A Multi-layer Approach for Customizing Business Services

Yehia Taher[1], Rafiqul Haque[2], Michael Parkin[1], Willem-Jan van den Heuvel[1], Ita Richardson[2], and Eoin Whelan[2]

[1] European Research Institute of Service Science (ERISS), Tilburg University, TIlburg, The Netherlands
{y.taher,m.s.parkin,W.J.A.M.vdnHeuvel}@uvt.nl
[2] Lero – the Irish Software Engineering Research Institute, University of Limerick, Limerick, Ireland
{Rafiqul.Haque,Ita.Richardson}@lero.ie, Eoin.Whelan@ul.ie

Abstract. The reusability of services is a cornerstone of the Service-Oriented Architecture design paradigm as it leads to a reduction in the costs associated with software development, integration and maintenance. However, reusability is difficult to achieve in practice as services are either too generic or over-specified for the tasks they are required to complete. This paper presents our work in defining an approach for achieving service reusability in Service-Based Applications (SBAs) by decomposing the reusability requirements into two layers and then into separate views that allow the customization of business policies, quality of service, tasks and control (i.e., orchestration/choreography) parameters. The objective of defining such an approach is to provide an appropriate solution that will guide the customization of a service's functional and non-functional properties to allow it to be reused in different business contexts.

Keywords: Service, Reusability, Customization, Service Oriented Computing (SOC), Service Based Application (SBA).

1 Introduction

The reusability of services is a cornerstone of Service-Oriented Architectures as it allows the linking together of services to solve an end-to-end business problem or process to create a Service-Based Application (SBA). For instance, services such as *order processing,* and *shipment processing* can be reused to build an *order management* application. Reusability can be deemed as one of the most significant qualities of services within the domain of SBAs for several reasons. In particular, reusability facilitates Just-in-time (JIT) service integration that plays vital role in meeting other important service qualities such as *customer satisfaction*. For example, if a client purchases *goods* from a provider who does not provide an *insurance* service for their delivery and the client asks for shipping insurance the provider should be able to provide this service to promote customer satisfaction, which in turn maximizes the return for provider. In such situations, the provider can integrate a (reusable)

C. Huemer and T. Setzer (Eds.): EC-Web 2011, LNBIP 85, pp. 64–76, 2011.

insurance service with the running business application just in time instead of developing the service from scratch, reducing the up-front costs, for the service provider.

Although reusability has many merits, it has two limitations: generalization and *over-specification. Generalization* facilitates designing services from generic point of view; for example, a generic order management application can be designed to meet most requirements by abstracting away its specificity. This means generic services cannot be used in a specific context since they lack the ability to satisfy the specific requirements of any context. Over-specification is the opposite of over-abstraction. An over-specified service has attributes that are highly-specific in a certain context. Unlike generic services, over-specified services may be reused in a specific context, but the target context has to match exactly the source context in terms of requirements. In practice, this is impractical because the requirements between contexts cannot be symmetric. As an example, payment service developed for a business organization operating in the United States cannot be reused directly by any organization in Europe. This example covers wider area; in fact, it is highly unlikely the service could be reused by any other organization in the US. This implies neither generic nor over-specified (reusable) services can be reused to build SBAs directly.

These considerations give the rise to the concept of *customization* that supports fine-tuning generic as well as over-specified services so they may be reused in different target contexts. This method of service customization has been emerging steadily as a concept as the popularity of service oriented computing technologies increases (e.g., WSDL, BPEL, and so on). Earlier research into service customization, including [1], [18] and [10], focused on configurable EPC approach that is limited to service functions, which is obviously important but not adequate for the modern service-driven business environment. The service-driven business environment of today also involves diverse non-functional requirements including quality-of-service, business policy, and security. Customizing services covering such a wide variety of requirements from disparate domains is a non-trivial task and organizations hire experts to perform these tasks, which increases development costs. This implies that it is paramount to enable users to customize both functional and non-functional aspects of services.

