TicketingPreference`. Rules having the same type can be switched dynamically at runtime. The definition of *condition* and *action* follows the typical event-condition-action (ECA) pattern but the *event* part is specified in the weave mechanism (see the next subsection for details) because the triggering of rules is determined by point cuts in the aspect.

We have designed a natural-language-like high-level rule language to facilitate the specification of PCLM. This language is defined based on the propositional logic based constraint languages such as WSCoL [3] and JML [4], . The syntax of this rule language is defined as follows:

```
<rule>      ::= <type>, <cond> , <action>

<type>      ::= <concept>

<cond>      ::= not <cond> | <cond> and <cond> |
                <cond> or <cond> | <term> <relop> <term>
<term>      ::= <property> | <term> <arop> <term> |
                <const> | <fun> (<term> <term>*)
<property>  ::= <concept>(_<n>)?(.<obj_prop>)*.
                <datatype_prop>
<relop>     ::= less than | less than or equal to |
                equal to | greater than or equal to |
                greater than
<arop>      ::= + | - | * | /
<n>         ::= 1 | 2 | 3 |...

<fun>       ::= <predef> | <usrdef>
<predef>    ::= abs | replace | substring | sum | avg
              | min | max | ...

<action>    ::= (<activity>) | (<property> | <concept>(_<n>)?
                = <term> | <activity>))*
```

As we can see, ontology concepts and properties are used in the specification of a PCLM rule. Because of the *atomic* feature of rules, in many situations, only one instance (or variable) of the same concept/type is involved in the definition of a rule. In such cases, the name of an ontology concept is directly used to represent one of its instances to bring certain convenience to the rule author. For example, to define the condition "*if the customer is price conscious*", we can just write the following natural-language-like condition expression: "*Customer.PriceConscious* *equal to true*", in which the ontology concept `Customer` actually means a specific customer in the context of the rule. If more than one instance of the same concept is needed in a rule expression, number subscriptions, such as `Customer_1`, `Customer_2`, are used to identify a specific instance. Based on Web Ontology Language (OWL) [19], an ontology concept could be a complex structure having both *object properties* and *datatype properties*, where an object property navigates to another concept in the ontology and a datatype property has a specific primitive data type such as *integer*, *boolean*, or *string*. For example, suppose the `Customer` concept has an object property `contact` whose range is the concept `Contact`, and `phoneNumber` is a *string* datatype property of `Contact`. Finally, for the action part, we can either assign the result of a term expression to a variable, or assign the result of the invocation of a business activity to a variable.

The following are examples of three PCLM rules:

$\mathcal{R}_1$*: If a customer is brand conscious, use the airline with the specified brand.*

```
[type]      TicketingPreference
[Cond]      Customer.Preference.brandConscious equal to
    "true"
[Action]    BookTicket(Customer.Preference.brand).
```

$\mathcal{R}_2$*: If it rains at the arrival airport, use the pickup service:*

```
[type]      Weather
[Cond]      ArrivalAirport.weatherCondition equal to "
    Rainy"
[Action]    Pickup(Customer).
```

$\mathcal{R}_3$*: If it rains at the arrival airport, use the rent-car service:*

```
[type]      Weather
[Cond]      ArrivalAirport.weatherCondition equal to "
    Rainy"
[Action]    RentCar(Customer).
```

The user can dynamically put rules into his/her own PCLM. For example, Alice's PCLM is composed of two rules R_1 and R_2: $PCLM_1 = \{R_1, R_2\}$, while Bob's PCLM contains one rule R_3 only: $PCLM_2 = \{R_3\}$. It is worth noting that because the rule type is used for dynamic switching between rules, rules with the same type are not allowed to be put in the same PCLM to achieve deterministic selection.

3.4 The Weave Mechanism and Personalization Mechanism

The weave mechanism connect PCLM rules to BFM business activities based on the concept of *aspect*: each aspect *asp* weaves a type of PCLM rules to a BFM activity: $asp \in \{Before, Around, After\} \times \mathcal{T} \times R.Type$, where $\mathcal{T}$ is the set of

BFM business activities and *R.Type* is the set of PCLM rule types. Similar to AspectJ [11], we also identify three types of aspect: *before aspects*, *around aspects*, and *after aspects*. An aspect is always associated with a business activity. Both before aspects and around aspects are executed before the execution of the associated activity, but if an activity has an around aspect, the execution of this activity will be skipped after the execution of the around aspect. In another word, the around aspect *replaces* its associated activity. From the perspective of the ECA pattern, $event \in \{Before, Around, After\} \times \mathcal{T}$ becomes the triggering event of a PCLM rule.

