An Extensible Framework for Tracing Model Evolution in SOA Solution Design

Renuka Sindhgatta
IBM India Research Laboratory
Bangalore, India
renuka.sr@in.ibm.com

Bikram Sengupta
IBM India Research Laboratory
Bangalore, India
bsengupt@in.ibm.com

Abstract

Existing tools for model-driven development support automated change management across predefined models with precisely known dependencies. These tools cannot be easily applied to scenarios where we have a diverse set of models and relationships, and where human judgment and impact analysis are critical to introducing and managing changes. Such scenarios arise in model-based development of service oriented architectures (SOA), where a plethora of high-level models representing different aspects of the business (requirements, processes, data) need to be translated into service models, and changes across these models need to be carefully analyzed and propagated. To support the process of model evolution, we present an extensible framework that can automatically identify possible changes in any MOF-compliant model. Changes across different model types can be easily related through a user interface and via rules that are programmed at specified plug-in points. At runtime, when an instance of a model is changed, the framework performs fine-grained analysis to identify impacted models and elements therein. It also allows analysts to selectively apply or reject changes based on the specific context and summarizes the incremental impact on downstream elements as choices are made. We share our experience in using our framework during the design of a SOA-based system that underwent several changes in business models, necessitating changes in the associated service design.

Categories and Subject Descriptors D.2.2 [**Software Engineering**]: Design Tools and Techniques

General Terms Documentation, Design, Verification.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
OOPSLA 2009, October 25-29, 2009, Orlando, Florida, USA.
Copyright © 2009 ACM 978-1-60558-768-4/09/10…$10.00.

Keywords Model Driven Development; Model Transformation, Change Impact; Business Process; Service Design

1. Introduction

Service Oriented Architecture (SOA) helps in realizing business processes by assembling a set of services, where-in each service provides the functionality required for accomplishing a business task. SOA development methods [1, 2] propose a set of steps that help in identifying, specifying and realizing services. A Model Driven Development (MDD) approach for developing SOA solutions provides a common platform (based on conceptual models) for business analysts and application architects to exchange views and share understanding. High-level models representing the business domain are translated into a service model, which in turn is refined into lower-level design models (class diagrams, sequence diagrams etc) followed by implementation. A set of dependent models thus need to evolve through the development process.

At IBM Research, we are actively engaged in building tools and methodologies for enabling service-orientation. Our work is informed by the experiences we gather while working with IBM software architects on field engagements related to SOA. Such engagements typically begin with detailed modeling of the business domain, including the competencies and functions, business requirements, processes and enterprise information models. As these models evolve, abstract service specifications are derived from them, and are then refined over a period of time with more detailed specifications of operations, service messages and components. Frequently, these activities proceed hand-in-hand, with periodic sync-ing on major releases, when all changes to the various business models since the last release need to be consistently propagated to the service models. Once the business and service models start to stabilize (typically after 4-5 iterations), work on detailed design begins. While the volume of business-driven changes start to diminish thereafter, the impact of any change (e.g. necessitated by a new requirement) is significant and can cause major rework of the stable service models, and the design elements derived from them, thereby

necessitating careful analysis when introducing such a change.

Analyzing and propagating change across the many inter-dependent models spread over multiple layers, is thus a major activity in any MDD-based development of SOA. At the detailed design level, inter-model relationships are well-understood and there is sufficient tool-support for automated change management [14], and indeed, much of prior research has also focused on the same [4, 5]. For example, class diagrams and sequence diagrams are usually co-developed on a single tool, with sequence diagram lifelines and messages being created directly from classes and methods therein – thus a change in a method signature directly impacts the corresponding message in a sequence diagram. However, at the business and service layers, we discovered critical gaps in available tool support for change management. Different aspects of the business e.g. business processes, use cases, data etc are often modeled by different roles (business analysts, information architects, IT architects) using specialized tools [15, 16, 17] that may not integrate seamlessly with each other or with tools used for service and lower-level design. Some tools [13, 14] do allow business and service models to be created, or imported from other tools, but since these models are conceptually different, they become silo-ed with distinct modeling profiles created for them, with at best some coarse traceability links across the models and no native tool support for fine-grained change management or impact analysis. Model Transformations [6, 7] automatically transform a target model when a source model is changed and typically work well across low-level design and implementation elements with very precise inter-dependencies. However, they do not meet the needs for change propagation across more abstract business and service models, where human judgment is often necessary to select the right change alternative based on the business context and adopted SOA methodology, and impact analysis is critical prior to taking a decision to introduce a change.

These factors pose significant challenges for SOA practitioners, particularly because the number of business and service model elements can be very large in practice. For example, during our engagements, we have come across systems where process models, used as a primary input for identifying key services and their specifications, have over 50 processes that are progressively refined. Each such process consists of several tasks that consume or create data entities relevant to the enterprise; the sequence of tasks or the data can undergo changes in response to new business requirements. In the absence of adequate tool support, business analysts and architects use their domain understanding to analyze and manually propagate all such business changes downstream, but given the size of most of these models, this activity is labor-intensive and error-prone. Subsequently, substantial investment has to be made in (manual) model validation activities, often carried out by a separate quality team, to ensure that all the models are consistent. Needless to say, the overall approach is highly inefficient. Thus, change management across the business-service layers in SOA development calls for a mechanism that *(i) supports more flexibility in adapting to diverse business and service models and their relationships, (ii) provides greater automation for fine-grained change analysis and propagation across the models, while (iii) allows interactivity in support of human judgment and incremental impact analysis.*

Towards that end, we present a change management framework that addresses these challenges and has been motivated by our experiences in model-driven development of SOA solutions as described above. An overview of some of these models, and change management scenarios across them are presented in Section 2 to set the context. The key novelties of the framework are its flexibility and extensibility. As long as the models imported into the framework adhere to Meta-Object Facility (MOF) compliant meta-models [9], it can automatically generate all possible types of change the model elements can undergo, and supports an intuitive user interface using which change types across different categories of models may be linked. This flexibility is an important feature as the type of business models from which service models are derived may vary according to the methodology employed or the specific engagement context, and the platform should allow new categories of models to be incorporated when needed. At runtime, when an instance of a model is changed, the tool performs fine-grained analysis to identify impacted elements. Rules that help identify these elements are programmable, and can be incorporated into the framework through defined plug-in points. The framework also allows analysts to selectively apply or reject changes based on the context and summarizes the incremental impact on downstream elements as such choices are made. Technical details about the framework are provided in Section 3. We have instantiated the framework with some of the models and rules we have typically seen being used in SOA engagements and used it to manage the evolution of a SOA system that underwent several changes in the business process models, requiring changes in the associated service design. This case study is reported in Section 4. Section 5 presents some of the lessons learnt. Related work is discussed in Section 6, while Section 7 concludes the paper.