In this paper, we propose a multi-layered approach to building SBAs that supports the customization of component services at different layers. The goal of this research is to ease the complexities of service customization and allow non-IT experts without a background in service related technologies (e.g., business analysts) to customize services. The proposed solution provides guidelines for the non-IT expert to allow them to customize services with respect to the specific context.

We organize this article as follows: section 2 describes the motivating example; the proposed solution is explained in section 3; section 4 explains discusses the works related to this research and finally section 5 concludes the research work and briefly outlines the future extension of this research.

2 Motivating Example

In this section we describe an example order management application, composed of reusable services. Figure 1 demonstrates the BPMN model of the application

containing services that do not consider any specific context or situation. We have chosen BPMN to represent the application because, in our view, BPMN is an ideal option for service customization in the design phase; it provides graphical notations that are easily understood by non IT-experts, including business analysts.

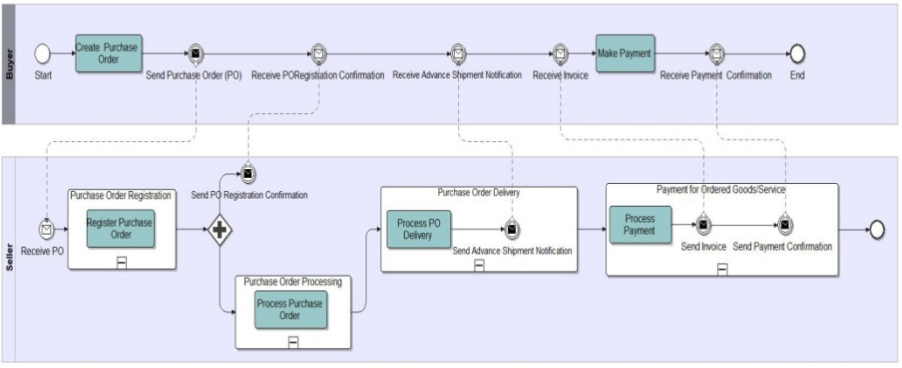

Fig. 1. A purchase order application encapsulates reusable services including order processing, delivery, and payment services

In Figure 1, the two pools represent the partners (*Buyer* and *Seller*) involved in the purchase order process. No specific partner name is given as the process is a generic. The process contains generic activities including *register purchase order, process purchase order*, *process purchase order delivery* and *process payment*. These activities are generic because they are captured from global point of view, i.e., these are activities commonly used by selling organizations. From the perspective of buying organizations, the two generic activities involved in order management are *purchase order creation* and *make payment*. The commonality has also shown in flow of messages between buyer and seller. Besides, the business logic that describes the order of activities has been abstracted (generalized) as well. These generic features of this process facilitate the business organization to reuse it.

As we already mentioned earlier, services designed from global or common perspective cannot be reused directly in a specific context. However, *what exactly are the factors that preclude the direct reusability of generic services*? The simple answer is contextual requirements that vary as the circumstances the service is used in change. For example, the *delivery service* shown in Figure 1 will vary among business types, organizations and the locations it is used in. Another example is the *payment service* which relies on business-specific policies. For example, in business-to-business (B2B) scenarios, buying companies in Europe must issue a *letter of credit* to sellers in South-east Asia before the order is processed, with the letter of credit a legal confirmation from the buyer to credit a certain amount from his account to the seller's account. However, this may not be required for buyers from the United States. The requirements can be more diverse in case of business-to-consumer (B2C) and are enormously important because they are the driving factors that ensure customer

satisfaction and provide a competitive advantage for businesses. As discussed above, context-independent reusable services mostly do not adopt these requirements because they are specific to certain context.

Apart from the non-functional perspective, the functional requirements are also important and vary from context to context. Thus, the functional features of a generic service also may not satisfy the requirement of specific context.

The contribution of this research is largely focused on an appropriate solution which will guide the customization of services for a specific context and present our contribution in the following sections.

