Towards Adaptive Service Development

Aries Tao Tao
supervised by Jian Yang
{tao, jian}@ics.mq.edi.au

Computing Department, Macquarie University, Australia
http://www.mq.edu.au

Abstract. In the dynamic e-Business environment, it is desirable for a service to meet the requirements of different users. The current available technologies rather supports a service with a single fixed business process without considering user needs. Such design makes it difficult for user to integrate the provided service, hence obstruct the service provider to expand the business. In this paper we proposed an alternative service design method - Adaptive Service Design. Inspired by the concept of *Abstraction* and *Polymorphism* in Object Oriented Computing, service adaptation allows an Abstract Business Process class to be configured by Policy and user required interface, hence dynamically generate multiple business processes to meet different user interaction requirements.

Key words: service differentiation, service adaptation, matchmaking

1 Introduction

While improvement in Service Oriented Computing (SOC) has proved effective for integration at lower levels (e.g. wrap traditional software component up to be service), service integration through service interface is till a challenge due to the heterogeneity of service specification. Service interface is a specification for user (client program) to interact with the service. It is supported by underlying business process(es), and consisted of activities which are implemented by corresponding tasks of the business processes.

In e-business world, it is common for a set of service consumer with similar service requirements to have different interfaces. The differences could be caused either by user contexts [1] **(1) at business level** such as different physical locations, market conditions, policy regulations, competitive threats, or by service implementation **(2) at application level**s such as different message formats or sequences. All of these situations and more drive the need for a service design which enable the service provider to respond quickly in support of diverse service users' requirements.

On the other hand, ignoring the user heterogeneity, the current service is designed to be context free which supports the fixed monolithic business process and interface to all users. This inflexible design makes it difficult for service consumer to perform service integration with the provided service, hence obstruct

the service provider to expand the business. SOC research community has been aware of the issue. The solutions are proposed which mainly use process matchmaking [2] to identify the mismatch patterns, and adapter [3] to overcome the mismatch. However, as functionality of adapter is to split/combine messages or swap message sequences, such solution only resolve the certain mismatch at application level.

To overcome the problem with current service design, we propose a new design that supports policy negotiation, process modification, and interface alternation according to use needs, hence cater service to quickly adapt to diver service consumer requirements. We refer this design as Adaptive Service Design (ASD). An *User Adaptation* mechanism is developed based on ASD which enables the service provider to:

- identify user functionality and interaction requirements from the user's interface.
- dynamically generate the business process and service interface to meet the specific requirements.

The basic design philosophy of ours, and one that distinguishes us from others, is that the ASD supports service provider to easily vary the policy, business process and service interface to dynamically adapt different user integration requirements in terms of functionality and interaction patterns.

This paper is organized as follows: Section 2 discusses the related work. The structure of *Adaptive Service Design (DSD)* are specified in Section 3. We finally concludes our work in section 4.

2 Related Work

The related work can be divided in following areas: (1) *Service Description* which focuses on enriching the service interface for user accessibility; (2) *Service Differentiation* which supports services to provide different functionalities to users; (3) *Service Matchmaking* which allows the mismatches between provided service and user required services to be identified; (4) *Service Adaptation* overcomes the mismatches identified by the *Service Matchmaking*.

We now look at the work that has been done in the area of service interface design. Chiu et al [4] presented a meta-model for service interface as workflow views, which provided a novel approach to derive workflow view from a workflow. By abstracting service interface as a subset of service, it allows internal information to be hidden from external users. However, the work only focused on abstracting a single service interface. In order to support users playing different role in Business Collaboration, Zhao et al [5] proposed the concept of *relative workflow view* by explicitly specifying visibility constraints (Invisible, Traceable,and 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 that have different relationship with the service (e.g. the retailer service has different relationship with customer and wholesaler).

The idea of applying the concept of *differentiation* in software filed was firstly proposed by Kang et al [6] in the study of Feature-Oriented Domain Analysis which was based on Abstraction and Refinement in a domain. In their work, *Abstraction* represented generic domain products; *Refinement* was used to extend the *Abstraction* to support different domain applications. Cao et al [7] further extended the work in Feature-Oriented Domain Analysis by providing an algorithm to automate abstraction refinement. The idea of service differentiation (DiffServ) was firstly proposed in the area of managing traffic streams in networking applications [8]. For example, certain traffic is treated better than the others in terms of faster handling, more average bandwidth, and lower average loss rate. Veryard [9] argued that the differentiated services should be used as a design pattern in SOC area. He pointed out the need for service differentiation in E-business using an airport example - the airport service needs to meet different requirements of passengers in terms of security, performance and etc. However, no design method is provided to realize the service differentiation. In our previous work [10] [1], a Differentiated Service Design that use policy to control the service to provide differentiated functionalities to users, hence realize service differentiation.