PCLM rules are associated with an aspect based on their types. So it is a type (or set) of PCLM rules that are associated with a BFM activity instead of a single PCLM rule. For example, we can define two context-awareness aspects for the travel booking service discussed in Section 2:

$$asp_1 = \{Around, BookTicket, TicketingPreference\}$$
$$asp_2 = \{After, ArrivalService, Weather\}$$

In asp_2, Because R_2 and R_3 belong to the same type `Weather`, they can be dynamically switched and applied to the `ArrivalService` activity.

It is worth noting that the interoperability between an BFM activity and its associated PCLM rules is established through the predefined ontology. For example, the input parameters of the `BookTicket` activity must contain two parameters having semantic annotation `DepartureAirport` and `ArrivalAirport`, so that the associated rule (for example R_2) can use these properties in its definition.

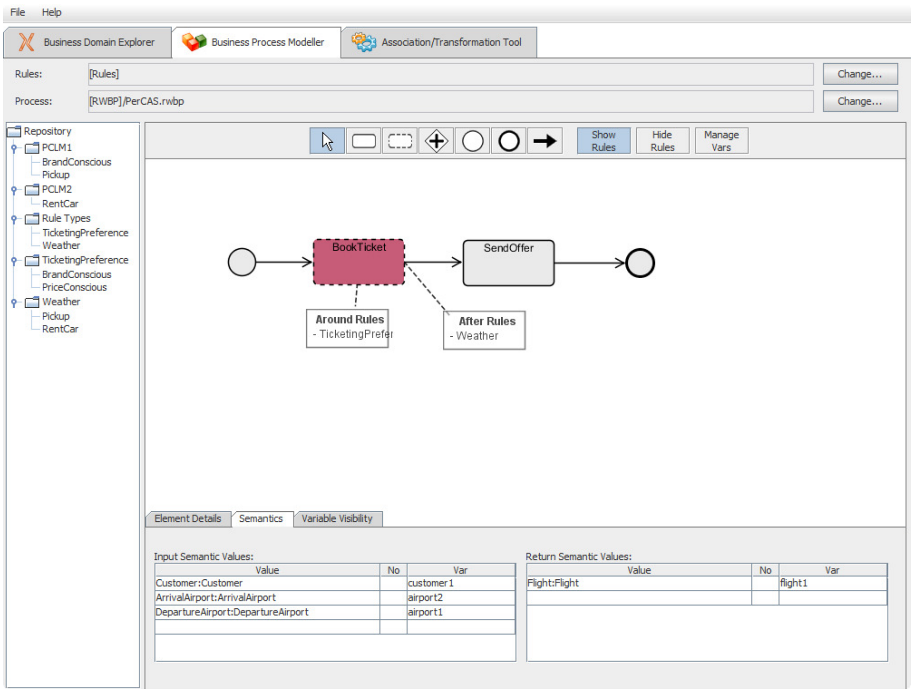
Finally the personalization mechanism is used to associate a user to a PCLM, for example $PCLM_1.user = Alice$, so that at runtime when it is identified that the invocation is from this user, his/her specific PCLM will be used, and rules will be selected from this PCLM to apply to the corresponding BFM. If a context-awareness aspect is defined while there is no rule can be used (based on the rule type) in the specific PCLM, then this aspect will be ignored. For example, suppose Bob's PCLM has no rule with type $TicketingPreference$, then asp_1 will be ignored when Bob invokes the travel booking service.

4 The PerCAS Development Platform

We have implemented a model-driven platform for graphically specifying the PerCAS PIM models and for automatic transformation of these models to executable PSM code.

