Combining Quality Assurance and Model Transformations in Business-Driven Development

Jana Koehler, Thomas Gschwind, Jochen Küster,
Cesare Pautasso, Ksenia Ryndina, Jussi Vanhatalo, Hagen Völzer

IBM Zurich Research Laboratory
8803 Rüschlikon, Switzerland

Abstract. Business-driven development is a methodology for developing IT solutions that directly satisfy business requirements. At its core are business processes, which are usually modeled by combining graphical and textual notations. During business-driven development, business process models are taken to the IT level, where they are implemented in a Service-Oriented Architecture. A major challenge in business-driven development is the semantic gap between models captured at the business and the IT level. Model transformations play a major role in bridging this gap.
This paper presents a transformation framework for IBM WebSphere Business Modeler that enables programmers to quickly develop in-place model transformations, which are then made available to users of this tool. They address various user needs such as quickly correcting modeling errors, refining a process model, or applying a number of refactoring operations. Transformations are combined with quality assurance techniques, which help users to preserve or improve the correctness of their business process models when applying transformations.

1 Introduction

Traditionally, the models of a business process and its implementation in an information system are considered separate artefacts. A business process model, in the best case, serves as documentation for the implemented system. However, as business process models and their implementation evolve independently, they quickly become inconsistent with each other.

Today, an increasing pressure from regulations combined with opportunities provided by new technologies such as those related to Service-Oriented Architecture [1] require models to reflect the reality of the implemented business processes. Furthermore, implementations should be derived more directly from business needs, which is often referred to as business-driven development [2–4]. Consequently, modeling tools increasingly address the transition from business to IT and vice versa. We observe two major trends. On the one hand, *quality assurance* strives to enable users to create business process models of higher quality from which correct, executable code can be obtained in a lean development process. On the other hand, *model transformations* aim at automating the transition across the semantic gap between business and IT. Both trends reflect the need to make modeling a less heavy-weight activity with the vision of moving towards more *agile* modeling tools where users can quickly respond to change

in processes and systems, where they obtain immediate feedback on the quality of the models and receive help to build software from models in shorter iterations.

The need to constantly adapt and revise process models because of unforeseeable changes leads to an increased interest in providing users with pre-implemented model transformations that enhance the usability of the modeling tools and the productivity of the user. For example, many of the business-level modeling tools available today allow users to generate part of the implementation on the basis of the processes modeled. This comprises the generation of Web service descriptions, usually represented in the Web Service Description Language (WSDL) [5], and of the corresponding service orchestrations represented in the Business Process Execution Language (BPEL) [6]. Many process modeling tools give a lot of freedom to business analysts, which may even include the possibility to define their own extensions to the modeling language. The interpretation of such extensions often lies with the user and is thus not accessible to the modeling tool, making code generation difficult or even impossible. Moreover, technical details that are required at the IT level are usually missing in models drawn by business analysts.

To address this business-IT gap, modeling tools begin to constrain business users by imposing a service-oriented and more technical modeling style so that technical details must be added to the models in a refinement step. Frequently, however, business analysts have difficulties in providing this kind of information. This requires a tool-supported hand-shake between business and IT that is not yet very well understood. It seems that this hand-shake comprises a continuum in which a business process model is refined from an *analysis model* to a *design model* and then further into *executable code*. To develop and provide the necessary refinement, refactoring, and abstraction operations in a modeling tool, a model transformation framework is needed that is seamlessly integrated into the modeling tool's architecture.

In this paper, we discuss such a model transformation framework that we developed for IBM WebSphere Business Modeler [7], an Eclipse-based commercial product for business process modeling and analysis. In Section 2, we relate this framework to other frameworks developed in the academic community and discuss our requirements. In Section 3, an example scenario is introduced and the challenges encountered during the transition from a business process analysis model via a business process design model to executable code are reviewed in more detail. Section 4 explains why linking model transformations with quality assurance is essential for their success in a commercial environment and gives a short overview on our quality assurance techniques. In Section 5, the transformation framework is described in more detail. An overview of a selected set of model transformations that we implemented is given and shortly evaluated in Section 6. Section 7 concludes the paper.

