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

Service Oriented Computing (SOC) allows resources on a network to be made available as services. For a business service, differentiated services can be provided based on the usage context, i.e., location, age, purpose and user profiles. In differentiated services, service outcomes depend on context. Currently there is no efficient technical solution for supporting differentiated service development. In this paper we present an approach to deliver differentiated services realized by configurable business processes so that service flexibility, manageability and reusability can be achieved.

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[4]. To better understand how to design and develop services, it is important to understand the relationship between service, service interface, and business process as shown in Figure 1-(a):

- A service is a business concept that should be specified with an application or the user of the service in mind[13].
- Service interface is the specification for user (or program) to interact with the service. It is supported by business process(es). It consists of user visible activities which are derived from the corresponding business process(es).
- The business process(es) implements the functionality of service. The business process consists of activities that perform service functions. There are two kinds of activities in business process:
  - User invisible activities that are hidden from service interface in order to hide detailed business logics and certain implementation details from the users.
  - User visible activities that are presented in service interface and used for user/application interactions.

Quite often we find that the users may require differentiated functionalities from the same service depends on the usage contexts such as location, age, and purpose. Take Online Pharmacy Service (OPS) as an example, for loyal customers, we want to offer them extra 10% discount; for people who purchase prescribed medicine, we need to check if they have valid prescription. Therefore, there is a need for providing different business processes to support differentiated services and service interfaces. Implementing a thin SOAP[21], WSDL[25] and UDDI[22] layer on top of existing applications or components that realize the services is not enough to build configurable, manageable and re-usable services. To support different outcomes, the service needs to have different business processes and provide multiple service interfaces as responses to different user needs.

In current service design approaches as summarized in Figure 1-(a), usage context is hard coded in the business process as different execution conditions. As a result, services are supported by fixed business processes which provides the same functionalities to all the users from the same service interfaces. Such approach limits the flexibility and extensibility of the service offerings.

In order to mend the situation, we propose a new service design approach that separate the generic business activities that are applicable to all the circumstances from those only applicable to particular group of people or under specific conditions. As showed in Figure 1-(b) our design is based on the following ideas:

- Use a "core business process" to support the activities that are generic and required by all the users regardless of the context differences.
- Separate the usage contexts from the core business process. Different user groups or purpose of usage can be modelled as a set of usage contexts.
Figure 1. Current & Proposed Service Design Approach
• By applying different usage contexts to the core business process will generate different context configured business processes.

• Service interfaces are determined by the context configured business processes.

As the context configured business processes can provide different functionalities to different users via multiple service interfaces, service flexibility and adaptability can be achieved because any changes happened to the context related policies, they will be reflected to the underlying supporting business processes automatically.

This paper is organized as follows: Section 2 will discuss the related work. The case study will be introduced in Section 3. Section 4 will demonstrate the proposed service design and development. We finally concludes our work in section 5.

2 Related Work

In this section we will discuss related work from four aspects. Firstly we shall take a general look into the research work in context. Secondly we review the current state of arts of research in usage context in SOC area. Thirdly we review the current service describing techniques, arguing that they can not effectively describe differentiate services. Finally we analyse the related Web Service Standards which can be used for our work.

The context is widely applied in computing area: Want et al [16] has introduced an Active Badge system forwards phone calls according to the user context such as location. Abowd et al [1] made use of context to implement a tourist guide system called Cyberguide. Cheverst et al [5] also made use of context in their tour guide application called “the GUIDE project”. By using an object-oriented approach, the Context Toolkit [8] provided a framework and a number of reusable components to support rapid prototyping of sensor-based context-aware applications.

The importance of applying contexts to SOC has been stated in following literatures: Baldauf, Dustdar and Rosenberg stated the tight association between context and service in their survey [3]. Maar et al [11] clearly stated the needs of contexts in order to derive personalized service. They also classified the context into several catalogues in terms of their effect to the service. Hong, Chiu and Shen [10] proposed a model to analyse the effect of contexts on service. Gu, Pung and Zhang [9] proposed a formal context model based on ontology using Web Ontology Language to address issues including semantic representation, context reasoning, context classification and dependency. However, no methodology has been provided to specify how to configure business process based on usage contexts, based on which differentiated services can be provided. Our work complements their work by providing a working mechanism to derive context configured business processes for a business service.