2. SOA Modeling Context

We illustrate the SOA modeling context with a widely used method for deriving services – Business Process Decomposition. We explain how the business and service entities, as

well as detailed design elements, are modeled and related. Finally, we discuss limitations in existing tools when it comes to managing changes across these models.

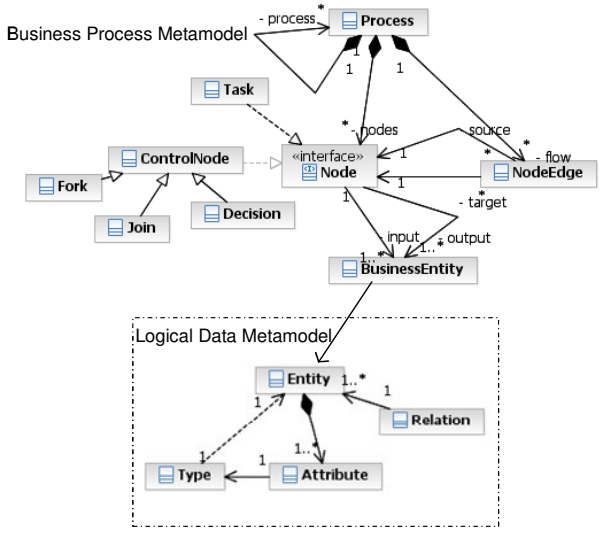

Figure 1. Business Process and Logical Data Metamodel

2.1 Business Modeling

A business process is a repeatable sequence of activities for delivering a service or a product to a stakeholder. A meta-model for a business process is shown is Figure 1. A *Process* consists of *Node*, that could be a *Task* or *ControlNode*, a *NodeEdge* and (sub)*Process*. Each Node has Input and Output, the NodeEdge representing the flow of data between a source Node and a target Node. Data is modeled as a *BusinessEntity*, which represents a business object relevant in the given domain. A business entity is refined by linking it to an entity in the logical data model (information model). An entity contains a set of typed attributes, which may include other entities and also has relations relevant to the enterprise undergoing SOA transformation. Note that a business process is likely to contain organization roles and events but for simplicity, we have not included them in the meta-model. Similarly, the logical data meta-model is also condensed highlighting only the key elements.

Figure 2 shows the example of a business process that provides an insurance quote for a vehicle. The figure depicts the steps (tasks) involved in authenticating the user, validating the policy information entered by the user and verifying if the risk can be accepted before deciding on issuing the quote to the user or notifying of rejecting the policy.

Note that apart from process and information models, there are other business models that are often useful in defining services. These models depend on the method used for service identification. For example, use cases models may be used to represent business functional requirements that serve as a source for service identification. Again, there are models [19] for representing the structure of the business in terms of domains, competencies and functional areas, which may be used to group together services once they are derived. For brevity, we do not provide details of these, but focus mostly on process and information models in this paper.

2.2 Service Modeling and Detailed Design

The business models provide the main inputs for the Service Model. A meta-model for the services layer is shown in figure 3. The meta-model comprises of a *CandidateService* indicating a functionality that is a candidate for being realized as a *Service*. A CandidateService that fulfills a *ServiceCriterion* is identified as a Service. A Service comprises of *ServiceOperation* with Input and Output Messages. A set of Services is realized by a *ServiceComponent*. A CandidateService also can be realized by a *FunctionalComponent* (e.g. Java components). The ServiceComponent here refers to the Component defined in the Service Component Architecture SCA standard [11].

A *subset* of the Service model for the example process model in Figure 2, is given in Figure 4. All the Tasks of the process model are initially defined as CandidateServices in the Service Model. A Service criterion is applied on each of the candidate services and a candidate service is either defined as a Service or realized as a Functional Component. In the example, the CandidateServices RetrieveLocationDetails, RetrieveCreditHistoryDetails and CalculatePremium are identified as Service and the other CandidateServices are implemented as Functional components. The CalculatePremium Service is realized by the PolicyManager Ser-

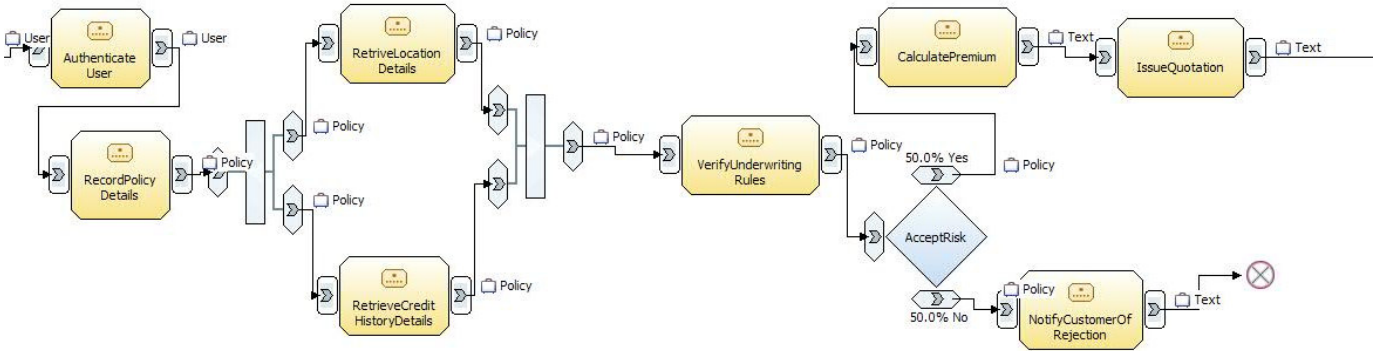

Figure 2. Example Business Process Model

viceComponent. Policy and User are the data (messages) that are used by the operations of the service elements.