2 Approaches to Model Transformation

Model transformations are key to success of the Model Driven Architecture (MDA) initiative [8] by the Object Management Group. Consequently, a considerable amount of research is devoted to the area of model transformations. Different types of model transformations are distinguished in the literature [9–11]: When the source and target

models belong to the same metamodel, one usually speaks of *endogenous transformations*, otherwise of *exogenous transformations*. The former is more relevant in our case, because our transformations mostly address the transition from the analysis to the design model of a business process, which we currently consider to be represented by the same metamodel. Exogenous transformations are typically used when mapping models across domains, e.g., when generating code from business process models. Endogenous transformations can be further classified depending on whether the source and target models are physically the same or belong to a separate model. *Out-place* transformations produce a new target model, whereas *in-place* transformations modify the source model. A *vertical* transformation transforms abstract models into more refined models or vice versa, whereas a *horizontal* transformation remains at the same abstraction level. Typical examples of vertical transformations are refinement and abstraction, whereas refactoring is a semantics-preserving horizontal transformation. Research has also distinguished *destructive* and *non-destructive* transformations [10]. A destructive transformation can delete existing model elements, but a non-destructive transformation can only add elements.

For our work, it is important that all types of transformations can be implemented using the framework presented in Section 5. However, in-place transformations play a major role, because they meet our requirements of volatility and rapid execution when transforming models that share the same metamodel. *Rapid execution* is important to provide immediate feedback to users about the results of a transformation. *Volatility* of transformation results enables users to quickly undo (completely or partially) a transformation that was incorrectly applied. Once users are satisfied with the result of a transformation, they can persist the modified model. A transformation should be *applicable to an entire model or to a part thereof*, as indicated by the current user selection. Another requirement for our framework is its *extensibility with new transformations*, i.e., adding new transformations should be easy for developers. The framework should also enable *full integration* of the transformations with the modeling environment so that choosing and running a transformation does not require more than a few mouse clicks and users perceive transformations as being part of the normal editing flow. Finally, to facilitate a possible shipping of the framework in a future version of IBM WebSphere Business Modeler, the product team emphasized the importance of architecting a *lightweight* framework that does not significantly extend the tool's code base.

Several of the Eclipse-based transformation frameworks developed by the academic community provide features that are relevant to our requirements. These are in particular approaches that provide the compilation of transformations and combine declarative with imperative approaches. For example, Biermann et al. [12] present a transformation framework for defining and executing in-place transformations defined visually by graph transformation rules. Transformation execution relies on an interpretation by the AGG graph transformation tool [13] or compilation of the rules into Java. The ATL approach [14] allows developers of transformations to declaratively specify rules in textual form. In addition, it provides imperative constructs to specify the control flow of rule executions. Rules are compiled into byte code for execution on a virtual machine. Mens [15] describes a refactoring plug-in for the Fujaba environment which allows users to interactively call refactorings such as pulling up a method. MATE [16], which

is implemented using the Fujaba tool, links model transformations with model analysis to provide users with repair and beautifier operations for MATLAB, Simulink, and Stateflow models. Furthermore, many frameworks provide debugging support specific to transformations.

Our transformation framework does not provide a general solution in the sense of those sophisticated frameworks developed by the academic community, such as for example ATL [14], VIATRA [17], GreAT [18], ATOM3 [19], BOTL [20], Fujaba [21], and SiTRA [22]. Our solution only focuses on transforming business process models given in a specific metamodel used within a specific tool. As such, implementing or using the QVT standard [23] was also not in our focus.

Our transformations are written by specialized developers and currently cannot be composed by business users into larger composite transformations. A simple recording feature that allows them to generate sequences of transformations that require no further user input is nevertheless straightforward. However, providing business users with composition support for iteration and branching as is available in other transformation frameworks seems to require the exposure of transformation rules at the business level. An interesting alternative would be the learning of transformations by observing a user and generalizing her editing operations on a business process model. A first discussion of such an approach has been described by Varró [24].

