Business Driven SOA Customization

Pietro Mazzoleni and Biplav Srivastava

IBM T. J. Watson Research Center
Hawthorne, USA 10532
{pietro@us,sbiplav@in}.ibm.com

Abstract. Service Oriented Architecture, e.g., Web services, as building blocks for IT based on open standards, assist enterprises become more responsive to the changing business environment when they are implemented and used in the context of business processes. In this direction, packaged integration platforms like IBM's Composite Business Services or SAP have pre-configured business processes offered as web services. When the demand for a new capability arises, it can be addressed by building new services or by customizing an existing service. Service providers try to cover as much of the potential customer requirements as possible with provided capabilities but a complete coverage is not possible as individual industries might have unique requirements and customers can integrate services from multiple parties. In this situation, the problem is not whether a particular customization method will work but rather how to determine the overall impact of a new requirement in a complex SOA environment in terms of activities to be done and at what cost.

In this paper, we propose a solution to these problems by introducing the notion of business driven customization of SOA (specifically web services). We introduce a formal model capturing properties and relationships of business objects and business processes, and their implementing services and messages. We also have instance-independent, impact propagation rules to encode the desirable customization behavior of any implementation. Now, we can capture new requirements as change triggers in the model and using the modeled rules, can precisely compute the scope of their overall impact spanning both business and IT domains. Overall, we introduce the customization and impact model, describe its implementation, and illustrate its application in an industry scenario with large number of services with complex characteristics (SAP).

1 Introduction

Service Oriented Architecture (SOA) e.g., Web services, is the popular building blocks for open-standards based IT today. Here, service providers publish specification of their IT capabilities wrapped as services onto registries. The services can be discovered by potential clients later and then invoked on the providers, all using standardized interfaces. They can assist businesses become more responsive to the changing business environment when they are implemented and used in the context of business processes.

Packaged integration platforms like IBM's Composite Business Services[1] or SAP have pre-configured business processes expose as services. As an example, to simplify the process of adopting SOA, SAP is splitting up its application functionalities

A. Bouguettaya, I. Krueger, and T. Margaria (Eds.): ICSOC 2008, LNCS 5364, pp. 286–301, 2008.

(e.g. ERP, CRM as well as industry specific solutions) into thousands of ready-to-consume services. These services are grouped into sets (or bundles, in SAP terminology) along business scenarios (e.g. complaint management or order to cash) and they have a built-in semantics which partitions data into pre-defined changeable business processes and business objects. Similar semantic is adopted by custom-built services as well as partner-built services which may be created for functionality gaps that are not covered directly by SAP. Other vendors admit industry-standard business processes like RosettaNet, and web services are used to implement these processes. In business, there is an increasing need for service customization as many industries as well as software vendors are moving their architectures towards services.

Given that changes will continuously happen in any business that can affect their business objects and business processes, we are interested in precise methods that can characterize the impact of these changes on their services (specifically, web services) implementation. The business changes rarely impact a single service. Today, the business to IT alignment is implicit and it is impossible to precisely determine the major changes from the minor ones.

Knowing the changes to be made in a complex SOA brings several advantages. On the business side, it provides an understanding on the parts of the businesses which could be affected by the new requirement leading to valuable insights for project plan, cost of implementation, and best practices in building customizable Services. On the technical side, it supports software engineers in identifying which tasks should be implemented in the system. We also aim to provide methods that will help an organization determine guidelines about when to customize an existing service versus create new ones with richer templates so that they have a rich collection of distinguishing services that can be highly reusable. The issues we consider are complementary to how customization may be implemented, and hence, our approach will work with any method for the latter in literature[2,3,4,5]. In [6], the authors propose a method to customize Web Services published by a provider by using WS-Policy based customization points that a consumer can select. The service with the selected customizations is now created and hosted by the provider. The consumer can call the customized service and do its processing. Closely related to the notion of customization are the concepts of *personalization* and *adaptation* of web services. As discussed in detail later in Section 6, personalization deals with how to modify the content output from a web service specific to the user at run time whereas adaptation deals with how to modify the behavior of a deployed web service at runtime based on environmental considerations. A new service is neither created nor deployed for the unique service request. Our contributions are that we:

1. Introduce a business driven model for computing the scope of customization of SOA (specifically web services)
2. Present a prototype implementation, investigate the source of complexity in the model and discuss how it can be used to expose the trade-offs /issues in different types of customization.
3. Show its generality and usefulness by applying the model to a complex service scenario, i.e., SAP.

The paper is organized as follows: we start with preliminaries on service customization, discuss a motivational scenario to bring out the issues in customization and then present our model for customization. Next we discuss its implementation and apply to the complex services scenario of SAP. We round up the paper by discussing the salient features of our work, the limitations and related work.