A service is realized by a ServiceComponent through a set of classes. For each ServiceOperation there is an Interaction/Sequence diagram created detailing the interaction between the classes. Figure 4 depicts the class diagram for the service component PolicyManager. Since class diagrams and sequence diagrams are part of the well-known UML standard, we do not provide their detailed description here.

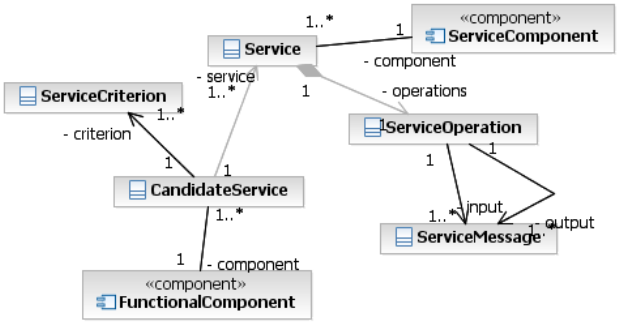

Figure 3. Service Metamodel

2.3 Managing Changes across the Layers

There is significant relationship between the business, service and detailed design layer entities that needs to be understood and used as the basis for change management across these layers. Usually, Tasks from the business process models form good candidates for services. Service operations are derived by identifying the functionality of the task the service is associated with. The business entities are transformed into service messages: the input and output business entities of the task become the input and output parameters of the service operation. Sometimes, however, a newly added business entity in the information model is realized more naturally as a (information) service – a service providing the basic CRUD (create, read, update, delete) operations on the entity. Some types of services are derived by analyzing business use case models such as services providing process performance reports, audit reports. These are often termed as visibility services with use cases as their primary inputs. A set of service components usually realizes a business functional area, and the component boundaries may change on business reorganization. Control nodes (Fork, Join) in the process model are used in the definition of service compositions and define BPEL [20] when translated to code. Similar to the Service meta-model, a service composition meta-model containing service flow, operations and messages is defined and semantically linked to the process model. Thus a change in the process model control flow impacts the associated service composition model and derived BPEL code. Finally, a change in the Service model may necessitate changes in the detailed design elements. For example, a change in one of the services realized by a Service Component may impact the class diagrams implementing the component. Similarly, a service operation change may lead to a change in an associated interaction diagram.

2.4 Limitations in Current Tools

Given the complex dependencies between the business, service and design models, any change originating in the business layer needs to be carefully analyzed and propagated, so that all dependent models are updated in a consistent manner. For example, let us suppose that in the Insurance Quote Issue process model (Figure. 2), there is a

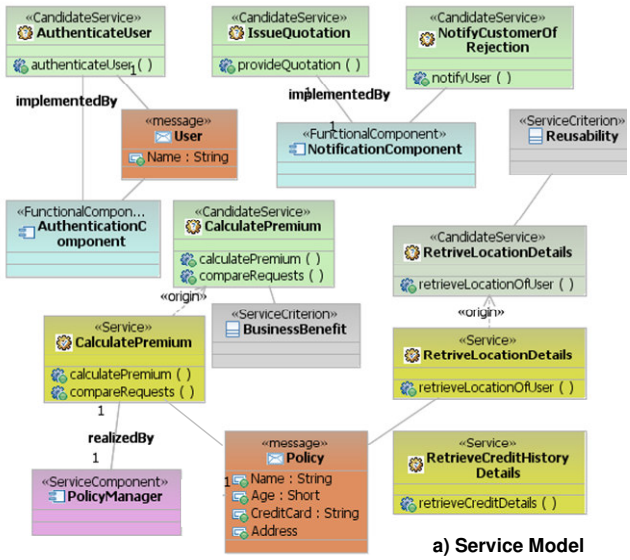

Figure 4. Example Service Model and Service Design

new requirement that mandates the processing of previous claim history. There is a new specification of Premium that needs to be provided by CalculatePremium task. There is also the need to send a quotation document that contains details of the quote in a given format. The changes made by the business analyst are:

- Add new task RetrieveClaimHistory to retrieve previous claim after RecordPolicyDetails and in parallel to RetriveCreditHistoryDetails
- Create a new BusinessEntity named QuotationDocument that contains details of the quote
- Create a new BusinessEntity named Premium
- Change the Output of CalculatePremium task that provide Premium as the output
- Change in the input and output of the task IssueQuotation that takes in the Premium as input and provides QuotationDocument as output

To address the above business changes, the architect would have to analyze and create a new candidate service to retrieve claim history, create new service messages corresponding to QuotationDocument and Premium business entities, and modify the input/output messages for services that correspond to CalculatePremium and IssueQuotation tasks. Moreover, if Quote information is used by several other Insurance applications, then it could potentially be used as an Information Service that other applications would use for accessing or updating – this is a decision the architect needs to take when introducing the change. Finally, for each change made at the service layer, the corresponding changes at the detailed design layer would need to be carried out.