3 Multi-layer Approach for Customization

The fundamental principles of the proposed approach are *Personalization* and *Localization* that can be viewed primarily from the perspective of service users (e.g., organizations or individuals). These principles have been used extensively within various domains such as web page development. We adopt them in our solution for two very significant reasons. Firstly, they promote reusability by allowing the customization of services recursively for the specific contexts. As an example, a service provider customizes a payment service, taking the organizational policy into account, and stores a description of the customized service in a service repository. The customization function (when considering customization as a function) can be recalled for specific products if the organization has a large number of products in its pipeline as the payment policy may vary based on product type. This is an example of the personalization of services, which allows the tailoring of reusable services according to the requirements of a context and, subsequently, a business object (e.g., product). Furthermore, localization recalls the customization function for diverse requirements of different geographical locations.

Secondly, both personalization and localization provide a comprehensive understanding of what the requirements should be glued with services and when. This helps to ensure the correctness in specifying or choosing the right requirement parameters at the correct time - i.e., services are personalized and localized at different phases of the customization process Thus, it is important for the users to be aware what customization parameters should be used in which phase.

The solution we propose in this paper supports the personalization and localization of services to simplify the service customization process, which is the primary reason to provide a multi-layered approach for service customization. We present the solution as a reference model for service customization. Figure 2 shows the reference model, which has two-layers: the Service-view Segmentation Layer (SSL) and Service Customization Layer (SCL). We describe both layers in the following sections.

3.1 Service-View Segmentation Layer (SSL)

Based on our study ([4], [15]), a service has various views that are categorized into functional and non-functional viewpoints, as in classical requirements engineering. However, we believe this categorization is not adequate to provide a comprehensive

understanding on services, especially in a modern, service driven-business environment. Thus, in this research, we refine the classical functional and non-functional views into four, finer grained views: the *task view*, *control view*, *quality view* and *policy view*. This granularity will help magnifying the knowledge on services and their requirements for specific contexts. Additionally, they are vital to simplify the customization process and 'fine-tune' the services to their context. The Service-view Segmentation Layer (SSL) comprises of these four views. The SSL is the top layer of the reference model and initializes the customization process through decomposing a service into the different views which are used to devise its requirements. We now explain these views.

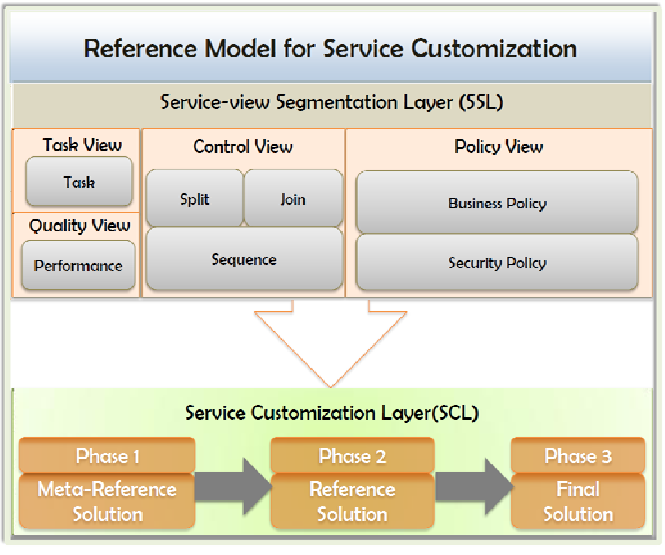

Fig. 2. The Service Customization Reference Model, containing the Service-view Segmentation Layer (SSL) and Service Customization Layer (SCL)

A. Policy View: In a modern-day service-oriented business environment, the business policy is an important requirement of services due to fact that it is the key to gaining a competitive advantage over competitors [17] and differs greatly between organizations. Since business policies are of critical importance for the organization, we separate the policy concern that facilitates explicitness in terms of policy requirements. The policy view helps by analyzing and accumulating the policy requirements of services that can be used during service customization in the Service Customization Layer (SCL). The policy view consists of two types of policies, business policies and security policies. A business policy is a plan, or course of action that is intended to influence and cause decisions and actions, which concern an enterprise in a way that guarantees that the enterprise operates within specified guidelines and regulation [16]. This implies business policies are critical since they influence business decision as well as action.