2 Background and Motivating Scenario

2.1 Background

When a requirement for a service arises, it can be addressed by building a new service or customizing an existing one. By *customization* of web services, we mean the process by which the behavior of existing web services can be modified to meet the requester's requirements. Specifically, we consider customization as:

- Building parameterized service interfaces so that they can capture a wide range of situations (template);
- Building extensibility mechanisms in the middleware (e.g., proxy) to integrate and extend available services; to meet new requirements unanticipated by the service provider.
- Creating a new customized service instance that is deployed specifically for the unique service requester (and may later cater to others as needed). Customization is expected to be an offline activity unless the middleware supports the new customized service to be deployed on-the-fly.

While any service can be customized in theory to meet any requirement, doing it indiscriminately runs the risk of ending up in the registry with a proliferation of service versions that are not much distinct from each other. In practice, customization should be used to build distinguishing services that provides a robust, reusable service portfolio across changing requirements. Hence, when to build a new service and when to customize from an existing service should be used as a complementary strategy.

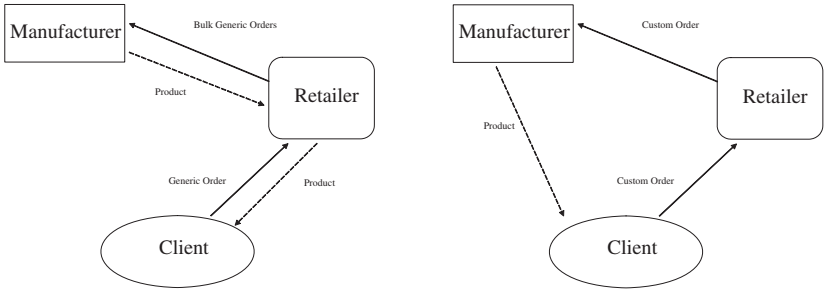
Service providers try to cover as much of the differences as they can anticipate using some of the following techniques:

- *Parametrization*. In here, the actual parameters of the service can be modified based on the data in the arguments [7];
- *Function overloading*. In here, multiple instances of the same operation are defined with different set of arguments [8];
- *Templatization*. In here, a service template is available to capture the typical service of interest. Users are guided to select parameters to instantiate the template and create the service matching their requirements [9].

However, a provider cannot anticipate all types of requirements from potential consumers. The types of differences that can appear between the requirements of a requester and the available services from provider can be along business objects, processes, services and messages.

2.2 Motivating Scenario

We present a common scenario to motivate the problem. Consider a simplified business process for product supply-chain, called *General Delivery* where a manufacturer *M* sells products to its clients *C* via retailers *R* (see Figure 1(left)). *C* could place an order for the product with *R* and get its delivery. *R* on its part can periodically place bulk orders with *M* and take delivery to replenish its inventory.



(a) General Delivery (b) Custom Delivery

Fig. 1. Two differing processes in the example scenario

Suppose that often, some clients want to place custom orders which are different from general products. A typical example is for an apparel manufacturer to allow customers to provide their own logo on the shirts they order. The manufacturer now wants to change the supply-chain slightly to allow these clients to be able to place their custom orders and get the products delivered. Custom orders can still be delivered from *M* to *C* through *R*. However, because custom orders take time to build, *M* may want to ship the product directly to *C* in new business process called *Custom Delivery*. Such a process for custom orders is shown in Figure 1 (right).

The changes in the scenario come from:

1. Changes in the business objects: Custom orders are now introduced.
2. Changes in the business process: The product now can be shipped to both *R* and *C* via *General Delivery* and *Custom Delivery* processes. Orders from *R* can now be both periodic for general products and unexpected for custom products.
3. The changes in the business object and processes could lead to changes in the interfaces of IT services used to implement them (e.g., OrderPlacementService, ProductTrackingService) and the messages involved in their communication.

The changes 1 and 2 are business changes necessitated by world events. We want to take them as inputs and automatically determine business and IT changes as shown in 3 for the specific Service-Oriented Architecture (SOA) implementation (at the manufacturer in the example).

3 Business-Driven Service Customization Model

In this section, we introduce the model capturing properties and relationships of a business-process driven SOA implementation. The model consists of facts and rules. Facts represent claims about the problem universe and they may be true or false. Rules are used to encode relationships among facts and to infer new facts from the ones in the model. Using the same model, we propose a set of rules to compute the overall impact of a new business requirement spanning both business and IT domains.

We will use logic programming to define the model. Specifically, we used Smodels [10], an implementation of stable model semantics [11] and well-founded semantics [12] for normal logic programs. An answer to a problem is a set of facts, called stable model, which tell which facts are true.

3.1 A Simplified Model for Business-Process Driven SOA Implementation

We envisage that one can define an industry in terms of a collection of scenarios describing what the enterprise does and how. There can be cross-industry scenarios which describe the common activities any enterprise has to perform regardless of its area of business (e.g., *Annual tax filing*). Then there are activities specific to particular industry like *prepare clinical trial* for *Healthcare* or *emission management* for *Mining*.