Clearly, this is a non-trivial task and if carried out manually, the process is bound to become very laborious and error prone. Unfortunately, current tool support for analyzing and propagating changes across the different modeling layers in SOA design is fairly limited. Some tools [13, 14] can coarsely compare business/service model versions and detect that changes have occurred, but they are unable to filter out the important changes that will impact related models, classify changes according to types, or suggest consequent changes. These tools provide much more intelligent support for change management across class diagrams, sequence diagrams etc. within the design layer, but for other models, what they provide analysts are the results of a basic "diff" operation across models, including changes in documentation, package restructuring etc., most of which will not have a bearing on design of the services. The implicit semantic relationship between the business, service and design layer models as explained above, cannot be easily captured using these tools. It is left to the analysts to sift through the results of model comparison, detect important changes and manually propagate the changes to impacted elements. Note that some of the tools support automated model-to-model transformation techniques (e.g. MOF 2.0 Query/View/Transformation (QVT) [6]) for change propagation. However, these techniques do not provide opportunities for impact analysis prior to making a change, or allow user judgment in selectively applying/rejecting change alternatives at runtime. Hence they work well only for detailed design models with one-to-one mapping between elements, where the focus is mainly on automation and there is no need for human judgment and analysis

3. An Extensible Framework for Tracing Changes across Models

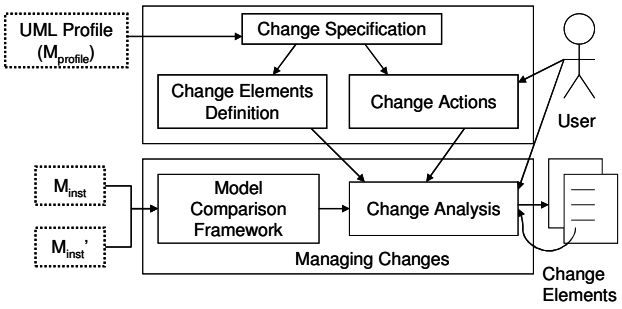

Figure 5. Change Analysis and Propagation Framework

We will now outline a framework for tracing changes across different types of models that addresses many of the challenges discussed above. Our approach has been motivated by the following design considerations. First, the framework needs to be *extensible;* it should not be hard-wired for a fixed set of model types, but should allow users to incorporate new types of models and dependencies with relative ease. Second, change detection and propagation within the framework should be *efficient*; the impact of a change can be detected or analyzed at multiple levels, but these should be grouped together with facilities for drill-down for finer-grained analysis as needed. Finally, the framework should be *interactive*; at run-time, users should have the option to selectively apply or reject changes based on the context, and view the incremental impact of those decisions on downstream artifacts.

The overall process of tracing changes across multiple types of models is supported within the framework as a sequence of two steps – Specifying Changes and Managing Changes. This is shown in Figure 5. The steps for specifying changes are

- Given a meta-model, atomic *change elements* representing all possible change types are automatically determined.

- Hierarchical relationship between atomic change elements are automatically identified to help analyze changes at multiple levels
- *Change actions* linking change elements of one meta-model to the change elements of a dependent meta-model are defined by a user through an interface, and through localized programming of rules at defined extension points.

The above steps can be performed in an "offline" mode, based only on the meta-models underlying the model elements of interest. At runtime, when the actual model elements undergo changes, the steps involved in managing the changes are

- Automatic comparison of two versions of a model and classification of the changes into atomic changes by identifying the change elements
- Identification of the change impacts on the dependent models using change actions.
- Upon selection of applicable change elements by the user, incremental analysis of the impact of the change elements on dependent model elements.

We will now explain the above steps in more detail.

3.1 Specifying Changes

This represents the pre-processing steps that needs to be done only once per meta-model (to identify and relate change elements within the meta-model) and once for every pair of dependent meta-models (to relate change elements across meta-models and generate change actions).

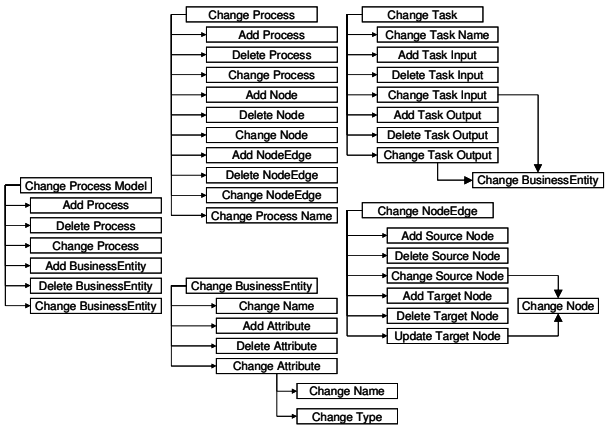

Figure 6. Change Elements and Dependencies for Business Process Metamodel

3.1.1 Identify Change Elements for a Meta-Model

Given a meta-model, we define all changes to the meta-model instance as *ChangeElement*. A change element *ce,* when applied on an instance of the model element $m \in M$ (meta-model instance) results in *m'*. For each change element, a change parameter is defined. As each change element is associated to the attribute of the model element that changes, the attribute forms change parameter *p* such that *ce (m, p) = m'*. For example a change element 'ChangeTaskName' referring to a change in the name of a Task, would require a change parameter – the string value for changing the name.

3.1.2 Identify Dependencies between Change Elements

Change Elements within a meta-model instance have dependencies. Identifying these dependencies helps in determining the order in which the change elements can be applied on a Model M to get the version M'. The dependencies may be automatically arrived at using the dependencies of the model elements. A subset of dependencies between change elements for the business process meta-model (Figure. 1) are depicted in Figure 6. Each leaf node represents a change to the corresponding model element attribute or property. The non-terminal nodes represent change elements that occur only when the leaf node change elements are present e.g. 'ChangedProcessModel' $\prec$ 'ChangedTask' $\prec$ 'AddTaskInput' i.e. a change in a business process may be induced by a change in a constituent task, which in turn may be caused by the addition of a new input to the task. Thus the impact of a change may be considered at multiple levels of granularity.

