# Supporting Differentiated Services With Configurable Business Processes

Aries Tao Tao & Jian Yang
Department of Computing, Macquarie University,
Sydney, NSW 2109, Australia
{tao, jian} @ics.mq.edu.au

**Abstract:** In order to support business service flexible and reusable, it is desirable to provide users or applications the same service but with different service quality, different interaction paths, or different outcomes. We call this design principle as Service Differentiation. In this paper we present a fully working service design method where variability is externalized as business policies so that the business process(es) does not need to be altered for any anticipated changes. Service differentiation is realized by configured business processes and interfaces, and the dynamic 'binding' between user/application with a specific interface is determined by policy during service invocation time.

## 1 Introduction

Supported by Web Service technology, Service Oriented Computing (SOC) allows resources on a network to be made available as services that can be accessed without the knowledge of their underlying platform implementation[CHT03]. To better understand how services are designed and developed, it is important to understand the relationship among three core concepts in SOC: service, service interface, and business process:

- A service is a business concept that should be specified with an application or users of the service in mind[PY02].
- Service interface is the specification for user (or program) to interact with the service. It is supported by business process(es). It consists of a list of directed messages that allow the users to interact with the corresponding business process(es).
- The business process(es) implements the functionality of service. It consists of activities that perform service functions.

Now days, the main aim of a business is to provide better, and more flexible services to its customers. Quite often we find that different users (client programs) may need different ways to interact with the same service, or may expect different outcomes from the same service. These differences can be either hidden from the user, for example, the outcomes are dependant on the user profile; or presented to the user so that different interfaces for the

same service should be provided. Therefore from business process design point of view, two types of supports are required:

- operational support - business process variability needs to be provided for different users and usages. Take Online Pharmacy Service (OPS) as an example, OPS provides different discount rates depending on customers' profiles:
  - offers 10% discount for VIP customers.
  - offers no discount for normal customers.
- interface support - users with different behavior semantics need to access the same service through different interfaces. Continuing with the same OPS example, different approval procedures are required for different types of medicines that customers intend to purchase:
  - requires doctor's prescription from the customers who purchase prescribed medicine.
  - does not require any approval from the customers who purchase un-prescribed medicine such as Panadol.

Given the fact that different users or applications often have different (but overlapping) requirements, they may require different quality levels, interaction patterns or outcomes from the same service. As a result, it is more desirable to provide a single service with variations than several unrelated services. We refer this design principle as *service differentiation*. The way we want to realize service differentiation, i.e., single service with different interaction patterns for different users/applications presents an interesting analogy to object orientation in terms of overloading and polymorphism. The *service differentiation* rules for "binding" a specific service interface and outcome with user *behavior* or *profile* are governed by business *policies*, e.g., binding '10% discount' with 'VIP customers'.

Now let us have a close look at current service design approaches in relation to internal business process and service interface design. In the past, Business policies or rules are hard coded in the business process as different activity execution conditions. As a result, service is supported by a fixed business process which provides the same functionalities to all the users through the same service interface. Recently, there is a common understanding that the business rules or policies should be separated from the business application in order to achieve flexibility and manageability ([AN04], [BBC06], [TPP05]), but these work focus on solving problem in security domain rather than service differentiation. However, there is no work that has been done on describing how differentiated service is developed by externalizing policies so that different users/applications can be 'treated' differently by the same service in terms of different interfaces or functionalities.

The basic design philosophy of ours, and one that distinguishes us from others, is that an Abstract Business Process is developed for all users in all circumstances with different *policy configured* business processes generated for different users and different interaction patterns. This requires a new service design approach that separates the generic business activities that are applicable to all the circumstances from those only applicable to particular group of people or under specific conditions. Thus we propose a new service design

approach that supports single service with multiple business processes which supports *service differentiation*. Our design is based on the following ideas:

- Use *Abstract Business Process (ABP)* to support the generic functionalities that are required by all the users.
- Use *Policy* to specify how the service should be differentiated for different users.
- Multiple *Policy Configured Business Processes (PCBPs)* and related service interfaces can be derived based on *ABP* and *Policies* to support different users/applications.

Ultimately, our priorities and the focus of the paper are:

- to separately maintain business policies,
- to dynamically generate business processes if necessary,
- to present different service interfaces to the users/applications based on business policies, and
- to have minimal impact on business service when policies are modified.

This paper is organized as follows: Section 2 discusses the related work. The case study is introduced in Section 3. Service design are discussed in Section 4. We finally concludes our work in Section 5.

## 2 Related Work

In this section we will discuss related work from three aspects. Firstly we review the research work that has been done on *differentiated services*. Secondly we review the current service description techniques, arguing that they can not effectively describe differentiated services. Finally we analyze the related Web Service[WS] Standards which can be used for our work.

The idea of service differentiation (DiffServ) [RFC2475] was firstly proposed in the area of networking to manage traffic streams in networking applications. For example, some traffic is treated better than the others (faster handling, more average bandwidth, and lower average loss rate). Richard Veryard in [VE00] argued that the differentiated services should be used as a design pattern in SOC area. However, no methodology has been proposed for service differentiation. In this paper we will demonstrate a differentiated service design specification based on the framework that presented in our previous work[TY07].