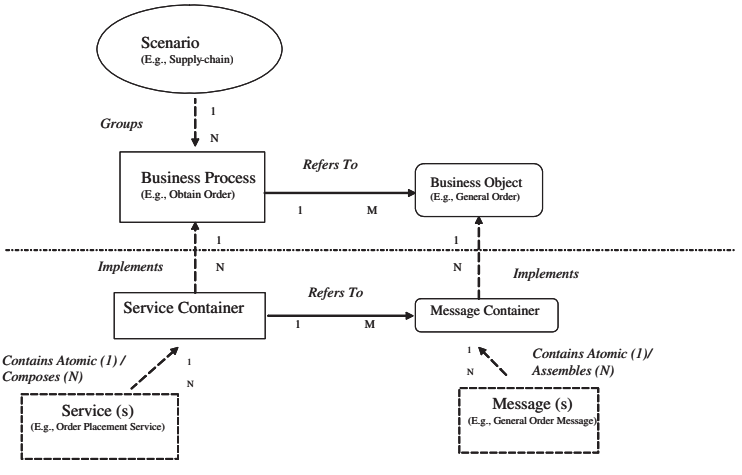


Fig. 2. A simplified model for business-process driven SOA implementation

In Figure 2, we present our simplified model for business-process driven SOA implementation. Note that we explicitly separate business from IT elements and we use directional arrows to indicate dependencies between elements.

At business level, an enterprise domain is decomposed into multiple scenarios. Each scenario is realized by one or multiple business processes (*BP*s) and by one or multiple business objects (*BO*s). Each BP is defined with a business logic and references to BOs.

```
% Business Objects Structural Rules
1:  busObj(product)
2:  busObj(customerAddress)
3:  busObj(retailerAddress)
4:  busObj(order)
5:  boDependsOnBo(order,product)
6:  boDependsOnBo(order,clientAddress)
7:  boDependsOnBo(order,retailerAddress)
% Business Processes Structural Rules
8:  busProc(shipOrder_bp)
9:  busProc(receiveOrder_bp)
10: busLogic(shipOrderLogic)
11: busLogic(receiveOrderLogic)
12: bpHasLo(shipOrder_bp,shipOrderLogic)
13: bpHasLo(receiveOrder_bp,receiveOrderLogic)
14: bpRefersToBo(receiveOrder_bp,order)
15: bpRefersToBo(shipOrder_bp,product)
16: bpRefersToBo(shipOrder_bp,retailerAddress)
```

Fig. 3. Business level model elements

Each BO can be referred by multiple processes as well as by other BOs. In our model, we denote business scenarios, BPs, business logic, and BOs as *facts* whereas we use *structural rules* to define rules capturing relations between facts.

Example 1. *Consider the General Delivery scenario presented in Figure 1. For the sake of illustration, we can assume the domain of interest to be represented by four BOs (*`product`*,* `customerAddress`*,* `retailerAddress`*, and* `order`*) and two BPs called* `receiveOrder_bp` *and* `shipOrder_bp`*. The former BP collects new orders whereas the latter loads existing orders and prepares new shipments. The business logics of the two BPs are defined in* `shipOrderLogic` *and* `receiveOrderLogic` *respectively. Figure 3 shows facts and structural rules composing our simple model. As example, structural rule No. 11 tells that BP* `receiveOrder_bp` *refers to (i.e. uses as part of its logic) BO* `order`*.*

At IT level, each BP is realized by a Service Container (*SC*) referring to one or multiple (Web) service(s). In case a SC refers to multiple services (due to service composition), it also connects to a logic describing the process flow among those services. In addition, each SC refers to one or multiple Message Containers (*MC*) describing atomic or aggregate messages that constitute its inputs and outputs. Messages are IT realizations of BOs in specific formats of interest (e.g., order information in XML format). Once again, we use *facts* to generally indicate the base element of the model (i.e. SCs, services, service logics, MCs, and messages) and *structural rules* represent existing relations among facts.

Example 2. *Figure 4 shows facts and structural rules modeling the IT aspects of our simple SOA example.*

```
% Service Container Structural Rules
1 : srvContainer(mngReceivedOrder_ws)
2 : srvContainer(processOrder_ws)
3 : scImplementsBp(mngReceivedOrder_ws, receiveOrder_bp)
4 : scImplementsBp(processOrder_ws, shipOrder_bp)
5 : scDependsOnSc(processOrder_ws,mngReceivedOrder_ws)
6 : scHasLogic(processOrder_ws,processOrderlogic)
% Message Container Structural Rules
7 : messCont(storeOrder_ms)
8 : messCont(shipOrder_ms)
9 : mcImplementBO(storeOrder_ms,order)
10: mcImplementBO(shipOrder_ms,product)
11: mcImplementBO(shipOrder_ms,retailAddress)
12: scRefersToMc(mngReceivedOrder_ws,storeOrder_ms)
13: scRefersToMc(processOrder_ws,shipOrder_ms)
```

Fig. 4. IT level model elements

We assume two SCs exist: `mngReceivedOrder_ws` *and* `processOrder_ws`. *The first SC implements* `receiveOrder_bp` *BP while the latter implements* `shipOrder_bp` *BP. In the Figure, structural rule No. 5 states that* `processOrder_ws` *depends on* `MngReceivedOrder_ws` *which means that the first SC is a composed service using, among others, some of the messages referred by the second SC. The logic of the SC is captured in* `processOrderlogic`.