3.1.3 Define Change Actions Representing Impact of Changes

A change action defines the impact of a change element on other model elements – either in the same model or a de-

1. Name=AddBusinesEntityAction
Input Change Element = AddBusinessEntity
pre-condition = None
Output Change Element = AddCandidateService, AddServiceMessage

2. Name=ChangeInputOfTask
Input Change Element = ChangeTaskInput
Output Change Element = ChangeService, ChangeServiceOperation, ChangeInputParameter
pre-condition =
CandidateService isLinkedTo Task *AND* (Input BusinessEntity isLinkedTo ServiceMessage)

3. Name=Change of Service Operation
Input Change Element = ChangeServiceOperation
Output Change Element= Change Lifeline, ChangeMessage(InteractionDiagram)
pre-condition = Service isLinkedTo Lifeline AND ServiceOperation isLinkedTo Message(InteractionDiagram)

Figure 7. Example Change Actions

pendent model. A change action takes a set of input change elements of a model, validates the input change elements against a set of pre-conditions and generates a set of associated output change elements. A change action ca_x can be represented as an inference rule that verifies a set of (pre-) conditions on change elements in the source model ($ce_{s1},\ldots ce_{sn}$) and infers the resulting change elements in the target model ($ce_{t1}, \ldots ce_{tm}$).

$$ca_x = \frac{\{cond_{s1}\}ce_{s1},\{cond_{s2}\}ce_{s2},...\{cond_{sn}\}ce_{sn}}{ce_{t1},ce_{t2},...ce_{tm}}$$

Example change actions between process, service and design model change elements are shown in Figure 7. A change action could result in one or more output change elements provided the stated pre-conditions are met.

3.2 Managing Changes

Once the change elements and change actions are specified, the next step is to detect, analyze and propagate changes across the models. These steps are explained next.

3.2.1 Detect Changes in a Model Instance

Given a model and its modified version, the changes are detected by comparing them. The basic support for this is available in most modeling tools such as Rational Software Architect [14]. However, the detected changes need to be interpreted and classified into atomic change elements; changes that do not relate to defined change elements need not be considered for analysis and propagation. The composite change elements are built using the dependencies between the atomic change elements.

3.2.2 Identify Change Actions and Analyze Impact

Given the list of change elements, the applicable list of change actions is identified. The algorithm works such that the array of change elements CE containing the source model changes is retrieved. All the possible change actions CA containing change elements of CE as input elements are retrieved. For each change action, the pre-conditions are verified. The execution requires extracting the target element and change parameters for creating the output change element. It is possible to transitively study the impact of the newly created output change elements. In such case, the algorithm is re-run till there are no more change elements added to CE. It may also be required for the user to do an incremental analysis, in which case, only based on the user's input, further change actions are executed. This occurs, for example, when there may be more than one possible change action for a given change element, requiring the user to make a choice.

```
AnalyzeImpactFor CE= {ce1,…,cen}
1 Do
2   Get CA= {ca1,…,can} where cei∈ INPUT(caj)
3   For each caj ∈ CA
4       verify preCondition(caj)
5       getTargetElementAndParameter (cei)
6       create cek ∈ OUTPUT (caj)
7       Add cek to CE
8   End for
9 Until (No changes to CE)
```

Apart from the downstream ripple effect of change from the business to the service layer, there is often a need for bi-directional change propagation. In the domain of SOA, this happens, for example, when services – as re-usable entities - are used to support multiple business tasks. A change to a task results in a change to the service. This change to the service would in turn impact the other dependent task supported by the same service. To realize this, change actions for the reverse direction may also be defined during the change specification phase. A default pre-condition is tested that prevents cyclic execution of change actions. This requires each change element to keep a list of the source change elements that caused it. The pre-condition checks if the model element of the input change element is different from model element of the output change element.

Finally, once all the relevant change elements across the different model artifacts have been generated, the end-to-end impact of the top-level changes is known. The analyst can then choose to apply the selected changes on the appropriate model elements. The change elements in our framework have been designed to automate this process, since the type of change to apply, the change parameter, and the model element on which to apply the change are already captured within each change element. This completes the process of managing changes across the different model layers.

3.3 Implementation

Rational Software Architect (RSA) [14] provides a mature environment for designing SOA solutions and is built over the Eclipse platform supporting plug-in development. Our change analysis and propagation framework is an RSA plug-in. The UML representation of the key design elements of the plug-in is presented in Figure 8. The Chan-

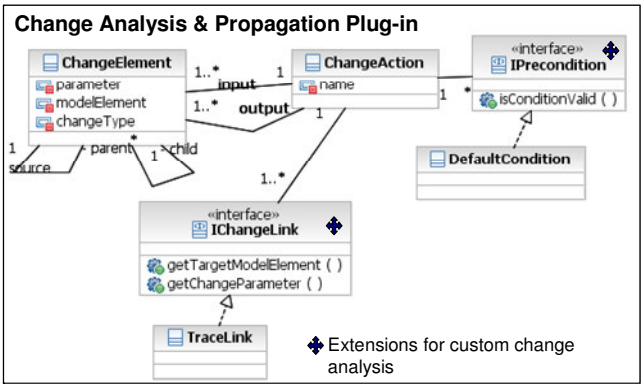

Figure 8. Extensions for Change Analysis

geElement stores details of the model element it is associated with, the parameter and the source change that caused it. It also contains the dependency (parent-child relationship). The ChangeAction contains the input and output ChangeElement(s). ChangeLink at runtime identifies the model element in the target model to which the change element needs to be associated with. It also processes the parameter of the source change elements and creates the change parameter of the target change element. The pre-conditions are validated by implementations of IPrecondition. In our implementation, we used EMF APIs [13] to validate the pre-conditions (IPrecondition) and extract re