Work has been done in the area of specifying public interface for business processes. Chiu et al [CCK02] presented a meta-model for Service Interface as Workflow Views, which provided a novel approach to derive Service Interface (as Workflow View) from a Service (as workflow). By abstracting Service Interface as a certain subset of a Service, it allows internal information to be hidden from external users. To support different user groups,

Zhao, Liu and Yang[ZLY05] proposed the concept of *relative workflow view* by explicitly extracting visibility constraints (Invisible, Traceable, Contactable) on activities of workflow. Based on different visibility constraint for different users over the same workflow, multiple relative workflow views could be derived for different users with different relationship with the service. However, the service provider has to manually set constraints for every new service user, the system does not scale well. Using a completely different strategy, our work allows service interface to be generated from configured business processes, which are derived automatically based on business policies. As a result, the maintenance effort is shifted from managing business processes to managing policies.

Now let us have a close look at current relevant web service standards. WSCI[AAF02] allows the service to be described as a sequence of Web Service calls binding to WSDL [CCM01]. In advance, BPEL[ACD03] allows the service interface to be described as an abstract business process, which is a subset of BPEL process. BPEL allows several abstract business processes to be derived for a service. However, BPEL specification does not provide any mechanism or standard to generate multiple service interfaces, which is often required by business as illustrated later in our example. Several Semantic Web Service Description standards such as OWL-S[MBD04], WSMO[RLK04] have been proposed. Comparing with WSDL and BPEL, Semantic Web Service provides better support in common understanding of policy semantics and reasoning on complex relations between policy concepts[MFN05]. However, service is designed to have only one Service Process Model. Such design limits the service flexibility to support different user interactions. Our work can be used to extend current Semantic Web Services by allowing multiple service Process Model to be derived for a single service, and hence allow users to access the same service in different ways.

In order to relate the policy with Web Services, Web Services Policy Framework (WS-Policy)[BBC06] was proposed by the World Wide Web Consortium (W3C). It is a general framework for specifying various Web service properties in a way that complements WSDL and BPEL. On the other hand, Web Services Policy Language (WSPL) [AN04] was proposed by the Organization for the Advancement of Structured Information Standards (OASIS). It is suitable for specifying a wide rage of policies, e.g., acceptable and supported encryption algorithms or privacy guarantees. Both WS-Policy and WSPL focus on supporting security domain only. Vladimir et al[TPP05] extends the WS-Policy for monitoring and adaptation of Web services and their composition. However, their work also does not support service differentiation. Our work complements their work by providing a service design which allows policy based service differentiation.

## 3 Motivating Example

In order to understand the rational behind the proposed approach and the concept of *differentiated service*, we use Online Pharmacy Service (OPS) as a case study. The aim is to develop multiple policy configured business processes which provide different functionalities to different customers (e.g. further discount for VIP customer) via multiple service interface(s). For space limitation, we only use the Checkout process of OPS as an example.

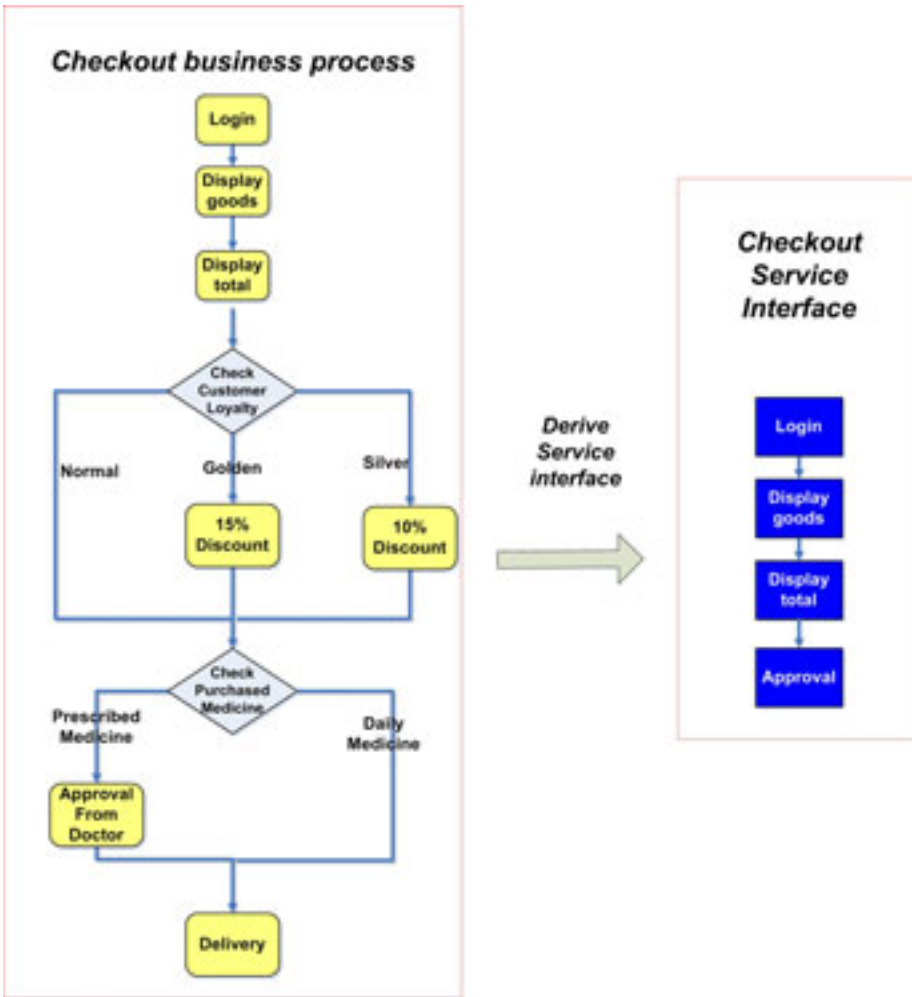



Figure 1: Current Service Design For Checkout Process

As showed in Figure 1, the current design of Checkout business process consists of following activities, some of which may not be available for all users:

- Login - receives the user name and password in order to identify the customer.
- Display Goods Selection - displays a list of goods selected by the customer.
- Display Total Price - displays the total price the customer needs to pay for the goods.
- Display discounted price - provides different discount rates depending on the customer profile:
  - For normal customer - there is no discount available.
  - For silver VIP customer - 10% discount is offered.
  - For golden VIP customer - 15% discount is offered.
- Approval - may require extra approval depending on the type of purchasing medicine:
  - For prescribed medicine - prescription from a doctor is required.
  - For daily(un-prescribed) medicine - no approval is required.
- Receive Payment - receives payment from customer and return invoice.
- Product Delivery - contacts delivery company to deliver the product to customers.

The Checkout process provides the functionalities that is required by all customers, however its interface description is not specific, and can be irrelevant to some user group:

- The interface does not provide enough information for different user group. For example, different users may receive different discount rates depending on their profile: for silver VIP customers and golden VIP customers, they can receive 10%/15% discount of the total amount; and there is no discount for non-VIP users. However, the discount information is not presented in the service interface.
- Some information supported by the interface is unnecessary for certain users. For example, the Approval activity in interface is required for prescribed medicine purchase only. It is unnecessary for un-prescribed medicine purchase. However, the users who purchase the un-prescribed medicine still have to go through the Approval activity.

Instead of hard-coding all the information into one checkout business process as most current service developments do, we separate the policy from the business process in order to describe different interfaces for different groups of users. Each interface provides richer information to help users to access the service, for example: specify different discount rates for different users, remove the Approval activity from the interface for non-prescribed medicine purchase. Each service interface is supported by one Policy Configured Business Process. In the next section we will show how Policy Configured Business Processes are determined to support different users.

## 4 Differentiated Service Design

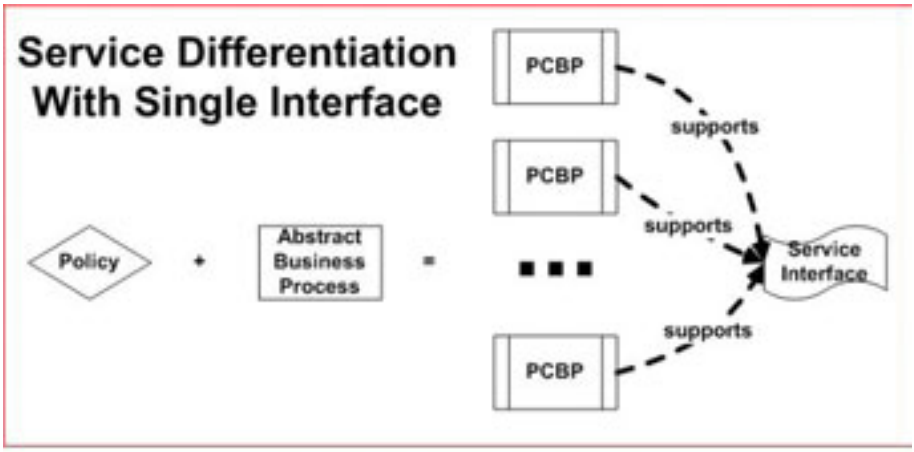
Instead of supporting a service with one flat and monolithic business process, our design philosophy is to separate the specific functionalities only available for certain users from the generic functionalities that are available for all users. The design consists of four components:

- *Abstract Business Process (ABP)* that implements the generic functionalities by a set of activities which are provided to all users.
- *Business policies* that provides different functionalities for different users based on:
  - **User profile** which is determined prior to the user-service interaction. Take the Online Pharmacy Service (OPS) as an example, OPS provides different discount rates depending on customers' profiles which is determined before user login.
    - offers 10% discount for silver VIP customers.
    - offers 15% discount for golden VIP customers.
    - offers no discount for normal customers.
  - **User behavior** which is determined during the user-service interaction at runtime. Continuing with the OPS example, different approval procedures are required for different types of purchasing medicine which is determined during the Checkout process:
    - requires doctor's prescription from people who purchase prescribed medicine.
    - does not require any approval from people who purchase un-prescribed medicine.

  As showed in Figure 2, depends on the user needs, the policies could support two different kinds of service differentiation:

  - *Service Differentiation With Single Interface* that supports different functionalities to users through **the same service** interface. It is realized by deriving multiple *Policy Configured Business Processes (PCBPs)* which share a single service interface.
  - *Service Differentiation With Multiple Interfaces* that supports different functionalities to users through **multiple** service interfaces. It is realized by generating multiple *Policy Configured Business Processes (PCBPs)* to support different service interfaces to users.