4.1 The Graphical Development Interface

Figure 5 shows the main graphical development interface of the PerCAS platform. There are totally three tabs: the left tab is an OWL viewer used for users to explore ontology concepts defined in OWL; the middle tab, as shown in Figure 5, is the main environment for defining PerCAS models; the right tab is for



Input Semantic Values:

Value	No	Var
Customer:Customer		customer1
ArrivalAirport:ArrivalAirport		airport2
DepartureAirport:DepartureAirport		airport1

Return Semantic Values:

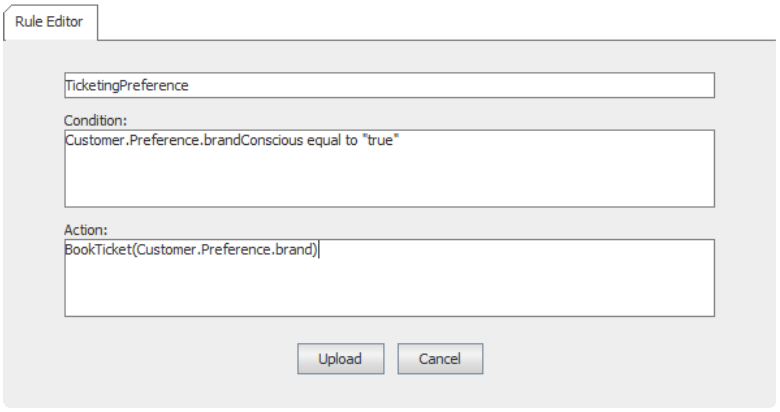
Value	No	Var
Flight:Flight		flight1

Fig. 5. Snapshot of the PerCAS Platform

transforming models to executable code. For space limitation, we only introduce the main graphical environment (the middle tab). As we can see, the left pane displays the structure of the PCLM rule repository: there are two PCLMs defined: one contains the `BrandConscious` rule (R_1) and the `Pickup` rule (R_2), and the other contains the `RentCar` rule (R_3). As discussed in Section 3.4, $PCLM_1$ may be used by Alice, and $PCLM_2$ by Bob. In the rest of the structure, two rule types: `TicketingPreference` and `Weather` are defined, with each type contains two rules.

The middle pane is the main canvas for composing a PerCAS service. BFM language constructs are displayed as a list of buttons on top of the canvas. When the user creates a BFM activity, its semantics can be specified in the bottom pane. If we select a concept or datatype property from the drop-down menu that contains all the concepts and datatype properties in the domain ontology as the type for a parameter, a variable is automatically created to represent this parameter. As shown in the snapshot, the `BookTicket` activity has three input parameters with type `Customer`, `DepartureAirport` and `ArrivalAirport`, and one output parameter with type `Flight`.

After the BFM model is created, the user may drag-n-drop one of the rule types from the left pane to an activity in the middle canvas. The platform then will ask the user whether weave the rule type $before$, $around$, or $after$ the activity. As shown in the figure, the `TicketingPreference` type is weaved as an



TicketingPreference

Condition:
Customer.Preference.brandConscious equal to "true"

Action:
BookTicket(Customer.Preference.brand)

Fig. 6. The PCLM Rule Editor

around aspect, and the `Weather` type is weaved as an after aspect. It is worth noting that if a BFM activity is attached with PCLM rules, then the solid line of its shape becomes the dash line to indicate that it is context-dependent and adaptive.

A new PCLM rule can be created in the "Rule Editor" dialog box, which will appear if we right-click one of the folder icons in the left rule repository pane and select "New Rule" from the pop-up menu. As shown in Figure 6, the rule editor uses the concepts in the domain ontology to define the *condition* and *action* components of a rule. It is worth noting that all the I/O parameter variables in the base model that are visible to a rule will be automatically bound to the corresponding concepts or properties in the rule.

4.2 Transformation

Before we can transform the defined PerCAS PIM to executable code, we need to associate each BFM activity with a Web service. This can be done in the "Association and Transformation" tab of the graphical interface.

Each PCLM rule is automatically transformed to an executable Drools rule. Figure 7 shows the generated Drools rule code for rule $\mathcal{R}_1$ discussed in Section 3.3. In order to keep the invocation of Web services associated with activities defined within rules self-contained, service information for Web services associated with activities defined within rules are encoded directly into the rule code. First, the bindings for ontology classes used in the rule as well as an *enabler* helper-class are defined (Lines 5-6), followed by the *condition* statement as translated into Drools syntax (Lines 8). If the condition is evaluated as true, the Web service associated with the *BookTicket* will be invoked, using the *enabler* helper class (Lines 12-25).