Furthermore, the advent of the Internet and the Extensible Markup Language (XML) has changed traditional service delivery practices and many services are now delivered electronically. However, the openness of the Internet has made service delivery prone to risks from different types of attacks, such as phishing, through which sensitive business information can be leaked. Thus, security has turned into an important concern for organizations and it is crucial to ensure and ratify the security of payload information during its transfer from sender to receiver. The security policy deals with the security requirements of services and contains technical requirements to ensure secure exchange of business information.

However, policies are also a set of rules or directives intended to influence or guide business behavior [13] and define or constrain some aspect of business [8]. In this research we consider rule as set of constraints that control the service behavior A typical example rule, in this case for the cancellation of an order, is "*a purchase order can be cancelled with money refund if the cancellation request is placed within 15 days from the day order has been placed*". In this example, the number of days is the constraint of the cancellation function. Our solution facilitates specifying such constraints using parameters. Through this research, we target to build a repository of parameters to underpin service customization. We introduce and discuss those parameters in section 3.2.

B. Quality View: non-functional properties described using the general term of Quality of Service (QoS*)* - "a set of non-functional attributes of those contextual entities that are considered relevant to the interaction between the service and its client, including the service and the client, that bear on the service's ability to satisfy stated or implied needs" [3]. QoS is a concept that accentuates the significance of different issues, such as processing time in services and plays pivotal role in aligning the interests (requirements and offerings) of service users and providers. Thus, QoS is important to both participants, and especially from the service client perspective where QoS could be the primary requirement. The satisfaction of the service client depends on the level of QoS provided by the service provider. Thus, service providers today largely concentrate on the quality requirements of the services. In these circumstances we offer a separate view called the *Quality View* that facilitates the analysis of service quality requirements. These requirements are incorporated with services during customization. Essentially, the key quality aspect of a service is *performance*, which involves time-based measurements such as *Response time*, *Processing Time* and so on.

***C. Task View*:** A service is an action that is performed by an entity (the provider) on behalf of another (the requester) [14] and is often a computational entity located at a specific unique location (e.g., URI) that has an internal state and is able to perform one or more functions [7]. The functions are tasks, such as *registering a purchase order*. A service encapsulates and performs these tasks as a blackbox to the outside world. During customization, a user (customizer) must have knowledge about what tasks are encapsulated in a service because a task may not the target context. In this regard, we separate the *task view* of services. The tasks view is the functional aspect of services. This view helps to analyze the required of target contexts. For instance, in the example shown in Figure 1, the register purchase order task may need to be customized to other tasks including check customer credit and check inventory

performed before the registration task. The tasks view is important to ensure the completeness of any functionality in a service. Additionally, it also helps to identify the tasks that are not required to the target context. In summary, this view underpins the capture and specification of the task-related requirements of services.

***D. Control View*:** Services contain tasks that must be performed coherently to produce a desired and effective outcome. Anomalies, such as incorrect task order or deadlock between tasks, jeopardize the service orchestration (the composition of tasks in a composite service) that, in consequence, may produce an incorrect outcome. Simply, tasks need to be controlled in an appropriate manner to obtain a desired outcome. A list of control flow patterns, defined in [2], has been adopted in many successful service technologies, and in particular BPEL. In this research, we create a new view, called the control view, to render the control structure of services. The objective of the control view is to provide users to with an understanding of the process of ordering, splitting and synchronizing of tasks so that the users can define tasks in right order. With this in mind, we include three control connectors: *sequential, split, join* ([2]) that are typically used in service definition. The control view assists in analyzing and defining the connector related requirements for the target service during customization

Form the above description of the reference model for service customization it is clear that the segmentation of views allows a service user to understand the 'nuts and bolts' of services and to analyze its requirements. In addition, the visualization of various aspects of service makes the customization process easier for both IT experts as well as non IT experts.