Work has been done in the area of specifying public interface for business processes. Chiu et al [7] 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 certain subset of Service, it allows internal information to be hidden from external users. However, the work only focused on abstract a single service interface which did not take usage context into consideration. To support different user groups, Zhao, Liu and Yang [18] 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 constraint every time for each new service user, therefore the system does not scale well. Using a completely different strategy, our work allows service interface generated from configured business processes, which are derived automatically based on usage contexts. As a result, the maintenance effort is shifted from managing business processes to managing usage contexts.

Now let us have a close look of current relevant web service standards. WSCI [24] allows the service to be described as a sequence of Web Service calls binding to WSDL [25]. In advance, BPEL [19] allows the service interface to be described as an abstract business process, which is a subset of BPEL process. BPEL allows several abstract business process to be derived for one service. However, BPEL specification does not provide any mechanism or standard to generate multiple service interfaces, which is often required by business as illustrated in our example. Several Semantic Web Service Description standards such as OWL-S [20], WSMO [26] has been proposed. Comparing with WSDL and BPEL, Semantic Web Service provides better support in common understanding of context semantics and reasoning on complex relations between contextual concepts [12]. They have a common way to describe service: (1) A Service Process Model to abstract internal processes, which describe how the service work. (2) A Service Grounding to describe how the service should be accessed. (3) A Service Profile contains the non-functional parameters, which describe what the service does. However, each service is designed to have only one Service Process Model only. Such design limits the service flexibility to support 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. As a result different users can access the service in different ways.

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 context configured business processes which provide different functionalities to different customers (e.g. further discount for loyal customer) via multiple service interfaces. For space limitation, we only use the Checkout process of OPS as an example.

The Checkout business process consists of 5 activities:

- **Login** - receives the user name and password in order to identify the user.
- **Display Goods Selection** - displays a list of goods selected by the user.
- **Display Total Price** - displays the total prices the customer needs to pay for the goods.
- **Receive Payment** - receives payment from user and return invoice.
- **Product Delivery** - contacts delivery company to deliver the product to customers.

The Checkout process provides basic functionalities that is required by all customers. However, the functionalities it supports is not enough for the loyal customers nor for the customers who use prescribed medicines because:

- For loyal customers, they can receive 10% discount on the total amount.
- For prescribed medicine (e.g. drugs) buyer, they need to provide doctor prescriptions.

Instead of hard-coding all the information into one checkout business process as most current service developments do, we separate the usage context from the business process. In the next section we will show how a Context Configured Business Process is determined for loyal customers who intend to buy prescribed medicines.

4 Differentiated Service Design

The aim of this section is to develop service interface for a group of people, e.g. loyal customers who intend to buy prescribed medicines. Each service interface is supported by a Context Configured Business Process (CCBP) that provides different interactions and outcomes from other business processes for the same service. As showed in Figure 1-(b), there are four elements in the design:

- **Core Business Process (CBP)** implements the generic functionalities required by all the users for all purposes.
- The set of **Usage Contexts** are complementary to the Core Business Process. They specify functionalities which are only required by specific users for specific purposes.
- **Context Configured Business Process (CCBP)** implements the functionalities needed by a specific user group. Applying different usage contexts over Core Business Process will result in different CCBPs.
- **Service Interface** acts as an interaction protocol. Each Service Interface is supported by one CCBP. It allows users to interact with the service based on their usage contexts.

Therefore four steps in service development can be identified based on the above discussion:

- **Core Business Process (CBP) Determination** models the Core Business Process.
- **Usage Contexts Creation** creates usage contexts for different user groups. Each user group can be modelled by a relevant set of usage contexts.
- **Context Configured Business Process (CCBP) Generation** generates CCBPs based on the Core Business Process and the each set of usage contexts.
- **Service Interface Derivation** derives Service Interfaces based on CCBPs. Note: one service interface corresponds to one CCBP.

4.1 Core Business Process (CBP) Determination

The Core Business Process implements the functionalities that are provided to all the users. The Core Business Process can be modelled by Finite State Machine (FSM) [17] which consists of two elements:

1. **State** corresponds to an Activity and the Preconditions that are associated with the Activity.

- **Activity** performs the functionalities of the Core Business Process, to which we associate two elements:
  - **Operation** which calls the associated functions.
  - **Visibility** which classifies the Activity as "User Visible Activity" or "User Invisible Activity".
2. Transition which connects between two states as departure state and destination state.