The weaved BFM model is transformed to a BPEL process. Constructs such as *Start Event* and *Activity* that does not have aspects are translated directly into their corresponding BPEL constructs (in this case, *receive* and *invoke*).

```
rule "BrandConscious"
  dialect "java"

when
  $enabler           : Enabler()
  $Customer          : Customer()

  Customer((Preference(brandConscious == "true"))

then

  try {
    String[] wsInfo = { "http://localhost:8080/
                        BookTicket",
                        "bookTicket", "BookTicketService",
                        "ContactServicePort",
                        "http://localhost:8080/
                        BookTicketService/BookTicketService
                           ?
                        wsdl"};

    String[][] varInfo  = {{ $Customer.getBrand()}};
    String[][] varNames = { { "CustomerBrand" }};

    $enabler.runService(wsInfo, varInfo, varNames);
  } catch (Exception e) { e.printStackTrace() };

end
```

Fig. 7. Drools rule code corresponding to Rule $\mathcal{R}_1$

For activities that have aspects, we use a special Web service called *aspect service* as the communication bridge between the BPEL process and the rules running on the Drools rule engine. An aspect service will be invoked before invoking an activity if it has *before* and/or *around* aspects, and another aspect service will be invoked after invoking an activity if it has *after* aspects. To achieve dynamic switching between PCLMs, each PCLM is translated to a Drools rule file, and the aspect service takes as input a URI to the Drools rule file, along with the values and ontology class names of all variables involved in the aspect. When a user invokes a PerCAS service, his/her unique PCLM URL will be used as an parameter to the aspect service. The aspect service returns two Boolean values corresponding to abort and skip evaluation outcomes, as well as the values of all variables that may have been updated based on rule evaluation. Finally, conditional constructs are inserted around the activity invocation to handle abort and skip actions based on the return of the aspect service.

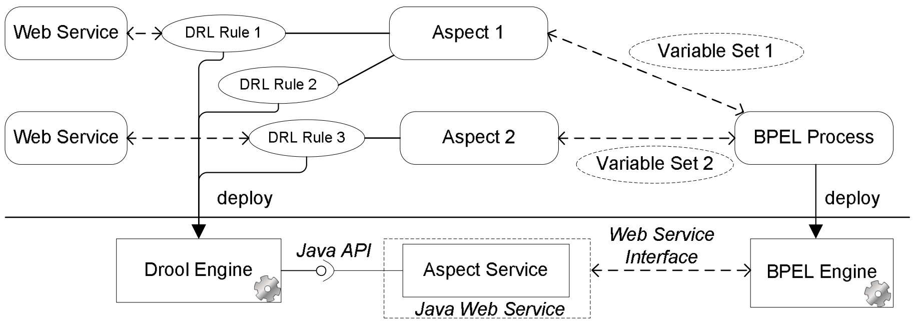


Fig. 8. The architecture of the PerCAS runtime environment

5 The Runtime Environment

We have implemented the PerCAS runtime environment based on Riftsaw-2.3.0 open source BPEL engine [3] and Drools-5.0 rule engine. Both engines are running inside the JBoss Application Server-7.0.

Figure 8 is the architecture of the PerCAS runtime environment. The bottom level of the anatomy includes the main components of the runtime environment: a *Drool engine*, a *BPEL engine*, and a *generic aspect service* that encapsulates the rule invocation logic. The aspect service is written in Java and exposed as a Web service for the BPEL process to invoke. Every time when an aspect in the process is reached for execution, the aspect service is invoked and corresponding variables (including the IO parameters of its associated activity and user selected variables) are passed from the process to it; these variables are used in the execution of the rules of the aspect. After all the rules in the aspect are executed, these variables are updated and passed back to the process.

Next we use this architecture and the motivating scenario to briefly demonstrate how dynamic and personalized adaptation is achieved in PerCAS. Suppose Alice is using the travel booking service, then the url to $PCLM_1$ (defined in Section 3.3) will be used as an parameter to the aspect service, and because asp_1 (defined in Section 3.4) is an around aspect to the `BookTicket` service, this aspect will select a rule with type `TicketingPreference` from $PCLM_1$, which is R_1. Similarly, when Bob is invoking the service, $PCLM_2$ will be used and R_2 will be selected and executed instead. Because the BFM process and the PCLM rules are separately deployed, it is possible to change the PCLM rules while the process is still running. For example, when the `BookTicket` service is still running, Alice may change the rules defined in $PCLM_1$, e.g., change her arrival service preference from `Pickup` to `RentCar`. If the modified $PCLM_1$ is successfully deployed before asp_2 (the *after* aspect) is executed, the new rule will be used in asp_2.