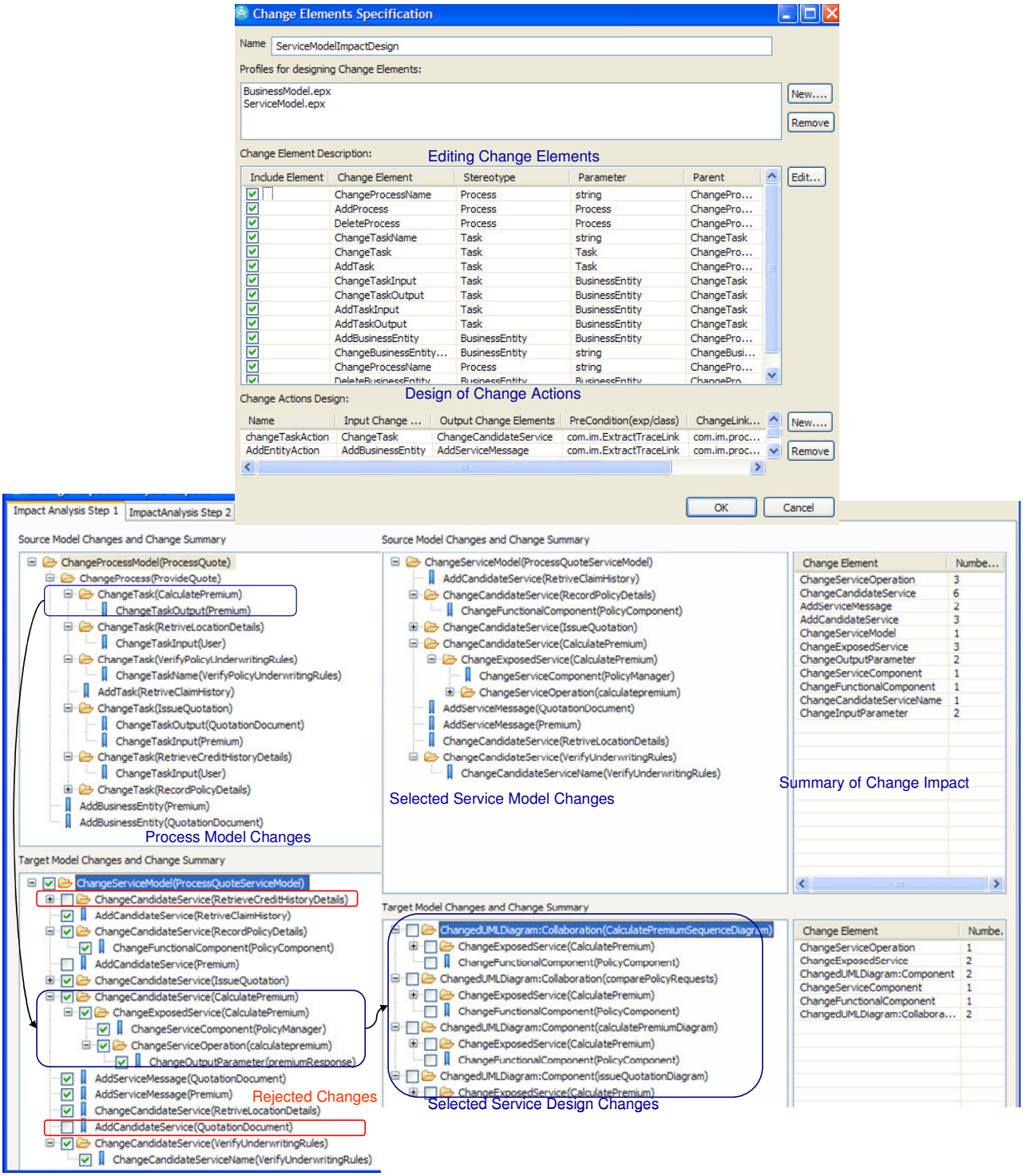

Figure 9. Change Specification and Analysis Framework

lated model elements and parameters (IChangeLink). However, it can also be an OCL [12] expression.

Figure 9 shows screenshots from our framework implementation. First, UML profiles of interest (e.g. "BusinessModel.epx", "ServiceModel.epx") are uploaded and automatically analyzed to detect all possible change elements. The user may select the change types of interest and those not selected are removed from further analysis ("Editing Change Elements"). Next, the user defines change actions; the input and output change elements are specified and appropriate implementations of IPrecondition and IChangeLink may be linked at this stage. Finally, a user can select two versions of the model and invoke the analysis. The model versions are compared and change elements are detected. For example, in Figure 9, changes made to the process model "ProcessQuote" are identified. All the possible resultant changes on "ProcessQuoteServiceModel" are computed and presented. The user selects few of the changes, which are considered for further analysis. The user discards changes to the Service "RetrieveCreditHistoryDetails" as it is an existing implemented service that cannot be changed. Similarly, "Add Candidate Service (Premium)" is discarded as the user wants to implement the business entity as a message and not as an information service. This change does not appear in the class or the sequence diagrams related to calculatePremium service. The impact analysis tabs refer to cascading impact analysis. Each tab refers to a change in a model and its impact on the dependent model - Process Model change impact on Service Model and Service Model change impact on design model. A summary of the change elements indicating the number of types of changes is also presented.

Development Artifact	**Role**	**Tool**
Business Process Model	Business Analyst	Websphere Business Modeler
Logical Data Model	Information Architect	Rational Data Architect
Service Model	IT Architect	Rational Software Architect (with Service Model Profile)
Service Design	IT Architect	Rational Software Architect
Service Implementation	Web service developer	Websphere Integration Developer

Table 1 Artifacts, Roles and Tools used in SOA Solution Development

4. Case Study

We applied our change impact analysis framework on a SOA design project developing services for Maintenance of Monitoring Equipment functional area in the Chemical and Petroleum Domain. The objective of the project was to identify reusable services in the business domain and to further design and implement them. Table 1 lists the development artifacts, the role of the users building the artifacts and the tools used. Process models were authored using Websphere Business Modeler [15] and Logical Data model modeled in Rational Data architect [16]. Both the models can be imported into RSA for service identification. The Service Model is built using the SOMA-ME [2]. RSA further supports UML 2.1 and model transformations. Its UML to SOA transformation transforms UML design artifacts to generate BPEL, XSD and WSDL. We use this transformation to create the code artifacts from the UML Service Model.