3.2 Service Customization Layer (SCL)

The service customization process consists of three phases. The customizations of service views are performed during these phases taking the requirements of the target contexts into account. The customization of at these phases produces solutions including meta-reference, reference, and final solution but only the final one is deployable. This implies meta-reference and reference solutions are not concrete solutions. The customization starts at the phase of meta-reference solution which is a generic service can be reused to any context. We assume that organizations import such a reusable service for instance, off-the-shelf one at this phase. The customization of this generic service produces a reference solution which is not final solution. Yet another customization is required to generate the final solution that can be deployed on execution engine. It is worth noting that the customizations of meta-reference and reference solution are treated as service personalization and localization respectively.

Now, *what is the most suitable approach for service customization*? Parameterization plays a pivotal role in customization: the parameterization process allows the setting of parameters for a target solution [9]. We believe parameterization is a relatively simple technique for all types of users because it does not require knowledge on technologies and instead only requires a basic understanding of services. In order to support the parameterization of services, we provide a collection of parameters. As we have mentioned above, a process of creating a repository of parameters is ongoing and we will integrate this repository with the customization tool (also ongoing work). However, we present a sample list of parameters in Table 1.

These parameters are extracted from work in diverse fields such as business, legislation, security and so on. We take the services view into special consideration while selecting parameters because views help to analyze and select the suitable parameters for customization. We cluster these parameters into performance, security, policy, and flow controlling. Mapping these clusters to service views, it can be easily understood by service user which parameters should be used for which view. Noticeably, the list of parameters in the table is influenced by [3], [11], and [19]. They explained these parameters extensively.

Table 1. The customization parameters and operators

Parameter				**Operator**
Performance Parameters	**Security Policy Parameters**	**Business Policy Parameters**	**Flow Controlling Parameters**	
Processing time	Authentication	Availability	MEChoice	Add
Response time	Authorization	Best effort	Before	Prune
Waiting Time	Non-repudiation	Guaranteed	After	Refine
Delay	Intelligibility	Prerequisite	Order	Rename
Throughput	Tamperproof	Co-requisite	Until	Select
		Inclusion	Parallel - Start - Finishes - During - Equal	Aggregate
		Exclusion		
		Segregation of Duty		

In addition, parameterization requires operators which underpin service users to perform customization. In this research, we enlisted a set of operators (see Table 1) that are explained briefly in the followings:

- *Add*: This primitive used to add tasks or functionalities that are required for the target context.
- *Prune*: A task can be removed from a service using this primitive.
- *Refine*: Refine allows a task to be refined into sub-tasks.
- *Aggregate*: Aggregation allows the combination of two or more tasks into a single task.
- *Select*: This operator is used to select tasks or functionalities (that need to be parameterized) and also the parameters. For instance, a user selects task *process payment* of permission process and then selects the performance parameter *processing time*.
- *Rename*: Reaming is used to re-label different parts of services. For instance, a task 'send invoice' of reusable service may be renamed to 'send payment receipt'.
- *Value Tagging*: It is not an operator listed in the table, but we offer value tagging facility for the users. The key idea of value tagging is to facilitate specifying the value of parameters. The framework provides the boolean values of *True* and *False*

as well as a numerical value. Using value tagging, a user can specify the target value for performance parameters for instance, the expected value of the parameter *processing time* equal to *5 days*.

In section 3.3, we briefly explain how to use these operators and parameters with an example.

3.3 Exemplification of Service Customization

In this example we try to show how the proposed solution can be used in practice to customize services. We use the payment service of order management application (see Section 2) for this task. It is a generic service which has been defined without consideration to any specific context (e.g., organization). Now a business analyst in Auto Inc. (a fictitious car-assembling firm) wants to reuse this service and customizes the service using our proposed approach. According to the customization reference model, the analyst places the service at SSL to map the service view and then may perform the following customizations:

A. Task view Customization: The analyst refines the task *process payment* into three different activities including 'create invoice', 'charge credit card', and 'process payment receipt'. Figure 3 shows the refinement. Additionally, the activity 'send invoice' is renamed to 'sent payment receipt'. Two operators *refine* and *rename* are used in this customization.