On the message side, two MCs exist, one for each SC. `storeOrder_ms` *represents the operation of accepting new orders (defined using BO* `order`*) whereas* `shipOrder_ms` *is in charge of creating a new shipment.*

Our notion of SC and MS are consistent with web services standards. SC can be represented by a WSDL and stands for an atomic service or a composite service. The latter logic can be described by abstract BPEL. The MC represent messages in WSDL both simple and complex message types.

In our model, we intentionally kept the definitions of services and messages simple. This is because we are interested in the relations between business and IT elements of a SOA and not to extensively represent all details of a SOA implementation. Existing standards, such as OWL-S[13] or broadly used ontologies like SAP Global Data Type (GDT)[8], can be used for more refined models.

3.2 Formalizing Business Process-Driven Impact on Services

In this subsection, we extend the model to include *impact rules* to encode the customization behaviors of a SOA implementation. When triggered by a new customization requirement (e.g. create a new BO for custom order), those rules can precisely scope the overall impact of the requirement to the SOA, spanning both business and IT domains.

In our model, we view customization changes as BO and BP driven impacts on implementing services and messages. When a business element of the model changes, we

use impact rules to propagate such change across the model to identify which (business and IT) elements will be affected.

Impact rules are defined using logic programming and reflect the dependencies depicted in Figure 2. An inference engine is used to compute the overall impact of a new business requirement. The inference engine takes three inputs: a) facts and structural rules, defined for the specific SOA implementation, b) impact rules, defined independently from any SOA implementation and c) the new business requirements expressed as additional facts for the model (e.g. `changeBusObj(order)`).

Figure 5 presents the list of impact propagation rules we defined using Smodel. For example, rule No. 2 (`changeBusObj(Y):-boDependsOnBo(X,Y), changeBsObj(Y)` would tell that if BO `X` depends on BO `Y` and BO `Y` changes, then BO `Y` will change. In the example, `X` and `Y` are variables whose values is not defined by the user but rather inferred by the engine starting from facts either in the model or inferred from evaluating other rules.

Instead of describing each rule in detail, in what follows we use an example to describe how the rules compute the overall impact of a new customization requirement.

Example 3. *Consider once again the scenario introduced in Section 2.2. The business requirement can be modeled with two facts:* `changeBusObj(order)`*, to represent the need to distinguish between custom and general orders, and* `changeBusLogic(ShipOrderLogic)` *to represent the change in the business logic for handling custom orders. Note there is not a predefined order according to which rules should be evaluated. Instead, the inference engine analyzes all possible combinations of fact searching for stable models. On the business side,* `changeBusObj(order)` *propagates, through impact-rules No. 2 and No. 3, to BP* `busProc(receiveOrder_bp)` *and business logic* `processOrderlogic`*. As result, the engine will introduce two new facts:* `changeBusObj(receiveOrder_bp)` *and* `changeBusLogic(processOrderlogic)`.

On the IT side, rule No.10 propagates the impact of the change to `ProcessOrderLogic` *Service logic whereas rule No. 11 extends it to MC* `storeOrder_ms` *. Finally rules No. 14 takes in input* `changeMessCont(storeOrder_ms)` *and propagates the impact of the requirement to* `mngReceivedOrder_ws`.

As result of the inferencing process, the inference engine highlights two different actions for the two SCs in the model: a change to the service logic of SC `processOrder_ws` *and a change for message (*`storeOrder_ms`*) in case of SC* `mngReceivedOrder_ws`*. Such advises are specific to the facts and structural rules defined in the previous subsection and would have been different in case of a different SOA implementation.*

The impact propagation rules presented in Figure 5 represent the "sufficient but not necessary" set of changes to be considered as result of a new business customization requirement. It is so because all possible changes in the SOA are identified. However, even though the scope of changes known, a system architect might decide not to take action as result of the new business requirement based on assessments that are not captured in the model. As an example, a system architect might decide not to change a

```
% Business Objects Rules
1:  % If a BO X changes than all BOs depending from X should change
    changeBusObj(Y):-boDependsOnBo(Y,X), changeBusObj(X)
% Business Processes Rules
2:  % If a BO X changes, all BPs referred by X might change
    changeBusProcess(Y):-boRefersToBp(Y,X), changeBusObj(X)
3:  % If a BP X changes, X's business logic should change
    changeBusLogic(Y):-changeBusProc(X), bpHasLo(X,Y)
4:  % If a Business Logic X changes, the BO using X should change
    changeBusProc(Y) :-changeBusLogic(X), bpHasLo(Y,X)
5:  % If a BP X changes, all BPs depending from X should change
    changeBusProc(Y):-bpDependsOnBp(Y,X), changeBusProc(X)
% Service Container Rules
6:  % If a Service X changes, all SCs depending from X should change
    changeServCont(Y):-scDependsOnSe(Y,X), changeService(X)
7:  % If SC X changes, X's service logic should change
    changeServLogic(Y):-scHasSl(X,Y), changeServCont(X)
8:  % If Service Logic Y changes, X's service logic should change
    changeServCont(Y) :-scHasSl(Y,X), changeServLogic(X)
9:  % If SC X changes, all SCs referred by X should change
    changeServCont(Y):-scDependsOnSc(Y,X), changeServCont(X)
10: % If the logic of BP X changes, the Logic of SC implementing X should change
    changeServLogic(K):-changeBusLogic(Y),
% Message Container Rules
11: % If a BO X changes, all Services implementing X should change
    changeMessCont(X):-changeBusObj(Y), mcImplementsBo(X,Y)
12: % If a MC X changes, all MCs depending from X should change
    changeMessCont(Y):-mcDependsOnMc(Y,X), changeMessCont(X)
13: % If a Message X changes, all MCs depending from X should change
    changeMessCont(X):-mcDependsOnMe(X,Y), changeMessage(Y)
14: % If a MC X changes, all SC referred by X should change
    changeServCont(Y):-mcRefersTosc(Y,X), changeMessCont(X)
```