The process model consisted of 10 processes each containing an average of 4-5 tasks. It went through several minor refinements and 3 major releases in 5 months time period. We extracted the models after the first check-in and at the major releases of the models. The model changes were identified, analyzed for impact and then propagated, all using our tool. The key change elements were defined

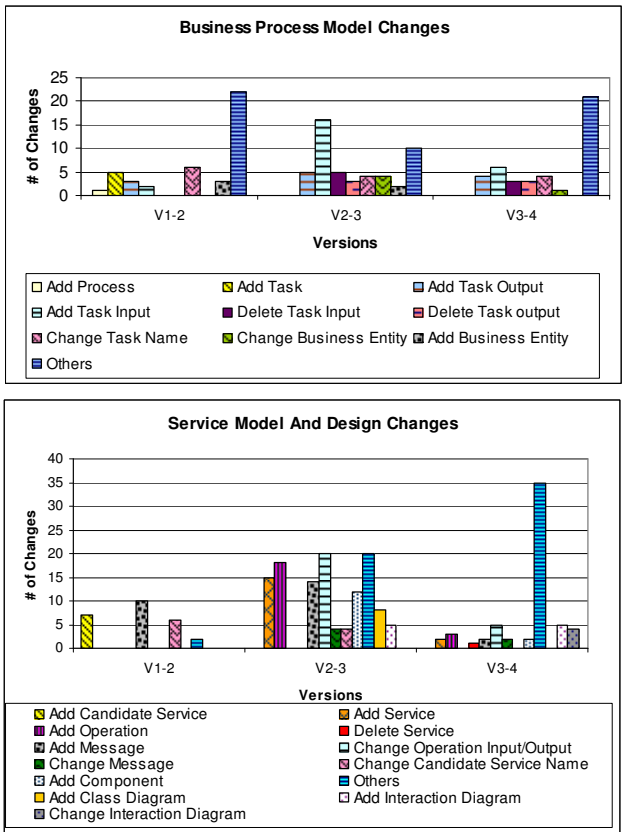

Figure 10. Changes across different versions of Process, Service and Design Models

by us in consultation with the architect. The change actions were also defined based on the understanding of the semantic linkages between the process model, logical data model and the service model. We identified 20 change actions for business process model and service model. There were 3 information services defined using the logical data model having 23 Entities.

Figure 10 shows the classification of changes across different versions of the process, service and design models. Only the key change elements have been highlighted in the figure (as the total number of change element types is high). The figures indicate a majority of the changes happen during one or two intermediate versions (in our case version 3). From then on, the models become relatively stable and the changes are primarily related to packaging and documentation (under "Others").

In the first revision of the service model, the impact of the process model changes on the service model was limited to 25 changes. As the service model in its initial version only contained the list of candidate services, the impact of changes was low. Only two types of changes were recorded – Addition of candidate services and Addition of service messages. 'Add Task' change in the process model caused 'Add Candidate Service' in the service model. There were two additional Candidate services that were added as a result of using the use cases and information model. 'Add Task Input' and 'Add Task Output' did not result in any change to the Service model as the model in its initial version did not have services and their specification defined.

However, as the service model evolved, service and service operations definition increased the number of impacted elements in the next revision, where around 100 changes were recorded. 'Add Task Input' and 'Add Task Output' changes in the process model led to 'Change Operation Input/Output' for the services that had the operations defined. Further, several messages and operations were defined during this version. This version also has the design level changes – addition of components, changes to class and interaction diagram.

In the final revision of the service model, there were a total of 60 changes, 35 of the changes were due to addition of descriptions (documentation) to services and messages. The other 25 changes were due to modifications made to messages, changes to the class and sequence diagrams and changes to operations. As we see, most of the changes are related to the design elements.

The analysis performed by our tool based on defined change elements allowed us to quickly zoom in on the change types that really matter to the architects. The ability to select or reject a change was useful as some changes required manual decision making. Change actions helped in automatically propagating the impact of selected change elements. Finally, model validation activities post change management is made largely redundant, as change propagation is based on rules designed by the architects themselves, and additionally, user inputs are taken at runtime whenever required. All of these lead to significant gains in productivity.

5. Lessons Learnt

We will now briefly outline some of the lessons we learnt from our experiences in developing and using the framework.

5.1 Constraints based Change Analysis

There are certain scenarios, where a change to the service specification may not be possible – for example when using a third-party service, it may not be possible to vary the implementation of a service. There may also be regulatory compliance (HIPAA, Sarbanes-Oxley) that mandate some specifications to be adhered to, and consequently, prohibit certain changes. It would be useful if such information can be defined and stored in the process or service models as constraints. Subsequently, variations of a process model that cannot be realized by 'in-variant' services can be pointed out to the architect during the change analysis. A set of alternate change element(s) can be provided to the user. For example, if a user is cannot make changes to a service or its operation, a list of relevant change elements such as 'Add Service', 'Add Operation' can be provided and the change can be propagated. This will help ensure all changes have been propagated to the dependent models.

5.2 Model Comparison

Model comparison frameworks use merge/match and differencing technique to identify changes between two versions of a model. We discovered that these matching techniques vary in different tools. RSA uses unique model identifiers to identify matching element. While the comparison framework is accurate as it relies on these unique IDs, in scenarios where model identifiers change (as a result of importing and exporting into different formats), the comparison fails. On the other hand, Eclipse EMF Compare uses name, type, content and relations to match the model elements of two versions. This offers more flexibility, although it can fail in the (rare) scenario where all these attributes change substantially. While our current framework uses RSA's model comparison techniques, in future, we would like to evaluate use of the Eclipse Compare framework in more detail.

5.3 User Experience

While the interactivity offered by the tool in terms of accepting user inputs and incrementally propagating change is useful, the overall user experience can be improved. For example, an architect, before applying a change, would like to analyze what has caused the change. While the list of source and target change elements are provided by the

framework for user inspection, the correspondence between pairs of these is not directly evident at present. Also, in some cases, it will be useful to record the reason for accepting/ rejecting a change, for example, why a new business entity is (not) being realized through an information service. A log of these design decisions will help other architects in further evolving the system. We will be incorporating such features in future versions of the framework.