[3] `http://www.jboss.org/riftsaw/`

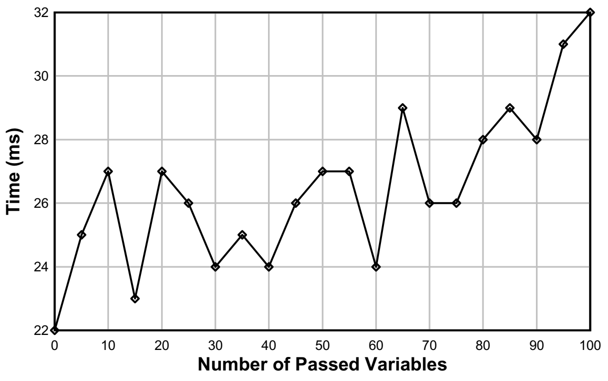


Fig. 9. Execution time of a single aspect service w.r.t. the number of passed variables

According to the above discussed architecture, we can see that the main performance impact of this runtime environment lies in the aspect service, which is responsible for executing the context-awareness logic outside the BPEL engine. We have conducted an initial experiment to test the impact of invoking a single aspect service with various number of randomly generated primitive type variables passed. Every setting is tested five times and the average execution time of an empty aspect service w.r.t. the number of passed variables is shown in Figure 9. As we can see, it costs 22 ms to invoke an empty aspect service without passing any variables and costs 32 ms to invoke an empty aspect service with 100 variables passed to it. This result shows that the variable exchange between the Riftsaw BPEL server and the Drools server is very fast, and there is only 10 ms increase from passing no variable to passing 100 variables. The reason could be that these two servers are two components that both run inside the same JBoss application server.

6 Related Work

The PerCAS approach presented in this paper is closely related to two categories of research work: one is model-driven development of context-aware services, and the other is dynamic context-aware process adaptation. In the rest of this section, we discuss related work from these two perspectives.

As mentioned by Kapitsaki et al. [16], the model-driven approach is a popular approach to developing context-aware services because of its strong support to the development process. ContextServ [26] is a platform that uses UML[4] to model contexts, context-awareness, and services, and then transforms the model to an executable context-aware Web service. Composite contexts can be modeled by composing atomic contexts using UML state diagrams. The main context-awareness features that can be modeled by ContextServ include *context binding*, which binds a context to the input parameter of a service, and

[4] `http://www.uml.org/`

context triggering, which modifies the output of a service to suit a specific context. CAMEL (Context Awareness ModEling Language) and its associated development tools [12,28] combine model-driven development and aspect-oriented design paradigms so that the design of the application core can be decoupled from the design of the adaptation logic. In particular, CAMEL categorizes context into state-based which characterizes the current situation of an entity and event-based which represents changes in an entity's state. Accordingly, state constraints, which are defined by logical predicates on the value of the attributes of a state-based context, and event constraints, which are defined as patterns of event, are used to specify context-aware adaptation feature of the application. CAAML (Context-aware Adaptive Activities Modeling Language) [17] aims at modeling context-aware adaptive learning activities in the E-learning domain. This language focuses on modeling two classes of rules - rules for context adaptation and rules for activity adaptation - to support pedagogical designing. The above approaches mainly focus on how to specify context-awareness features of a single service or software component at *design time*. The focus of the PerCAS approach instead is on making the context-awareness features changeable at *runtime* based on user preferences. Also, PerCAS supports to do context-aware adaptation on a process instead of a single service, which is the reason that we adopt BPMN instead of UML as the base modeling language.