Fig. 5. Impact Propagation Rules

service message until it is used as part of a business process. Again, even if a BO *A* becomes irrelevant for the business, a system architect might choose not to change service messages implementing *A* but simply ignore the content when appears in the services. To better inform the system architect on the changes to be made on the SOA implementation, one can refine the model introducing the concept of "type of change". In an extended version of our model, business requirements will be defined not only with the affected element (e.g. BO - product) but also with the type of change to implement (e.g. add new BO). Similarly, impact rules will be extended to consider the type of change when propagating a new requirement across the model.

4 Implementation Considerations

We now discuss how our approach for business-process driven service customization can be implemented. Recall that the solution consists of three key components - (a) *Facts* - a set of claim about BOs, BPs, and their implementing services and messages of interest, (b) *Structural rules* - rules capturing properties and relationships among facts, and (c) *Impact rules* - universal rules encoding the desirable propagation of customization behavior in the model. A logic checker can work with the facts and the rules to make decisions on what needs to be customized.

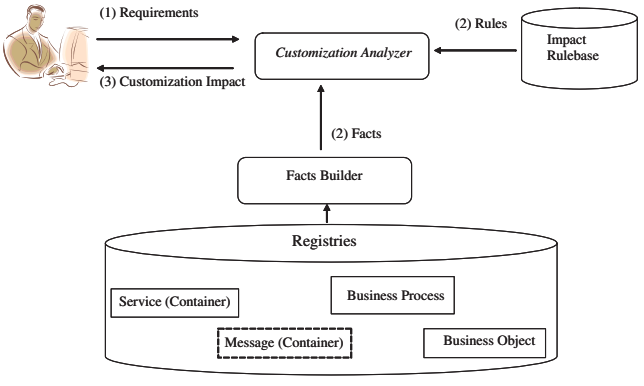


Fig. 6. A conceptual architecture for realizing *Customization Analyzer*

We envisage this approach to be implemented as a decision support aid as shown in Figure 6. The user wants to enquire about the impact of some business events (requirements). The *Customization Analyzer* translates the requirements to facts. The *Fact Builder* computes facts and structural rules of the SOA instance based on existing registries for known BOs, processes, services and messages[1]. The *Impact Rulebase* contains the impact rules. The *Customization Analyzer* uses the facts and the rules collectively from the requirements, the *Fact Builder* and the *Impact Rulebase* to determine what services could be customized to meet the requirement.

We chose a logic formalism to implement our approach. An alternative would have been to express the relationships using a graph theory formalism like that supported in UML. Unfortunately, such formalisms need extensions to represent constraints and were not our first preference.

Estimating the size of the model: With any model-based approach, it is always insightful to know what contributes to the size of the model and use that knowledge for solving problems of interest effectively. Let us consider the size of the model (facts and rules) given the numbers of BOs BPs, and their implementing services and messages as

[1] Current Web Services standards do not require all messages to be explicitly registered but rather that they are addressable and accessible through Uniform Resource Identifiers. Hence, they could be considered optional for the model.

Model Element	*# Facts*	*# Structural Rules*	*Description*
BO	N_{BO}	$(N_{BO} * (N_{BO} - 1))/2$	BO can depend on all BOs except itself and those that depend on it.
BP	2*N_{BP}	$((N_{BP} * (N_{BP} - 1))/2)$ + N_{BP}	No. of facts includes claims for BP and its business logic. No. of rules analogous to BOs plus the relation between BPs and their logic.
BP-BO	-	N_{BP}*$(N_{BO})^k$	Assume k is an upper limit on the number of BOs referred by a BP.
SC	(2*N_{BP})	(N_{BP})	Assume each BP is implemented by at most one SC. No. of facts includes claims for SC and its composition logic.
SC-BP	-	N_{BP}	Assume each BP is implemented by at most one SC.
MC	$(2 * N_{BO})$	(N_{BO})	Assume each BO is implemented by at most one MC. No. of facts includes claims for MC and its aggregation logic.
MC-BO	-	N_{BO}	Assume each BO is implemented by at most one MC.
SC-MC	-	N_S*$(N_M)^k$	Assume k is an upper limit on the number of MCs referred by a SC.
Message	N_M	-	Each message is claimed.
Service	N_S	-	Each service is claimed.
Service-SC	-	$(N_S)^k$	Assume k is an upper limit on the number of services referred by a SC's logic.
Message-MC	-	$(N_M)^k$	Assume k is an upper limit on the number of messages referred by a MC's logic.