6. Related Work

Briand et. al define Horizontal Impact Analysis (HIA) and Vertical Impact Analysis (VIA) for UML Models [5]. HIA focuses on changes and impacts at the same level of abstraction, and this corresponds to change impacts within a model, whereas VIA focuses on changes at one level of abstraction and their impacts at another level of abstraction (e.g. classes and sub-classes). In their initial work [4], the authors suggest a detailed analysis of the changes, organized in change taxonomy, to precisely study how changes propagate. Change impact analysis rules are defined that help in analyzing the impact of a change. Impact analysis is done in the context of UML class, sequence and state chart diagrams. Similarly, recent work by Ravichandar et. al [10] on change propagation defines a set of inferences rules between use cases, sequence diagrams and service specification (class diagram). The relationship between use cases and sequence diagrams is identified and rules that propagate the changes are defined. With reference to SOA design and development, a relevant work on managing changes in SOA-based solutions [3] discusses how to model the impact of a change in one design artifact upon the others. The authors discuss generic guidelines for assessing changes (variations) and their impacts to the related artifacts.

In contrast to the above approaches which primarily explore change rules in the context of specific types of models, the main goal of our work is to provide an extensible, efficient and interactive framework for identifying and propagating fine-grained changes across arbitrary model types (adhering to the MOF standard) and performing incremental impact analysis of changes. We have shown how the framework may be configured to trace changes across business, service and design layers, leveraging the MOF profiles of the relevant models, and by defining change actions based on the semantic relationships between these layers. In particular, the detailed study of change relationships between process, data and service models (Section 2) is another useful contribution of this work, since prior art has mainly focused on change propagation rules between design elements ([4, 5]), or on coarse-level relationships between process and service models as defined by Mazzoleni et. al. [18].

7. Conclusions

In this paper, we have presented a change analysis and propagation framework motivated by our experiences in model-driven development of SOA solutions, involving a variety of business, service and design models. The framework, developed as plug-in to the IBM Rational Software Architect (RSA), is extensible and can automatically identify possible change types in any MOF-compliant model. Changes across different model types can be easily related through a user interface and via rules that are programmed at specified plug-in points. At runtime, when an instance of a model is changed, the framework performs fine-grained analysis to identify impacted models and elements therein. It also allows analysts to selectively apply or reject changes based on his/her understanding of the context and summarizes the incremental impact on downstream elements as choices are made. We have shared our experience in using the framework during the design of a SOA-based system that underwent several changes in business models, necessitating changes in the associated service design. In future, we also plan to extend support of the tool for defining linkages between service model and test cases that would allow an end-to-end analysis of the impact of a change in the business requirement on the design, implementation and testing of Services.

References

[1] Ali Arsanjani, Abdul Allam: Service-Oriented Modeling and Architecture for Realization of an SOA. IEEE International Conference on Services Computing. pp 521, 2006

[2] Zhang L-J et al. SOMA-ME: A platform for the model-driven design of SOA solutions, IBM Systems Journal, Vol 47, Number 3, 2008.

[3] L.-J. Zhang, A. Arsanjani, A. Allam, D. Lu, and Y.-M. Chee. Variation-oriented analysis for SOA solution design. IEEE International Conference on Services Computing, pp. 560–568, 2007.

[4] Briand L. C., Labiche Y., O'Sullivan L. and Sowka M., Automated Impact Analysis of UML Models, Journal of Systems and Software, vol. 79 (3), pp. 339-352, 2006.

[5] Briand L.C., Labiche Y, and Yue T. Automated Traceability Analysis for UML Model Refinements. Technical report, Carleton University, TR SCE-06-06, Version 2, 2006.

[6] Query/View/Transformation. QVT-Merge Group, version 2.0 (2005-03-02), 2005. http://www.omg.org/cgi-bin/apps/doc?ad/05-03-02.pdf

[7] Ivkovic, I. and Kontogiannis, K. Tracing Evolution Changes of Software Artifacts through Model Synchronization. In Proceedings of the 20th IEEE international Conference on Software Maintenance, pp. 252-261, 2004.

[8] Anneke Kleppe, Jos Warmer, Wim Bast,: MDA Explained, The Model Driven Architecture: Practice and Promise, , Addison-Wesley, 2003, ISBN 0-321-19442-X

[9] OMG. Meta object facility (mof) specification version 1.4.Technical report, Object Management Group (OMG), http://www.omg.org/mof/

[10] Ravichandar R, Nanjangud N Ponnalagu K, Gangopadhyay D, Morpheus: Semantics-based Incremental Change Propagation in SOA-based Solutions, IEEE International Conference on Services Computing ,pp.193-201, 2008

[11] Service Component Architecture (SCA)- http://www.osoa.org/display/Main/Service+Component+Architecture+Specifications

[12] OMG, OCL 2.0 Specification, Object Management Group, Final Adopted Specification ptc/03-10-14, 2003.

[13] Eclipse Foundation, UML2: EMF-Based UML 2.0 Metamodel Implementation,www.eclipse.org/uml2

[14] Rational Software Architect, http://www-01.ibm.com/software/awdtools/architect/swarchitect/

[15] Websphere Business Modeler, http://www-01.ibm.com/software/integration/wbimodeler/

[16] Rational Data Architect, http://www-01.ibm.com/software/data/integration/rda/

[17] Telelogic System Architect, http://www.telelogic.com/products/systemarchitect/systemarchitect/index.cfm

[18] Mazzoleni, P. and Srivastava, B. Business Driven SOA Customization. 6th international Conference on Service-Oriented Computing, pp. 286-301, 2008.

[19] Flaxer, D. Anil Nigam Vergo, J. Using component business modeling to facilitate business enterprise architecture and business services at the US Department of Defense, ICEBE, International Conference on eBusiness Engineering, 2005.

[20] Web Services Business Process Execution Language Version 2.0 http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.pdf.