Wombacher et al [2] used finite state machine based model to describe service interface, a matchmaking algorithm was provided based on the model to identify if provided interface is compatible with the user required interface. However, simply showing compatible (or incompatible) is insufficient to help resolve the mismatches. Benatallah et al [11] [12] further classified the compatibility into several categories. Aalst et al [13] define the conformance of service behavior based on fitness and appropriateness. All these work provides a foundation to service adaptation which overcomes the incompatibilities identified.

Based on the matchmaking research work been done, Benatallah et al [14] [15] [3] developed adapter based mechanisms that split/combine messages or swap message sequences to resolve certain types of mismatches. Because the adapter is developed only based on the interfaces, it can not support policy negotiation or process modification to overcome mismatches at policy or process level. Using a totally different approach, we propose Adaptive Service Design (ASD) for service provider. The ASD supports policy negotiation and process modification to overcome the mismatches that can not be resolved by adapters.

3 Adaptive Service Design

Instead of supporting a service with one monolithic business process and single service interface, our design method is to separate the generic tasks that are available to all users from the specific tasks with certain context dependent requirements. The design consists of four components: Abstract Business Process, Policy, Policy Configured Business Process (PCBP) and User Oriented Business Process (UOBP).

Abstract Business Process (ABP) consists of a set of activities and relevant edges linking between activities. There are two types of activities: (1) *concrete*

activity that actually performs general tasks for all users; and (2) *abstract activity* that executes different tasks or even execute different processes depending on *Policy*. Abstract activities are linked to *Policy* which defines how contexts aware tasks (or processes) should be performed.

Policy provides the mappings between usage contexts and tasks (or external business processes). Take online shopping service as an example, in the Checkout process, the "customer profile" as an usage context can be retrieved from the "login" activity. By applying 'provide discount' policy, "VIP customer" will get further 15% discount by invoking the 'offer 15% discount' task. Depending on the usage context values, the policy determines how to plug different tasks (or business processes) into the ABP, and generate *Policy Configured Business Processes (PCBP)* which perform different business functions.

After introducing the basic elements of *ABP* and *Policy* in the previous sections, we can now provide a complete picture of how a 'concrete' business process - *Policy Configured Business Process (PCBP)*, is generated. As discussed before, an *ABP* is a process that contains *abstract activities*, which refer to policy templates. A template consists of a set of mappings between context values and business processes (or tasks in a simple situation). The number of tuples (mappings) in a template determines the number of tasks can be performed for the corresponding *abstract activity*. By replacing all the *abstract activities* in *ABP* with policy specified tasks that come with different context conditions, multiple *PCBPs* can be generated. *PCBPs* support users with different functions, and thus realizes differentiated services based on usage contexts. For example, in Checkout process, the loyal customer and normal customer will be supported by different *Context Configured Business Processes*. We regard the process the generate different PCBPs as Service Differentiation. The details of the Service Differentiation can be found in our previous work [1].

Currently we are focus on constructing a mechanism to derive *User Oriented Business Process (UOBP)* based on PCBP for User Required Interface (URI). Each UOBP corresponds to one URI, it is totally compatible with the specific URI. The mechanism consists of three steps:

1. Generating User Service Interface Execution Paths. In this step a infinite set of execution paths [2] can be derived from the user service interface.
2. Each execution path will be matched by the PCBP. If there is any mismatch, the mismatch would be identified as one of followings: message mismatch, process mismatch, and policy mismatch. Solutions as adapter development, process modification and policy negotiation would be suggested correspondingly to resolve the mismatch. An Adaptation Effort List is hence derived to illustrate the total amount work needs to be done in order to adapt the specific execution paths.
3. An Adaptation Effort Tree can be derived by combining all the Adaptation Effort Lists for the execution paths. The tree illustrate the total work needs to be done to adapt the how service interface. User could also choose only certain branches of the Adaptation Effort Tree to partially adapt the user service interface.