Fig. 7. Estimated maximum ($\lceil\ \rceil$) number of facts and structural rules

N_{BO}, N_{BP}, N_S and N_M respectively. Table 7 presents a table estimating the *maximum possible* number of rules in the model with description.

The number of impact rules is a constant, I, and is independent of the number of entries in the registries. Adding up all the facts and rules up, size of the model is O($N_{BP} * (N_{BO})^k + N_S * (N_M)^k + (N_S)^k + (N_M)^k$), where k represents an appropriate constant. This implies that the size is dominated by the number of BOs and how many get referenced by BPs, the number of messages and how many get referenced by services, and the number of services.

In itself, the logic programming system we use, Smodels [10] with stable model semantics, is quite efficient and can handle models with up to a million facts and rules effortlessly[14]. The models in our examples are handled in less than a second. Note that since industry information about BPs and BOs are organized along scenarios, and a user is usually interested in a few scenarios at a time based on their expertise, it should be generally possible to scope the model for most problems of interest.

5 Industry Case Study: Customization with SAP Services

In the introduction, we discussed how SAP is moving its architecture towards services. Grouped into sets (called bundles), these services have built-in semantics which partition data into an extensible set of BPs and BOs. In this section, we adopt SAP to

illustrate the application of our approach in an industry scenario with large number of services with complex characteristics.

In our study, we implemented the ordering scenario in Section 2 using the SAP services publicly accessible from SAP Enterprise Workplace [2]. Specifically, we focused on "Sales Order Processing"[3], the set of capabilities which "allows sales representatives to easily configure, price, and create sales orders for customers" .

To understand the level of complexity in creating an SOA solution using services like the one offered by SAP, consider that SAP implements a large number of services covering different business processes and industry scenarios. Not only, the same process might be offered in different variants (e.g. SAP Workplace lists 11 variants for "Sales Order Processing"). Enterprise services from SAP might be very complex, exposing interfaces (WSDL) with several thousands attributes in input and output. In fact, the approach used by SAP in creating services is to include all possible variations directly as optional elements of the service interface. In our example, if we focus on "the operations that sales employees can use to read or process data about a customer" (`manage customer_in` in the Workplace), we can find 15 different operations (each implemented as service) for the BO called `customer`. `Customer` is a very complex object referring to multiple data elements including, among others, company name and address, communication data (phone, email, etc), contact person, bank details, industry sectors, and marketing attributes. Not all 15 services uses all `customer`'s attributes. In fact, as presented in Figure 8, there can be multiple versions of the same service to perform slightly different functionalities on different subsets of the BO data element.

ServiceName	*Input*	*Output*
`CustomerBasicDataByID QueryResponse_In`	CustomerID	Read Customer Address and CommunicationData
`CustomerERPBasicDataByID QueryResponse_In_V1`	CustomerID	Read Customer Address and CommunicationData
`CustomerERPBasicDataByID QueryResponse_In_V2`	Range of CustomerIDs	Customer Address and CommunicationData

Fig. 8. SAP offers multiple version of the same service

The reason for having multiple versions of the same service is because customizing SAP-based services to meet specific user requirements is a complex and tedious task. The user has to (a) identify the service(s) to be enhanced, (b) extend the corresponding BO or BP, and (c) choose among different versions of the same service to find the one which fits better with the new requirement and (d) write the corresponding code to model the new behavior [15]. More importantly, if the enhanced BO is used in multiple services, developer has to manually identify and then implement the code for all affected services to ensure consistent behavior [16].

[2] https://www.sdn.sap.com/irj/sdn/esworkplace. Last access June 2008

[3] http://erp.esworkplace.sap.com/socoview(bD1lbiZjPTgwMCZkPW1pbg==)/render.asp?id=B8BE8D31D91E4B9EBAAACD83FF85A614&fragID=&packageid=DBBB6D8AA3B382F191E0000F20F64781&iv=