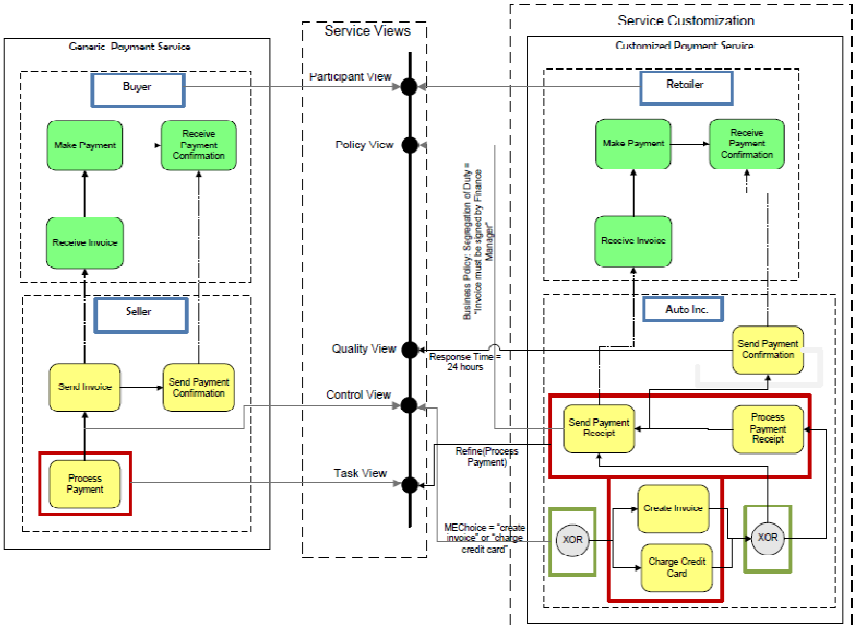

Fig. 3. The figure shows an example of customization of a generic (reusable) service using proposed multi-layer customization solution

B. Control view Customization: The analyst customizes split control the control connectors to connect create invoice and charge credit card activities. Both these activities should not be executed at the same time for the same instances. Thus, the analyst parameterizes the connectors using *MEChoice* parameters (the XOR notation in figure 3) that constrain the execution. *MEChoice* choice means mutually exclusive choice which allows choosing only one of multiple options. This implies, either charge credit card or create invoice should be executed for an instance but not both.

C. Policy view Customization: According to the business policy between the participants' retailer and Auto Inc., the payment receipt must be signed by finance manager. The analyst specifies this policy through parameterizing the 'process payment receipt' task using the business policy parameter *segregation of duty*. This parameter describes how only finance managers are allowed to sign payment receipts (otherwise receipts will not be legally accepted by the retailers).

D. Quality view Customization: In a service-driven business environment there are many quality aspects and we only provide a simple example here. For example, the retailer may require a payment confirmation from Auto Inc. to confirm the company has received the payment and may expect a notification within 24 hours. The analyst parameterizes the task 'send payment confirmation' using *response time* and specifying value 24 hours. In fact, tasks such as 'processing payment receipt' in this example should also be parameterized using quality parameters.

Although the customization in this example looks simple (since we tried to keep it straightforward for the convenience of readers) in practice service customization is enormously complex especially for large scale enterprise applications. This is the very initial phase of our research, and may have several missing points, but from research perspective we believe that this is a highly innovative approach with significant potential because, according to our extensive study, there are tools to customize functional aspects of services but there is no suitable tool for customizing business policies, quality of service and security requirements. Besides, we also believe the combination of view segmentation and parameterization simplifies the customization enormously, which enables any user of the proposed solution to customize services. This is one important contribution that may help to reduce the development costs of services since organizations may not need to hire too many experts from different fields in order to create the service. However, before putting this approach in practice, we plan to provide step-step-step customization guidelines for its users, which will help to reduce costs further.