In terms of dynamic context-aware process adaptation, Apto [13] is a model-driven approach for generating the process variants that corresponding to the changes in requirements and context. Necessary changes to a process is modeled as *evolution fragments* using UML, and a process variant can be created by applying an evolution fragment to the base process. Dynamic adaptation is achieved by first generating a new process variant, then transforming the process variant to a BPEL process, and then re-deploying this new BPEL process. Although both Apto and PerCAS can achieve the same goal of creating dynamic and personalized context-aware services, Apto clearly needs more professional experience to create a correct evolution fragment as it needs full understanding of both the process logic and the process language constructs, while PerCAS advocates to use natural language-like rules to define context-awareness logic. There are quite a few works aiming at extending the dynamic adaptability of BPEL processes using aspects and rules. AO4BPEL [6] is an aspect-oriented extension to BPEL that supports dynamic weaving of aspects in BPEL processes. Although they also advocate to use rules in an aspect, a rule engine is not integrated in their approach and rules are manually mapped to BPEL conditionals. Marconi et al. [18] also proposed a set of constructs and principles for embedding the adaptation logic within a flow language specification and showed how BPEL can be extended to support the proposed constructs. Rosenberg and Dustdar [24] proposed a runtime environment where both a BPEL engine and a rule engine are connected to a service bus. Dynamic adaptation is achieved through intercepting the messages exchanged between the process and a partner service and invoking business rules running on the rule engine before and after the execution of the partner service. This approach may not be able to implement the *around* aspect as rules are inserted before and after the invocation of a partner services while the partner service cannot be disabled or replaced. Paschke and Teymourian [22] discussed a rule based business process execution environment where a rule engine is deployed on an ESB (Enterprise Service Bus) and exposed as Web services. Dynamic adaptation is achieved by explicitly defining and integrating Rule Activities, which invoke the rule service, in the BPEL process, and rules can be modified and applied without re-deploying the process. The above works mainly focus on the execution language and environment, while PerCAS is a systematic engineering approach with a graphical modeling language and development platform.

7 Conclusion

In this paper, we have presented PerCAS, a model-driven approach for developing dynamic and personalized context-aware services using aspects and rules. We have introduced the models and methodology of separating the context-awareness logic from the base functionality logic of a service, as well as weaving the context-awareness logic to the base process. A natural language-like rule language is proposed for specifying context-awareness logic and personalized rules can be dynamically switched at runtime. We have also developed a development platform to support the graphical modeling and transformation of such services, and a runtime environment that integrates both a BPEL engine and a Drools rule engine for their execution. In the future, we plan to apply this approach in more real-life case studies to validate its effectiveness. We also plan to investigate runtime validation techniques that can be used to check the consistency of context-awareness logic switching.

References

1. Anderson, C.: The Long Tail: Why the Future of Business is Selling Less of More. Hyperion Books (2006)
2. Arvidsson, F., Flycht-Eriksson, A.: Ontologies I (2008), `http://www.ida.liu.se/~janma/SemWeb/Slides/ontologies1.pdf`
3. Baresi, L., Guinea, S.: Self-Supervising BPEL Processes. IEEE Transaction on Software Engineering 37(2), 247–263 (2011)
4. Burdy, L., Cheon, Y., Cok, D.R., Ernst, M.D., Kiniry, J.R., Leavens, G.T., Leino, K.R.M., Poll, E.: An Overview of JML Tools and Applications. Int'l J. Software Tools for Technology Transfer 25(3), 40–51 (2005)
5. Charfi, A., Mezini, M.: Hybrid Web Service Composition: Business Processes Meet Business Rules. In: Proc. of the 2nd International Conference on Service Oriented Computing (ICSOC 2004), pp. 30–38 (2004)
6. Charfi, A., Mezini, M.: AO4BPEL: An Aspect-oriented Extension to BPEL. World Wide Web 10, 309–344 (2007)
7. Dey, A.K., Mankoff, J.: Designing Mediation for Context-aware Applications. ACM Trans. on Computer-Human Interaction 12(1), 53–80 (2005)
8. Elrad, T., Filman, R.E., Bader, A.: Aspect-Oriented Programming: Introduction. Commun. ACM 44(10), 29–32 (2001)