  Note that Business policies should be independent from the *Abstract Business Process (ABP)* to achieve higher manageability and reusability.
- *Policy Process Connector* that plugs the *Business policies* into the *Abstract Business Process*, hence derive *Policy Configured Business Process (PCBP)*.
- *Policy Configured Business Process (PCBP)* consists of two kinds of elements:



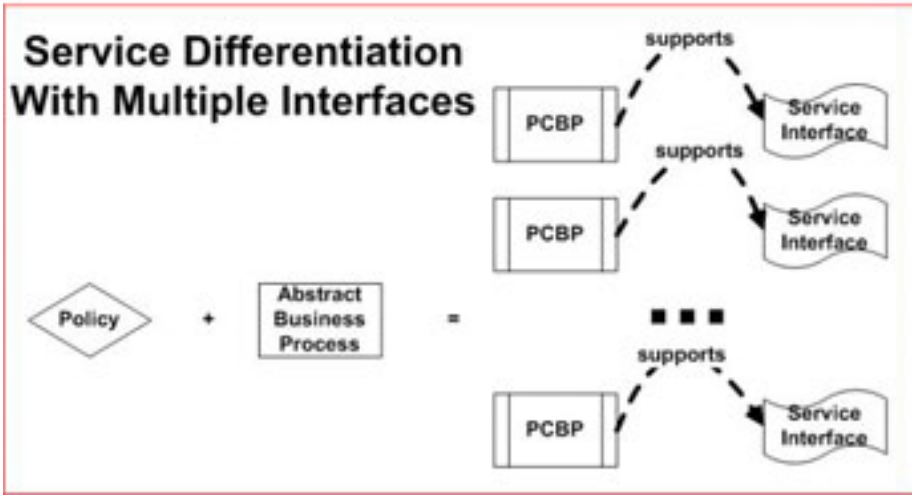




Figure 2: Service Differentiation With Single Interface & Service Differentiation With Multiple Interfaces

- Activity which delivers the common functionalities required by all users.
- *Policy* which delivers different functionalities or interaction patterns to meet different user needs.

We organize the rest of this section as follows:

- introduce Checkout process for OPS as the *Abstract Business Process*.
- demonstrate the example of *service differentiation with single interface* by differentiating Checkout process with the discount policy. Three PCBPs will be derived that provide different discount rates for different users through the same interface.
- demonstrate the example of *service differentiation with multiple interfaces* by differentiating Checkout process with the approval policy. Two PCBPs will be derived, two service interfaces will be derived correspondingly to apply different approving procedures on different users: one interface requires additional doctor confirmation for prescribed medicine purchase; the other one does not require anything for unprescribed medicine purchase.

### 4.1 Abstract Business Process

As the *Checkout Process* showed in Figure 3, the *Abstract Business Process* consists of three elements:

- A series of *activities* that perform well-defined business functions for *Abstract Business Process*. Each activity reads message(s) as input and returns message(s) as output.
  - The *Activities* in Checkout process are: loginAct, displayGoodSelectionAct, displayTotalPriceAct, receivePaymentAct, and deliveryAct.
- *Control flow* specifies how the set of activities is executed in terms of sequence, parallel, conditions, and coordination actions.
  - The *Control flow* in Checkout process executes the following activities in sequential order: loginAct, displayGoodSelectionAct, displayTotalPriceAct, receivePaymentAct, and deliveryAct.
- *Message*, on the other hand, describes the data been used in activities. The *public messages* are visible to external users in the service interface, while the *private messages* will be hidden from the users.
  - The *Message* in Checkout process sets all messages to public except the userDetailsMsg which is used by the loginAct.

```
AbstractBusinessProcess CheckoutProcess{

  Activities:
    Public:
      loginAct(userMsg, passwordMsg)
                          return userDetailsMsg ;
      displayGoodSelectionAct()
                          return goodListMsg ;
      displayTotalPriceAct(selectedGoodListMsg)
                          return priceMsg ;
      receivePaymentAct(paymentMsg);
    Private:
      deliveryAct();

  ControlFlow:
    T1 = Sequential(loginAct, displayGoodSelectionAct);
    T2 = Sequential(displayGoodSelectionAct, displayTotalPriceAct)
    T3 = Sequential(displayTotalPriceAct, receivePaymenAct);
    T4 = Sequential(receivePaymentAct, deliveryAct);

  Message:
    Public:
      userMsg, passwordMsg, goodListMsg,
      selectedGoodListMsg, priceMsg, paymentMsg;
    Private:
      userDetailsMsg;
}
```

Figure 3: The Checkout Process

### 4.2 Service Differentiation With Single Interface

As showed at the top of Figure 2, the *Service Differentiation With Single Interface* supports users with different functionalities through a single service interface. It is achieved by developing multiple *Policy Configured Business Processes* which share the same service interface. The advantage of *Service Differentiation With Single Interface* is that allows users to access the differentiated service through a single service interface, in this way the user development cost can be saved. On the other hand, *Service Differentiation With Multiple Service Interfaces* (bottom of Figure 2) requires users to access differentiated service through multiple service interfaces, thus increases extra user development cost. As a result, the *Service Differentiation With Single Interface* is always preferred than *Service Differentiation With Multiple Service Interfaces*. In this section, we will demonstrate an example of *Service Differentiation With Single Interface* by configuring the Checkout Process with Discount Policy(Figure 4).

#### 4.2.1 Policy

The *Policy* is separately and independently managed from the *Abstract Business Process*. Note that since *policy* is independent from the *Abstract Business Process*, the same *policy* can be plugged into different *Abstract Business Processes* through different *Policy Process Connectors* (Section 4.2.2). In this way the reusability and maintainability of both *Policies* and *Abstract Business Processes* can be achieved.