We can give the formal definition of a Core Business Process as follows:

- **Definition 1**: \( \text{CBP} = (S, T, S_0, S_F) \).
  - \( S \) is a finite set of states (E.g. \( S_i \)).
  - \( T \) is a finite set of transitions (E.g. \( T_i \)).
  - \( S_0 \) is the initial state of the CBP.
  - \( S_F \) is the final state of the CBP.

- **Definition 2**: A state \( s = (\text{Cond}, \text{Activity}) \) where \( s \in S \).
  - **Definition 3**: Activity = (Operation, Visibility)
    - \( \text{Cond} \) represents a set of preconditions. Nil-precondition can be represented as \( \emptyset \).

- **Definition 4**: Transition set \( T \subseteq S \times S \)

  Showed as the second column on the left of Figure 2, CBP of OPS can be represented as follows:

  - **OPS.CBP** = (OPS.S, OPS.T, OPS.S0, OPS.SF).
  - **OPS.S** contains 5 states:
    - OPS.S0 = (\( \emptyset \), ("Display Goods Selection", Visible))
    - OPS.S1 = (\( \emptyset \), ("Display Total Price", Visible))
    - OPS.S2 = (\( \emptyset \), ("Receive Payment", Visible))
    - OPS.SF = (\( \emptyset \), ("Product Delivery", Invisible))

  - **OPS.T** contains 4 transitions:
    - \( T_0 = (\text{OPS.S}_0, \text{OPS.S}_1) \)
    - \( T_1 = (\text{OPS.S}_1, \text{OPS.S}_2) \)
    - \( T_2 = (\text{OPS.S}_2, \text{OPS.S}_3) \)
    - \( T_3 = (\text{OPS.S}_3, \text{OPS.S}_F) \)

4.2 Usage Contexts Creation

As complementary to Core Business Process, different sets of usage contexts represent users with different properties. Each context consists of two parts:

- **Property** identifies a user relevant feature (e.g. location, interests, and behaviours) which distinguishes different user groups. Each Property associates one Precondition with certain state of the Core Business Process:
  - The **State** in the Core Business Process is where the Property needs to be checked.
  - The **Precondition** is the condition that needs to be satisfied by users in order to meet the requirement of the Property.

- **Effect** shows the impacts of the Property on the Core Business Process. It differentiates the Core Business Process once the corresponding Property requirement is satisfied. The Effect of one context can be further classified into three categories:
  - Adding a new state in certain place of Core Business Process to perform additional functions. For an example, providing customer with an additional "Discount" Activity.
  - Replacing a existing state in Core Business Process with a new state.
  - Removing an existing state from Core Business Process. For an example, the "Product Delivery" activity could be removed if the clients choose to pick up the medicines from the store.

Therefore the each usage context can be modelled as follows:

1. **Definition 5**: Context = (Property, Effect).
2. **Definition 6**: Property = (S., Cond).
3. The Effect supports following three kinds of operations:
   
   • (Add, $S_i$, T) adds a new state in certain place of Core Business Process.
   
   - $S_i$ is the new state needs to be added to Core Business Process.
   
   - T is a list of Transitions that connected $S_i$ with Core Business Process.
   
   • (Replace, $S_{old}$, $S_{new}$) replaces an existing state in Core Business Process with a new state.
   
   - $S_{old}$ is the old state needs to be replaced.
   
   - $S_{new}$ is the new state to replace $S_{old}$.
   
   • (Remove, $S_i$) removes an existing state from Core Business Process.
   
   - $S_i$ is the state to be removed.

As showed in the left column of Figure 2, the usage contexts of OPS can be modelled as follows:

1. $\text{OPS}_{UCM} = \{\text{context}_1, \text{context}_2\}$ models loyal customers who use prescribed medicine.

2. $\text{context}_1 = (\text{Property}_1, \text{Effect}_1)$ is used to model "10% discount for loyal customers".
   
   • $\text{Property}_1 = (\text{OPS}_S_{0}, \text{Cond}_1)$
   
   - $\text{Cond}_1 = \text{isLoyalCustomer}$
   
   • $\text{Effect}_1 = (\text{Add}, \text{OPS}_S_4, \text{T}_\text{list}_4)$
   
   - $\text{OPS}_S_4 = (\emptyset, ("10\% Discount", \text{Visible}))$
   
   - $\text{T}_\text{list}_4 = \{(\text{OPS}_S_2, \text{OPS}_S_4), (\text{OPS}_S_4, \text{OPS}_S_3)\}$

3. $\text{context}_2 = (\text{Property}_2, \text{Effect}_2)$ is used to model "Extra approval activity for prescribed medicine (e.g. drugs) buyer".
   