9. Evdemon, J., Arkin, A., Barreto, A., Curbera, B., Goland, F., Kartha, G., Khalaf, L., Marin, K., van der Rijn, M.T., Yiu, Y.: Web Services Business Process Execution Language Version 2.0. BPEL4WS Specifications (2007)
10. Ferscha, A.: 20 Years Past Weiser: What's Next? IEEE Pervasive Computing 11, 52–61 (2012)
11. Gradecki, J.D., Lesiecki, N.: Mastering AspectJ: Aspect-Oriented Programming in Java. Wiley (2003)
12. Grassi, V., Sindico, A.: Towards Model Driven Design of Service-Based Context-Aware Applications. In: Proc. of the International Workshop on Engineering of Software Services for Pervasive Environments: In Conjunction with the Sixth ESEC/FSE Joint Meeting, pp. 69–74 (2007)
13. Jaroucheh, Z., Liu, X., Smith, S.: Apto: A MDD-based Generic Framework for Context-Aware Deeply Adaptive Service-based Processes. In: Proc. of the 2010 IEEE International Conference on Web Services (ICWS 2010), pp. 219–226 (2010)
14. Julien, C., Roman, G.C.: EgoSpaces: Facilitating Rapid Development of Context-Aware Mobile Applications. IEEE Trans. on Software Engineering 32(5), 281–298 (2006)
15. Kambayashi, Y., Ledgard, H.F.: The Separation Principle: A Programming Paradigm. IEEE Software 21(2), 78–87 (2004)
16. Kapitsaki, G.M., et al.: Context-Aware Service Engineering: A Survey. J. Syst. Software (2009)
17. Malek, J., Laroussi, M., Derycke, A., Ben Ghezala, H.: Model-Driven Development of Context-aware Adaptive Learning Systems. In: Proc. of the 10th IEEE International Conference on Advanced Learning Technologies (ICALT 2010), Washington, DC, USA, pp. 432–434 (2010)
18. Marconi, A., Pistore, M., Sirbu, A., Eberle, H., Leymann, F., Unger, T.: Enabling Adaptation of Pervasive Flows: Built-in Contextual Adaptation. In: Baresi, L., Chi, C.-H., Suzuki, J. (eds.) ICSOC-ServiceWave 2009. LNCS, vol. 5900, pp. 445–454. Springer, Heidelberg (2009)
19. Mcguiness, D.L., van Harmelen, F.: OWL Web Ontology Language Overview. W3C Recommendation (February 2004), `http://www.w3.org/TR/owl-features/`
20. Morin, B., Barais, O., Nain, G., Jezequel, J.M.: Taming Dynamically Adaptive Systems using Models and Aspects. In: Proc. of the 31st International Conference on Software Engineering (ICSE 2009), pp. 122–132 (2009)
21. Papazoglou, M.P., Traverso, P., Dustdar, S., Leymann, F.: Service-Oriented Computing: State of the Art and Research Challenges. Computer 40(11), 38–45 (2007)
22. Paschke, A., Teymourian, K.: Rule Based Business Process Execution with BPEL+. In: Proc. of I-KNOW 2009 and I'SEMANTICS 2009, pp. 588–601 (2009)
23. Prezerakos, G.N., Tselikas, N., Cortese, G.: Model-Driven Composition of Context-Aware Web Services Using ContextUML and Aspects. In: Proc. of the IEEE International Conference on Web Services 2007 (ICWS 2007), pp. 320–329 (2007)
24. Rosenberg, F., Dustdar, S.: Usiness Rules Integration in BPEL - a Service-Oriented Approach. In: Proc. of the 7th IEEE International Conference on E-Commerce Technology, pp. 476–479 (2005)
25. Ross, R.G.: Principles of the Business Rules Approach. Addison-Wesley (2003)
26. Sheng, Q.Z., Pohlenz, S., Yu, J., Wong, H.S., Ngu, A.H.H., Maamar, Z.: ContextServ: A Platform for Rapid and Flexible Development of Context-Aware Web Services. In: Proc. of the 31st International Conference on Software Engineering (ICSE 2009), pp. 619–622 (2009)
27. Sheng, Q.Z., Yu, J., Dustdar, S. (eds.): Enabling Context-Aware Web Services: Methods, Architectures, and Technologies. CRC Press (2010)

28. Sindico, A., Grassi, V.: Model Driven Development of Context Aware Software Systems. In: Proc. of the International Workshop on Context-Oriented Programming (COP 2009), New York, NY, USA, pp. 7:1–7:5 (2009)
29. Truong, H.L., Dustdar, S.: A Survey on Context-Aware Web Service Systems. International Journal of Web Information Systems 5(1), 5–31 (2009)
30. Zhang, J., Cheng, B.H.C.: Model-Based Development of Dynamically Adaptive Software. In: Proc. of the 28th International Conference on Software Engineering (ICSE 2006), pp. 371–380 (2006)