The policies design needs to be bounded with a set of activities that provide different functionalities to users. Depending on user data at runtime, the policy would be instantiated to be one of those activities to support user specific functionalities. As the Discount Policy showed in Figure 5, each policy consists of the following elements:

- *User data* that represents the user profile or behavior. A policy performs differently depending on the user data.
  - The Discount Policy performs different activities depending on the loyaltyData in order to differentiate the service.
- *Activities* that defines different functionalities could be performed by the policy. At runtime, policy would be instantiated as one of these activities depending on the user data. Note that the activities could connect with external business processes in order to perform complex functionalities. We only demonstrate the activities that perform simple functionalities in our example for illustration purpose.
  - The Discount Policy can be instantiated into one of these three activities based on loyaltyData: 10PercentDiscountAct, 15PercentDicountAct or noDiscountAct.
- *Conditions* which instantiates the *Policy* into different activities according to user data value at runtime, and in this way differentiated service can be achieved by performing different activities for different users.

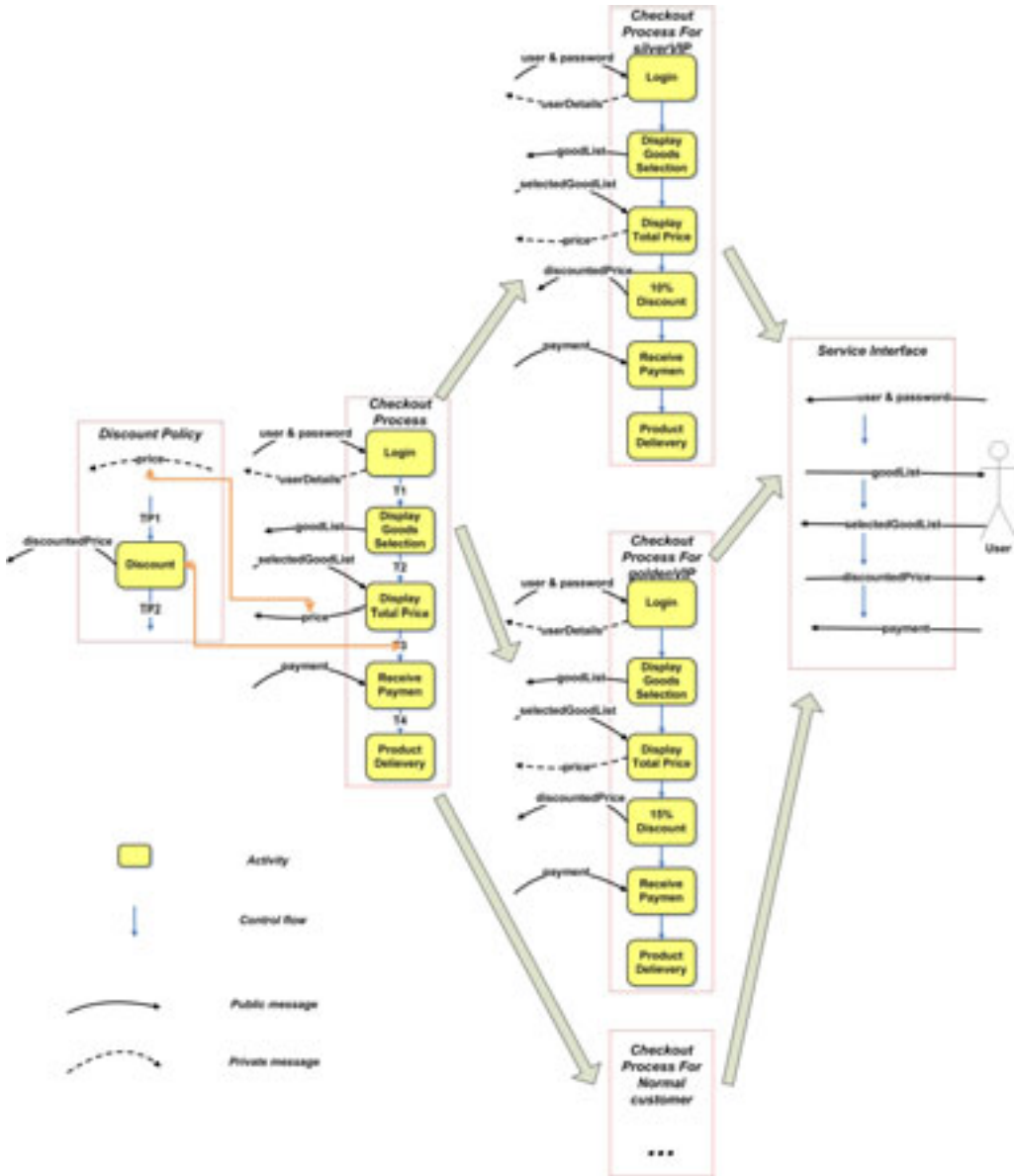



Figure 4: Configured Checkout Process & Service Interface by Discount Policy

```
Policy DisountPolicy
  UserData loyaltyData
{
  Activities:
    10PercentDiscountAct(priceMsg)
        return discountedPriceMsg;
    15PercentDicountAct(priceMsg)
        return discountedPriceMsg;
    noDiscountAct(priceMsg)
        return discountedPriceMsg;

  Conditions:
    If(loyaltyData = silverVip) 10PercentDiscountAct;
    Else if (loyaltyData = goldenVip) 15PercentDicountAct;
    Else noDiscountAct;
}
```

Figure 5: Discount Policy

- In Discount Policy, if the loyaltyData equals to silverVIP, then the policy will be instantiated as the activity 10PercentDiscountAct which outputs discountedPriceMsg as 10% off from the priceMsg received; else if the loyaltyData equals to the goldenVIP then the policy will be instantiated as the activity 15PercentDiscountAct which outputs discountedPriceMsg as 15% off from the priceMsg received; otherwise no discount is provided to normal customers.