The example transformations that we discuss in this paper not only illustrate that the entire spectrum of transformations is needed during business-driven development, but also illustrate the necessity to *combine model transformations with quality assurance*. Assuring the quality of the model resulting from the transformation is especially important when transforming models describing complex behavior, because errors such as deadlocks can only occur in behavioral models, but not in static models such as class diagrams. This means that the pre- and postconditions of a transformation involve a very elaborate model analysis, see Section 4. Formulating these conditions declaratively in a transformation framework such as those mentioned above has not yet been achieved, although it would have significant advantages, e.g., in documenting and analyzing transformation code, to study termination and confluence [25, 26], or test transformations [27]. To obtain clarity whether the combination of transformations with quality assurance would be possible in transformation frameworks, requires further investigation.

3 A Refinement Scenario

An analysis model of a business process as captured by a business analyst is shown in Fig. 1. The process model describes the (very simplified) handling of claims by an insurance company. First, the claim needs to be recorded, followed by a subprocess during which the coverage of the claim by the insurance policy of the customer is validated. If the customer has no coverage, then she is simply notified of this fact. Otherwise, the company continues to investigate whether the claim can be accepted. If the claim is accepted, a subprocess to offer a benefit is entered, which leads to the final settlement of the claim. If the claim is rejected, three activities take place in parallel: information in the Customer Relationship Management system (CRM) is updated to inform the cus-

tomer about the rejection, the decision is documented, and the case is sent to product development.

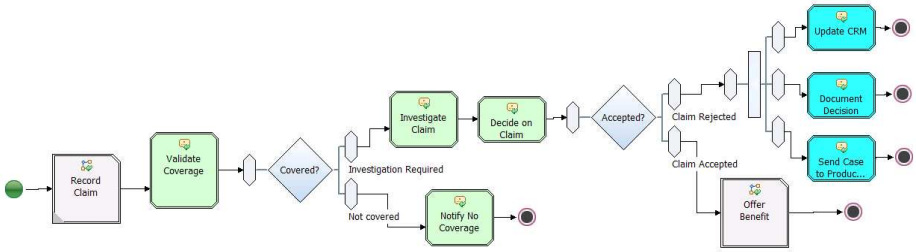


Fig. 1. Analysis model of a simplified claim-handling process in insurance.

Figure 1 only shows the control flow and omits many details of the process such as the data that is involved or the organizations responsible for various activities in the process. We notice that two decisions *Covered?* and *Accepted?* lead to exclusive choices in the process flow. If the claim is covered, but rejected, the process forks into a parallel process fragment containing three activities. This control flow leads in total to five branches in the process model, which consist of a mixture of sequential and parallel execution threads. Each branch ends individually with a single stop node. As soon as one of these stop nodes is reached, the entire process will terminate immediately. This can lead to problems for activities in the process that are executed in parallel as they may not have completely finished when a stop node terminates the process. Although this termination behavior was probably not intended by the business analysist drawing the process model, it can often be observed in real-world process models where it is caused by a modeling practice of not re-joining parallel branches [28].

In the implementing process, this modeling problem should be corrected. The BPEL process must end with a single reply followed by a single stop node to provide the result of the process back to the invoking service. Figure 2 shows an example of BPEL code generated by automatically exporting the process model, i.e., by applying an exogenous transformation from the business-process metamodel to the BPEL metamodel, which we do not consider further in this paper.

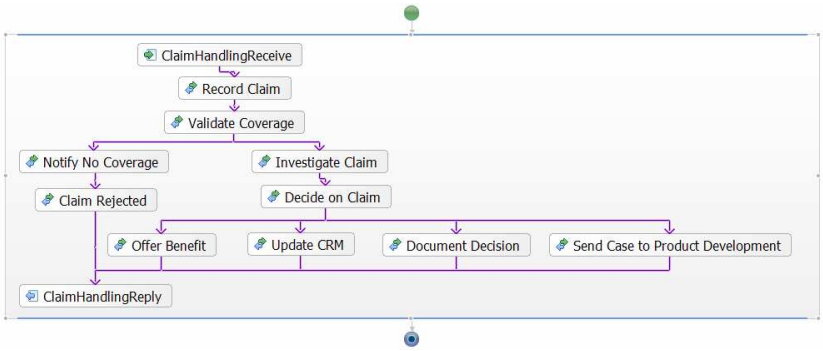


Fig. 2. BPEL code using “link style” generated from the analysis model.