   • $\text{Property}_2 = (\text{OPS}_S_1, \text{Cond}_2)$
   
   - $\text{Cond}_2 = \text{isPrescribedMedicineUser}$
   
   • $\text{Effect}_2 = (\text{Add}, \text{OPS}_S_5, \text{T}_\text{list}_5)$
   
   - $\text{OPS}_S_5 = (\emptyset, ("Prescribed Medicine Approval", \text{Visible}))$
   
   - $\text{T}_\text{list}_5 = \{(\text{OPS}_S_1, \text{OPS}_S_5), (\text{OPS}_S_5, \text{OPS}_S_2)\}$

4.3 Context Configured Business Process (CCBP) Generation

Context Configured Business Process (CCBP) is determined by applying different usage contexts on the Core Business Process. CCBP can be specified in the same manner as Core Business Process:

$$\text{addContexts :: CBP \times UCM -> CCBP}$$

$$\text{addContexts CBBP} \ = \ \text{CBBP}$$

$$\text{addContexts CBP context_list}$$

$\quad \text{addContexts CBP context_list}$

$\quad \quad \text{where}$

$\quad \quad \text{c_rest} = \text{init (context_list)}$

$\quad \text{context_1 = addContext CBP c_rest}$

$\quad \text{context_2 = addContext CBP c_rest}$

$\text{Figure 3. Algorithm for Context Configured Business Process (CCBP) Generation}$

- Definition 7: CCBP = $(S, T, S_0, S_F)$

In this paper we choose Haskell pseudo code to represent service development related algorithms. The reason of choosing Haskell is because it is a functional language, which can naturally represent Set related mathematical calculations. Showed below are some basic functions in Haskell, for full details please visit (http://haskell.org/):

- The operand ":=" is used to add an item to a list
- The operand "+" is used to add a list to another list.
- The operand "|" is used as a guard, which is equivalent to "if, else if".
- The operation "head" is used to pop out the first item from a list
- The operation "init" is used to extract a sub list from a list, which excludes the first item.

The algorithm for CCBP generation works as follows:

- In order to derive a CCBP, the corresponding usage contexts are applied to the Core Business Process.

- An help function can be used to apply a single context to the Core Business Process. In this help function, the property of context will be checked against a certain state, the effect of the context will then be executed.

Due to space limitation, only part of the code is showed in Figure 3:
4.4 Service Interface Derivation

Once CCBP is generated, the corresponding Service Interface can be derived by extracting all the User Visible Activities from the CCBP. Users therefore shall be able to interact with CCBP via User Visible Activities specified in Service Interface.

- **Definition 8:** ServiceInterface = (S, T, S₀, S_F).

The algorithm of Service Interface Derivation checks each state from the CCBP, and keeps track of all the User Visible Activities to construct the Service Interface. Part of the code is showed in Figure 4:

- **ServiceInterfaceDetermination** is the main function to derive the Service Interface.
- **removeInvisibleStates** is the help function for ServiceInterfaceDetermination to remove all the states contains the User Invisible Activities.
- **removeOneInvisibleState** is the help function for removeInvisibleStates. If one state contains User Invisible Activity, it removes the state from the CCBP.
- **isUserInvisibleState** is the help function for removeOneInvisibleState. It returns True if a state contains "User Invisible Activity", otherwise returns false.

Showed as the right column of Figure 2 is the determined OPS_ServiceInterface.

5 Conclusion

In this paper we present a mechanism for developing differentiated service, in which service outcomes are based on usage contexts. The main contribution of the work lies in the fact that usage contexts are separated from the underlying business process and are used to generate context configured business processes from the core business process. Therefore the maintenance efforts can be shifted from managing business process to managing usage contexts when context related business policy and rule change.

A formal model and its related algorithms for the differentiated service development are presented. We are currently investigating a way to incorporate our approach into OWL-S so that the differentiated services can be described properly. We are also look into the issues in modelling and specifying policy and rules to manage usage contexts.
References