#### 4.2.2 Policy Process Connector

Because *policies* are independent from the *Abstract Business Process*, for each *Policy*, a *Policy Process Connector* is required to plug the *policy* into a specific *Abstract Business Process*. As showed in Figure 6, the *Policy Process Connector* consists of six elements:

- *Policy* and *Abstract Business Process* that represent the *Policy Process Connector* is associated with.
  - DiscountCheckoutConnector is associated with DiscountPolicy and CheckoutProcess.
- *Policy Dependant User Data* which extracts the data that the policy depends on from the *messages* in *Abstract Business Processes*.
  - The loyaltyData can be extracted from the userDetailMsg in Checkout Process.
- *Policy Process Message Connection* handles the message flows between the *Policy* and *Abstract Business Process*.

```
PolicyProcessConnector DiscountCheckoutConnector{
  Policy :
    DiscountPolicy as dp;

  Process
    CheckoutProcess as cp;

  Policy Dependant User Data
    dp.loyaltyData = extractLoyalty(cp.userDetailMsg);

  Policy Pocess Message Connection
    dp.priceMsg = cp.priceMsg;

  ControlFlow :
    TP1: Sequential(cp.displayTotalPriceAct, dp);
    TP2: Sequential(dp, cp.receivePaymenAct);
    T3:  Sequential(TP1, TP2);

  Message :
    Public :
      dp.discountedPriceMsg;
    Private :
      dp.priceMsg;
      cp.priceMsg;
}
```

Figure 6: Discount Policy & Checkout Process Connector

**PolicyConfiguredBusinessProcess** `DiscountCheckoutProcess` =
**CoreBusinessProcess** `CheckoutProcess`
**configuredBy** **PolicyProcessConnector** `DiscountCheckoutConnector`

Figure 7: Discount Checkout Process

  - DiscountCheckoutConnector assigns the value of the priceMsg in Checkout Process into the priceMsg which is required in the Discount Policy.

- *ControlFlow* that determines the position in the *Abstract Business Process* which the *Policy* should be plugged into.

  - As showed in Figure 4, DiscountCheckoutConnector places the *Discount Policy* between the displayTotalPriceAct and receivePaymenAct in the Checkout Process by replacing the original T3 in Checkout Process with TP1 and TP2. TP1 and TP2 connect the Discount Policy with displayTotalPriceAct and receivePaymenAct.

- *Message* sets the message visibilities in the *Policy* and re-sets the message visibilities in the *Abstract Business Process* if necessary.

  - As showed in Figure 4, the *Message* sets the discountPriceMsg in the Discount Policy to public in order to display the discounted price to users, meanwhile re-sets the priceMsg in the Checkout Process to private in order to hide the original price from users.

In this section, we can see that *policy process connector* is fairly important because it does not only plug the *Policy* into the *Abstract Business Process*, but also controls service interface of the configured business process.

#### 4.2.3 Policy Configured Business Processes (PCBPs)

The PCBPs are determined by configuring the *Abstract Business Process* with *Policies*. The *service interface* derived from each *PCBP* is a directed sequence of message exchange. It can be determined by only showing the *public messages* and related *control flow* from the *PCBP*.

As showed in Figure 7, by configuring the *CheckoutProcess* with *DiscountCheckoutConnector*, we can see that three *Discount Checkout Processes* can be instantiated at runtime(Figure 4). Each *Discount Checkout Process* provides different discount activities to the users with different loyalty levels (e.g. normal, silver VIP and golden VIP). Further more, all the *Discount Checkout Processes* support the same service interface, we regard this as *Service Differentiation With Single Interface* that users do not need to aware of the service differentiation, they can access different functionalities through the same applications.

### 4.3 Service Differentiation With Multiple Interfaces

As showed at the bottom of Figure 2, the *Service Differentiation With Multiple Interfaces* supports users with different functionalities through multiple service interfaces. In this section, we shall demonstrate an example of *Service Differentiation With Multiple Interfaces* by configuring the Checkout Process with Approval Policy (Figure 8).

#### 4.3.1 Policy

As showed in Figure 9, we can see that the Medicine Approval Policy requires doctor confirmation from the medicine type which is prescribed. At runtime, depending on medicineTypeData, the Medicine Approval Policy can either be instantiated as checkConfirmationAct which requires doctorConfirmationMsg as input and approvalMsg as output, or be instantiated as noConfirmAct which does not require any message. Hence such policy results in two different interfaces (as showed in Figure 8):

- The top interface is for daily (un-prescribed) medicine purchase that does not require any approval.
- The bottom interface is for prescribed medicine purchase that requires doctor confirmation.