4 Related Works

In this section we position our solution with related works. This research revolves around two concepts including reusability and customization. Both these concepts are heavily documented throughout various bodies of literature. These concepts are substantial within service engineering domain. To-date, a list of interesting solutions around service customization has been proposed. In particular, [21] proposed a solution that facilitates fragmenting a complex business process into different parts that are intended to be reusable and customizable for target business process model.

The idea of customizing processes through fragmentation is interesting but the solution they proposed is limited to technical aspect and missing technique that facilitates customizing policy related requirements of services.

A collection of reference models (that are used developing business applications) widely known as SAP reference model was produced by [5]. These models are being used in many application services that are developed using technologies from SAP (http://www.sap.com/). This work has been cited heavily, yet criticized by [6]. According to [6], number of SAP reference models is structurally incorrect. Thus, they proposed configurable EPCs within the light of customization concept. Configurable EPCs was investigated by [1], [10], [18], and [12] to identify and model the service variability. They produced interesting results such as *configuration gateways* that support customizing the functional aspects of reusable processes. Noticeably, these works are limited within EPCs and SAP reference model. This means it is not clear whether the proposed solution is applicable to other process model. To solve such a problem, [9] proposed a framework with guidelines to transform a process model to SAP reference model. Now, this framework can map a process model (ignoring the model type) to SAP reference model with customization support. From our perspective, these solutions are too technical for analysts who do not possess solid understanding on different types of technologies (e.g., SAP, ARIS, etc). [22] and [20] proposed relatively simple customization solutions but like many other earlier ones, they ignored the non-functional aspects of services. As we already mentioned the non-functional aspects in particular, security, policy, and quality are critical importance for modern day business environment and thus service engineering.

Now, our multi-layer solution approach is an initiative to simplify service customization through parameterizing both business and technical requirements of services. Some of the earlier solutions also allow parameterizing services but they do not consider business level parameters. Parameterization is relatively simple technique that helps non IT-experts to customize services. Additionally, the segmentation of views helps analyzing the requirements of services especially what customization parameters should be used for the target context.

5 Conclusion

The multi-layer customization solution described in this article aims at supporting non IT experts for customizing services. The proposed solution helps in the customization of services by providing several necessary aspects, including the provision of a service customization reference model (the foundation of the proposed multi-layered solution approach), a comprehensive understanding of services and their customization requirements through service views (the top layer of reference model) and a list of parameters and operators that can be used in the customization of services (the bottom layer of the reference model).

The solution that has been described in this paper is core research in nature that requires extensions and refinement. A simple and user friendly tool implementation is the subject of an ongoing work. We are also developing the tool that will provide step-by-step customization guidelines to the users to ease the customization complexity for

non-IT experts. Additionally, we plan to build a repository of parameters which will be integrated with the tool in future. Therefore, we will continue enriching the repository of parameters.

Acknowledgment. The research leading to these results has received funding from the European Community's Seventh Framework Program [FP7/2007-2013] under grant agreement 215482 (S-CUBE).