1 : boDependsOnBo(customer,goodsRecipientParty)
2 : boDependsOnBo(customer,billToParty)
3 : boDependsOnBo(customer,salesTerms)
4 : boDependsOnBo(customer,bankAccount)
5 : boDependsOnBo(customer,items)
6 : mcImplementsBo(customerERPBasicDataByIDQuery_sync_V1,customerId)
7 : mcImplementsBo(customerERPBasicDataByIDResponse_sync_V1,customer)
8 : mcIsReferredbysc(customerERPBasicDataByIDQueryResponse_In_V1, customerERPBasicDataByIDQuery_sync_V1)
9 : mcIsReferredbysc(customerERPBasicDataByIDQueryResponse_In_V1, customerERPBasicDataByIDResponseMessage_sync_V1)
10: mcIsReferredbysc(StandardMessageFault, customerERPBasicDataByIDQueryResponse_In_V1)
11: mcImplementsBo(customerERPBasicDataByIDQuery_sync_V2,customerId)
12: mcImplementsBo(customerERPBasicDataByIDResponse_sync_V2,customer)
13: mcIsReferredbysc(customerERPBasicDataByIDQuery_sync_V2, customerERPBasicDataByIDQuery_sync_V2)
14: mcIsReferredbysc(customerERPBasicDataByIDResponse_sync_V2, customerERPBasicDataByIDResponseMessage_sync_V2)
15: mcIsReferredbysc(standardMessageFault, customerERPBasicDataByIDQueryResponse_In_V2)
16: boReferredbyBp(customer,manageCustomer)
17: scImplementsBp(customerERPBasicDataByIDQuery_sync_V1,manageCustomer)
18: boReferredbyBp(customer,createOrder)
19: boReferredbyBp(order,createOrder
20: scImplementsBp(customerERPBasicDataByIDQuery_sync_V1,manageCustomer)
21: scImplementsBp(createNewOrderService,createOrder)

Fig. 9. SAP Scenario - Facts and Structural Rules

Figure 9 shows a very small fragment of structural rules which have been built using our system for the actual SAP services. In the Figure, a BP `createOrder` creates new orders for the customer. `CreateOrder` composes service `customerERPBasicDataByIDQuery_sync_V1`, from SAP, and service `createNewOrderService`, from the specific back-end system used for billing.

In the example, facts and structural rules modeling the IT elements of the SOA solution have been automatically generated by parsing the interfaces of the services available in the service registry (e.g. IBM Web Service Registry and Repository - WSRR). On the other hand, we look at process model documentation (such as the one generated by IBM Websphere Business Modeler-WBM) to create facts and structural rules related to the business.

In what follows, we describe, using an example, how our model can help an organization to identify all services and processes affected by a business change.

Example 4. *New Requirement:*Remove bank account from BO Customer as transactions are always paid cash or via credit card.
Question: Which are the services to be changed as result of the requirement?

This question can be easily answer by our model. First, the system architect defines the requirement as new fact for the model. In the example, the fact `changeBusObj(BankAccount)` *is defined. Second, the* Customization Analyzer *computes the stable model given in input facts, structural and impact rules. Impact Rule No. 1 propagates* `change(BankAccount)` *to BO* `Customer` *and from there, rule No. 2, propagates it to BP* `CreateOrder`*. The same process is continued for the IT elements of the model.*

At anytime, the Customization Analyzer *can trace the impact rules from which a given fact is inferred. As example, it can distinguish (a) the SCs which should change because a referred MC changes (impact rule No. 14) from (b) the SCs which should change because the implemented BP changes (impact rule No. 8). Similarly, it can identify which SCs are implementing a BP (structural rule No. 17 in Figure 9). A system architect can now identify which services should be changed even thought they don't implement any BP (*`customerERPBasicDataByIDQuery_sync_V2` *the our scenario) and decide not to change them as result of the new business requirement. Such information can also be used to identify which versions of the same service exist and which one need to be changed to address the new requirement. This allows keeping fewer versions of the same service and to remove them when not longer in the business.*

6 Discussion and Related Work

In this section, we look at how our work can be extended and the related work. Until now, we have described how our approach can be used to determine the extent of change in a SOA implementation as necessitated by business requirements. Some representative approaches for customization from different disciplines are: AI [2], semantic web [3], graph theory [4] and Petri Nets [5].

An organization is likely to integrate services from multiple parties, each with a different approach to customization. In this context, it is very difficult to identify which are the actual tasks to be performed on the system and to estimate the overall cost of implementing the business requirement. Our work can be used to answer many other IT and business questions including: (a) if some existing services will be customized, determine what are the choices and associated costs and (b) determine whether new IT services should be created or existing ones be customized. The two questions are closely related as there exists multiple approaches to customize an element of the model and each alternative might be associated with a different cost. As example, the cost of changing the logic of a SC is different depending on if such a logic is defined externally to the composed service using a workflow specification language (such as BPEL) or tightly coupled (hard-coded) on it. Our approach can address this problem as the model can be extended to consider costs and customization requirements as Facts for the IT elements composing specific SOA implementation. The *Customization Analyzer* will now be able to compute the overall cost of the new business requirement by simply considering the cost of all affected elements. This will be a very valuable information for the business in order to estimate the cost of the operation. The system architects can use such information to properly choose the customization approach for the services they integrate in a SOA solution depending on the customization requirement it might

be subjected to during its lifecycle. At the time, we recognize the need to integrate such functionality and we plan to include cost and Customization approaches in the next release. Some work we recently done in this space could be applied here[17].