4 Conclusion

Service users or applications can often have the same goal but different interaction requirements. In this paper, we proposed and argued the need for *service adaptation* to serve the purpose mentioned above. We believe the adaptive service should dynamically derive multiple User Oriented Business Processes for different users required interfaces. The design is related with following research areas: service description, service differentiation, service matchmaking and service adaptation.

References

1. Tao, A.T., Yang, J.: Context aware differentiated services development with configurable business processes. In: EDOC, IEEE Computer Society (2007) 241–252
2. Wombacher, A., Mahleko, B., Neuhold, E.J.: Ipsi-pf - a business process matchmaking engine based on annotated finite state automata. Inf. Syst. E-Business Management **3**(2) (2005) 127–150
3. Nezhad, H.R.M., Benatallah, B., Martens, A., Curbera, F., Casati, F.: Semi-automated adaptation of service interactions. In Williamson, C.L., Zurko, M.E., Patel-Schneider, P.F., Shenoy, P.J., eds.: WWW, ACM (2007) 993–1002
4. Chiu, D.K.W., Cheung, S.C., Karlapalem, K., Li, Q., Till, S.: Workflow view driven cross-organizational interoperability in a web-service environment. In Bussler, C., Hull, R., McIlraith, S.A., Orlowska, M.E., Pernici, B., Yang, J., eds.: WES. Volume 2512 of Lecture Notes in Computer Science., Springer (2002) 41–56
5. Zhao, X., Liu, C., Yang, Y.: An organisational perspective on collaborative business processes. In van der Aalst, W.M.P., Benatallah, B., Casati, F., Curbera, F., eds.: Business Process Management. Volume 3649., Springer (2005) 17–31
6. Kang, K.C., Cohen, S.G., Hess, J.A., Novak, W.E., Peterson, A.S.: Feature-oriented domain analysis (foda) feasibility study. Technical report, Carnegie-Mellon University Software Engineering Institute (November 1990)
7. Cao, F., Bryant, B.R., Burt, C.C., Huang, Z., Raje, R.R., Olson, A.M., Auguston, M.: Automating feature-oriented domain analysis. In Al-Ani, B., Arabnia, H.R., Mun, Y., eds.: Software Engineering Research and Practice, CSREA Press (2003) 944–949
8. Blake, S., Black, D., Carlson, M., Davies, E., Wang, Z., Weiss, W.: Rfc247: An architecture for differentiated services. Available from: http://rfc.net/rfc2475.html (1998)
9. Veryard, R.: Design pattern: Differentiated service (fewer interfaces than components). CBDI (December 2000)
10. Tao, A.T., Yang, J.: Supporting differentiated services with configurable business processes. In: ICWS, IEEE Computer Society (2007) 1088–1095
11. Benatallah, B., Casati, F., Ponge, J., Toumani, F.: Compatibility and replaceability analysis for timed web service protocols. In Benzaken, V., ed.: 21èmes Journées Bases de Données Avancées, BDA 2005, Saint Malo, 17-20 octobre 2005, Actes (Informal Proceedings)(BDA). (2005)
12. Ponge, J., Benatallah, B., Casati, F., Toumani, F.: Fine-grained compatibility and replaceability analysis of timed web service protocols. In Parent, C., Schewe, K.D., Storey, V.C., Thalheim, B., eds.: Conceptual Modeling - ER 2007, 26th International Conference on Conceptual Modeling, Auckland, New Zealand, November

5-9, 2007, Proceedings (ER). Volume 4801 of Lecture Notes in Computer Science., Springer (2007) 599–614
13. van der Aalst, W.M.P., Dumas, M., Ouyang, C., Rozinat, A., Verbeek, E.: Conformance checking of service behavior. ACM Trans. Internet Techn. **8**(3) (2008)
14. Benatallah, B., Casati, F., Grigori, D., Nezhad, H.R.M., Toumani, F.: Developing adapters for web services integration. In Pastor, O., e Cunha, J.F., eds.: Advanced Information Systems Engineering, 17th International Conference, CAiSE 2005, Porto, Portugal, June 13-17, 2005, Proceedings (CAiSE). Volume 3520 of Lecture Notes in Computer Science., Springer (2005) 415–429
15. Kongdenfha, W., Saint-Paul, R., Benatallah, B., Casati, F.: An aspect-oriented framework for service adaptation. In Dan, A., Lamersdorf, W., eds.: ICSOC. Volume 4294 of Lecture Notes in Computer Science., Springer (2006) 15–26