References

1. van der Aalst, W.M.P., Dreiling, A., Gottschalk, F., Rosemann, M., Jansen-Vullers, M.H.: Configurable process models as a basis for reference modeling. In: Bussler, C.J., Haller, A. (eds.) BPM 2005. LNCS, vol. 3812, pp. 512–518. Springer, Heidelberg (2006)
2. van der Aalst, W.M.P., Hofstede, H.M.A., Kiepuszewski, B., Barros, P.A.: Workflow Patterns. Distributed and Parallel Databases 14(1), 551 (2003)
3. Benbernou, S., Ivona, B., Nitto, D.E., Carro, M., Kritikos, K., Parkins, M.: A Survey on Service Quality Description. ACM Computing Survey V(N), 1–77 (2010)
4. Curbera, Francisco, Khalaf, R., Mukhi, N., Tai, S., Weerawarana, S.: The Next Step in Web Services. Communication of the ACM 46(10), 29–34 (2003)
5. Curran, A.T., Keller, G., Ladd, A.: SAP R/3 Business Blueprint: Understanding the Business Process Reference Model. Enterprise Resource Planning Series. Prentice Hall PTR, Upper Saddle River (1997)
6. van Dongen, B.F., Jansen-Vullers, M.H., Verbeek, H.M.W., van der Aalst, W.M.P.: Verification of the SAP Reference Models using EPC Reduction, State Space Analysis, and Invariants. Technical Report (2006)
7. Guidi, C., Lucchi, R., Gorrieri, R., Busi, N., Tennenholtz, M.: A calculus for service oriented computing. In: Dan, A., Lamersdorf, W. (eds.) ICSOC 2006. LNCS, vol. 4294, pp. 327–338. Springer, Heidelberg (2006)
8. GUIDE Task Force: Defining Business Rules – What Are They (July 2001)
9. van der Heuvel, W.J., Jeusfeld, M.: Model transformation with Reference Models. In: Proc. 3rd International Conference Interoperability for Enterprise Software and Applications, Funchal, Portugal, pp. 63–75 (March 2007)
10. La Rosa, M., Dumas, M.: Configurable Process Models: How To Adopt Standard Practices In Your How Way? BPTrends Newsletter (November 4, 2008)
11. Lu, R., Sadiq, S., Governatori, G.: On Managing Business Processes Variants. Data & Knowledge Engineering Journal 68, 642–664 (2009)
12. Mendling, J., Moser, M., Neumann, G., Verbeek, H.M.W., van Dongen, B.F., van der Aalst, W.M.P.: Faulty ePCs in the SAP reference model. In: Dustdar, S., Fiadeiro, J.L., Sheth, A.P. (eds.) BPM 2006. LNCS, vol. 4102, pp. 451–457. Springer, Heidelberg (2006)
13. OMG's RFP: Organizing Business Plans: The Standard Model for Business Rule Motivation (2002)
14. O'Sullivan, J., Edmund, D., Hofstede, H.M.t.A.: Service Description: A survey of the general nature of services. Technical report (2002)
15. Papazoglou, P.M.: Service Oriented Computing: Concepts, Characteristics, and Directions. In: Proceeding of the Fourth International Conference on Web Information System Engineering, WISE 2003 (2003)
16. Papazoglou, P.M., van der Heuvel, W.J., Leymann, F.: Business Process Management: A Survey. ACM Computing Survey, 1–48 (September 2009)

17. Simchi-Levi, D., Kaminsky, P., Simchi-Levi, E.: Designing and Managing Supply Chain: Concepts Strategies and Case Studies, 3rd edn. McGraw-Hill Irwin, Boston (2008)
18. Stollberg, M., Muth, M.: Efficient Business Service Consumption by Customization with Variability Modeling. Journal of System Integration (2010)
19. UN/CEFACT Modeling Methodology (UMM): UMM Meta Model – Foundation Module Candidate for 2.0 (2009), http://umm-dev.org/ummspecification/
20. Wang, J., Yu, J.: A business-level service model supporting end-user customization. In: Di Nitto, E., Ripeanu, M. (eds.) ICSOC 2007. LNCS, vol. 4907, pp. 295–303. Springer, Heidelberg (2009)
21. Zhilei, M., Leymann, F.: A. Lifecycle Model for Using Process Fragment in Business Process Modelling. In: BPDMS (2008)
22. Zhu, X., Zheng, X.: A Template based Approach for mass Customization of Service Oriented E-business Applications. In: Kishino, F., Kitamura, Y., Kato, H., Nagata, N. (eds.) ICEC 2005. LNCS, vol. 3711, Springer, Heidelberg (2005)