#### 4.3.2 Policy Process Connector

As showed in Figure 10, we can see that the *policy process connector* ApprovalPolicyConnector consists of the following elements:

- *Policy* and *Abstract Business Process* that associate Approval Policy with Checkout Process.
- *Policy Dependant User Data* that extracts medicineTypeData for Approval Policy from the selectedGoodListMsg in Checkout process.
- *ControlFlow* that places the DiscountPolicy between diplayGoodSelectionAct and displayTotalPriceAct in Checkout process (as showed in Figure 8).
- *Messages* that sets approvalMsg and doctorConfirmationMsg in the Approval Policy to be public.

#### 4.3.3 Policy Configured Business Processes

As showed in Figure 11, we can see that the *ApprovalCheckoutProcess* can be derived by configuring the *CheckoutProcess* by *ApprovalPolicyConnector*. Note that, the Approval Policy can be instantiated into different activities which have different input and output messages at runtime: checkConfirmationAct which requires additional message as doctor

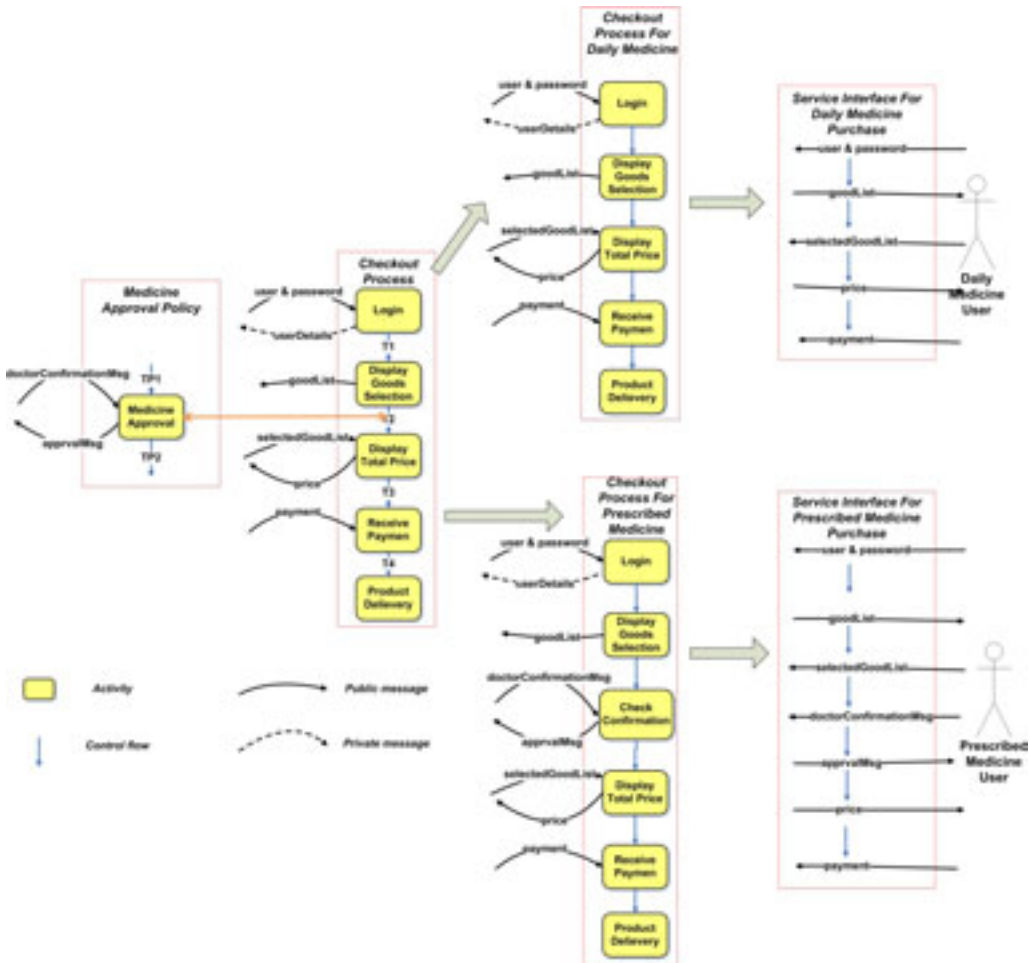



Figure 8: Configured Checkout Process & Service Interface by Approval Policy

```
Policy MedicineApprovalPolicy
  UserData medicineTypeData
{
  Activities:
    checkConfirmationAct(doctorConfirmationMsg)
                          return approvalMsg;
    noConfirmAct();

  Conditions:
    If(medicineTypeData.is(prescribed))
      checkConfirmationAct(doctorConfirmationMsg);
    Else noConfirmAct();
}
```

Figure 9: Approval Policy

```
PolicyProcessConnector ApprovalCheckoutConnector{

  Policy:
    ApprovalPolicy as ap;

  Abstract Business Process:
    CheckoutProcess as cp;

  Policy Dependant User Data
    ap.medicineTypeData
       = checkMedicine(cp.selectedGoodListMsg);

  ControlFlow:
    TP1: Sequential(cp.diplayGoodSelectionAct, ap);
    TP2: Sequential(ap,cp.displayTotalPriceAct);
    T3:  Sequential(TP1, TP2);

  Message:
    Public:
      ap.approvalMsg, ap.doctorConfirmationMsg;
}
```

Figure 10: Approval Policy & Checkout Process Connector

```
PolicyConfiguredBusinessProcess ApprovalCheckoutProcess=
  CoreBusinessProcess CheckoutProcess
  configuredBy PolicyProcessConnector ApprovalPolicyConnector
```

Figure 11: Approval Checkout Process

confirmation for the prescribed medicine purchase, or noConfirmAct requires nothing for daily medicine purchase. Because the checkConfirmationAct and noConfirmAct have different interaction patterns, two different *Policy Configured Business Processes* could be generated at runtime which support users through different service interfaces (Figure 8). We regard this as *Service Differentiation With Multiple Interfaces* that users need to aware of the service differentiation, and develop different applications to access the differentiated service through different service interfaces.