Closely related to the notion of customization are the concepts of *personalization* and *adaptation* of web services. In[18], the authors present a personalization approach where a general information service can provide different types of information specific to individual users using RSS. The generic schema is published and by selecting specific sections, the system can personalize the content. In this approach, the service is made specific to the user at run time and an instance of it is not created specifically for the user. In adaptation of web services, the behavior of a deployed web service is modified at runtime based on environmental considerations. Adapation already assumes that the available service met requester's requirement and we want to ensure this continues as the environment changes. Adaptation approaches can come in two main flavors – one where the primary aim is to compose and deploy a correct workflow, and adaptation is viewed as an afterthought [19]; and another where adaptation issues are viewed while composing workflows [20].

7 Conclusion

Motivated by the need to determine the overall impact of a new requirement in a complex SOA environment, in this paper, we introduced the notion of business driven customization of SOA. We introduced a formal model capturing properties and relationships of business objects and business processes, and their implementing services and messages. We discussed the implementation of our approach in multiple setting, and illustrate its application in an industry scenario with large number of services with complex characteristics (SAP). Though the approach was illustrated with web services, the approach is relevant for SOA in general. We believe that the work provides a valuable, explicit model of alignment between business processes to service-based IT implementations.

References

1. IBM-Global-Services: Accelerating business flexibility,while reducing costs, with composite business services (2007), `http://www-935.ibm.com/services/us/index.wss/offering/gbs/a1027243`
2. Richards, D., Sabou, M., van Splunter, S., Brazier: Artificial intelligence: A promised land for web services. In: The Proceedings of The 8th Australian and New Zealand Intelligent Information Systems Conference (ANZIIS 2003), Macquarie University, Sydney, Australia, pp. 205–210 (2003)
3. Fensel, D., Lausen, H., Polleres, A., Bruijn, J., Stollberg, M., Roman, D., Domingue, J.: Enabling Semantic Web Services: The Web Service Modeling Ontology. Springer, Heidelberg (2007)
4. Reichert, M., Dadam, P.: Adeptflex-supporting dynamic changes of workflows without losing control. Journal of Intelligent Information Systems 10(2), 17–93 (1998)

5. Ellis, C., Keddara, K., Rozonberg, G.: Dynamic change within workflow systems. In: COOCS, pp. 10–21 (1995)
6. Liang, H., Sun, W., Zhang, X., Jiang, Z.: A policy framework for collaborative web service customization. In: Proc. SOSE (2006)
7. Amazon: Amazon web services (Last Accessed June 2008), http://aws.amazon.com
8. Campbell, S., Mohun, V.: Mastering Enterprise SOA with SAP NetWeaver and mySAP ERP. John Wiley & Sons, Inc., New York (2006)
9. ten Teije, A., van Harmelen, F., Wielinga, B.: Configuration of web services as parametric design. In: Motta, E., Shadbolt, N.R., Stutt, A., Gibbins, N. (eds.) EKAW 2004. LNCS, vol. 3257, pp. 321–336. Springer, Heidelberg (2004)
10. Niemelä, I., Simons, P.: Smodels - an implementation of the stable model and well-founded semantics for normal lp. In: Fuhrbach, U., Dix, J., Nerode, A. (eds.) LPNMR 1997. LNCS, vol. 1265, pp. 421–430. Springer, Heidelberg (1997)
11. Gelfond, M., Lifschitz, V.: The stable model semantics for logic programming. In: Kowalski, R.A., Bowen, K. (eds.) Proceedings of the Fifth International Conference on Logic Programming, pp. 1070–1080. The MIT Press, Cambridge (1988)
12. van Gelder, A., Ross, K., Schlipf, J.S.: The well-founded semantics for general logic programs. Journal of the ACM 38(3), 620–650 (1991)
13. Martin, D., Paolucci, M., McIlraith, S., Burstein, M., McDermott, D., McGuinness, D., Parsia, B., Payne, T., Sabou, M., Solanki, M., Srinivasan, N., Sycara, K.: Bringing semantics to web services: The owl-s approach (2004)
14. East, D., Iakhiaev, M., Mikitiuk, A., Truszczyński, M.: Tools for modeling and solving search problems. AI Commun. 19(4), 301–312 (2006)
15. Hirsch, R.: Enterprise soa explorations: Options to deal with enterprise services that don't meet user requirements. Blog Entry at SAP sdn (2008), http://www.sdn.sap.com/irj/sdn/weblogs?blog=/pub/wlg/8665
16. SAP: Enterprise services enhancement guide (2007)
17. Chang, Y.C., Mazzoleni, P., Mihaila, G.A., Cohn, D.: Solving the service composition puzzle. In: Proc. SCC (2008)
18. Abiteboul, S., Amann, B., Baumgarten, J., Benjelloun, O., Ngoc, F.D., Milo, T.: Schema-driven customization of web services. In: Proc. VLDB (2003)
19. Au, T.C., Kuter, U., Nau, D.S.: Web service composition with volatile information. In: International Semantic Web Conference, pp. 52–66 (2005)
20. Chafle, G., Doshi, P., Harney, J., Mittal, S., Srivastava, B.: Improved adaptation of web service compositions using value of changed information. In: Proc. ICWS, Salt Lake City, USA (2007)