## 5 Conclusion

Service users or applications often have different interaction or service outcome requirement from a business service. In this paper, we proposed and argued the need for *service differentiation*. We believe multiple interface(s) supported by different underlying business processes should be provided for the same service. The main contribution of the work lies in the fact that policies are independently managed from the underlying business process and are used to generate policy configured business processes from the abstract business process. Therefore the maintenance efforts can be shifted from managing business process to managing policy when business policy and rule change.

From our experience, *if possible, service differentiation with single interface (e.g. Discount Checkout Processes) is preferred than the service differentiation with multiple interfaces (e.g. Approval Checkout Process)*, in this way users only need to develop single application to access differentiated service through the same service interface rather than develop several applications to access differentiated service through different service interfaces.

Future work will be carried out in the following areas:

- *service description:* we are currently investigating a way to incorporate our approach into BPEL by extending BPEL with the *Policy* and *Policy Process Connector*. In this way differentiated service can be described properly.
- *service interface binding:* we also focus on the way to bind an interface with user (application) behavior during service invocation time.

## References

[AAF02] Assaf Arkin, Sid Askary, Scott Fordin, Wolfgang Jekeli, Kohsuke Kawaguchi, David Orchard, Stefano Pogliani, Karsten Riemer, Susan Struble, Pal Takacsi-Nagy, Ivana Trickovic, Sinisa Zimek. Web Service Choreography Interface (WSCI) 1.0. http://www.w3.org/TR/wsci/

[ACD03] Tony Andrews, Francisco Curbera, Hitesh Dholakia, Yaron Goland, Johannes Klein, Frank Leymann, Kevin Liu, Dieter Roller, Doug Smith, Satish Thatte, Ivana Trickovic,

Sanjiva Weerawarana. Business Process Execution Language for Web Services version 1.1 http://www-128.ibm.com/developerworks/library/specification/ws-bpel/

[AN04] A.H. Anderson, An introduction to the Web Services Policy Language (WSPL). *in Policy 2004, IEEE, P. 442*

[BBC06] Siddharth Bajaj, Don Box, Dave Chappell, Francisco Curbera, et al. Web Services Policy (WS-Policy) *1.5, Nov. 2006. www.w3.org/TR/ws-policy*

[CCK02] Dickson K. W. Chiu, Shing-Chi Cheung, Kamalakar Karlapalem, Qing Li, and Sven Till. Workflow View Driven Cross-Organizational Interoperability in a Web-Service Environment. *WES 2002: 41-56*

[CCM01] Erik Christensen, Francisco Curbera, Greg Meredith, Sanjiva Weerawarana. Web Services Description Language (WSDL) 1.1. http://www.w3.org/TR/wsdl

[CHT03] Kishore Channabasavaiah, Kerrie Holley and Edward Tuggle. Migrating to a service-oriented architecture, *IBM DeveloperWorks, 16 Dec 2003*

[MBD04] David Martin, Mark Burstein, Grit Denker, Jerry Hobbs, Lalana Kagal, Ora Lassila, Drew McDermott, Sheila McIlraith, Massimo Paolucci, Bijan Parsia, Terry Payne, Marta Sabou, Evren Sirin, Monika Solanki, Naveen Srinivasan, Katia Sycara. OWL-S: Semantic Markup for Web Services. http://www.w3.org/Submission/-OWL-S/

[MFN05] Sonia Ben Mokhtar, Damien Fournier, Nikolaos Georgantas, Valrie Issarny. Context-Aware Service Composition in Pervasive Computing Environments. *RISE 2005: 129-144*

[PY02] Mike P. Papazoglou, Jian Yang. Design Methodology for Web Services and Business Processes, *TES 2002: 54-64.*

[RFC2475] RFC247: An Architecture for Differentiated Services. http://rfc.net/rfc2475.html

[RLK04] Dumitru Roman, Holger Lausen, Uwe Keller, Eyal Oren, Christoph Bussler, Michael Kifer, Dieter Fensel. Web Service Modeling Ontology (WSMO). http://www.w3.org/Submission/-WSMO

[TPP05] Vladimir Tosic, Bernard Pagurek, Kruti Patel, Babak Esfandiari, Wei Ma Management applications of the web service offerings language (WSOL) *Information Systems, Volume 30 , Issue 7 (November 2005), 564 - 586*

[TY07] Aries Tao Tao, Jian Yang. Supporting Differentiated ServicesWith Configurable Business Processes *International Conference on Web Services (ICWS), 2007*

[VE00] Richard Veryard Design Pattern: Differentiated Service (Fewer Interfaces than Components) *CBDI Journal, 1, Dec, 2000.*

[WS] Web Service. http://www.w3.org/2002/ws/

[ZLY05] Xiaohui Zhao, Chengfei Liu, and Yun Yang. An Organisational Perspective on Collaborative Business Processes, *Business Process Management 2005: 17-31.*