A link-based flow is used and the five process branches directly enter the *ClaimHandlingReply* at the end of the process. The join condition that combines these five links at the reply needs to be associated with the correct AND logic for the three right links combined with the XOR logic for the two left links. Automatically deriving the correct join condition requires to analyze the control flow of the process model.

The BPEL code in Fig. 3 uses a block style with explicit *switch* actvities for the two exclusive decisions and a *flow* activity encapsulating the three parallel process steps in case the claim is rejected. This notational variant makes the BPEL flow logic more readable. In this variant, an analysis of the process model is required that determines the scope for the *switch* and *flow* activities of the process.

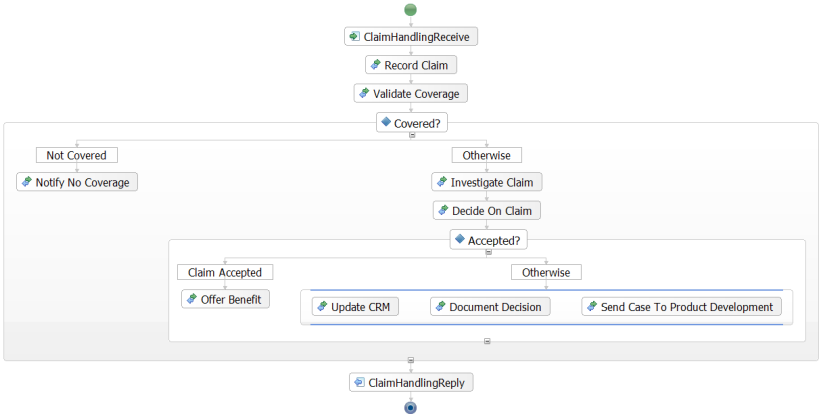


Fig. 3. BPEL code using “block style” generated from the analysis model.

Even in this very simplified example, both BPEL versions already show a slightly changed process model at the IT level, which corrects the termination behavior of the analysis model. Thus, the analysis model is no longer consistent with its implementation. Ideally, changes that have been applied during the business-to-IT transition should be reflected at the analysis level. One possibility is to recompute the analysis model as an abstract view on the BPEL code. However, this leads to two different process models, the one drawn by the business analyst and the other generated from the IT process, which need to be synchronized again. For a general approach to model synchronization see for example [29]. Another possibility is to use the analysis model as input to a business-driven development process in which transformations are applied until a design model is obtained from which BPEL code can be directly generated [3]. In this paper, we concentrate on this second scenario and investigate how a tool can support a user in this activity.

We assume that the user wants to apply transformations to the analysis model of the business process in order to obtain a process design that reflects the desired BPEL block structure. Ideally, the tool should inform the user that the model contains sequential and parallel branches that end in individual stop nodes. Then, it could either automatically apply transformations to make the model structurally aligned to the future BPEL or guide the user in applying the appropriate transformations. Figure 4 illustrates a first possible model transformation that joins the parallel branches of the process model.

It takes the three stop nodes ending the branches in the parallel process fragment and replaces them by a join followed by a single stop node.

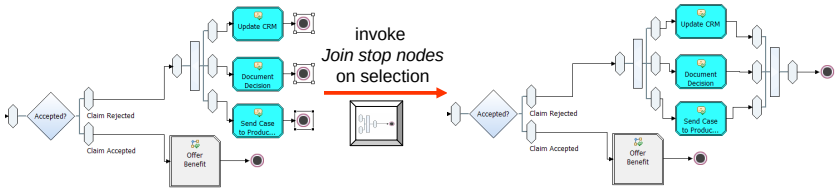


Fig. 4. Joining multiple stop nodes.

In a second transformation, the newly introduced stop node is merged with the two stop nodes ending the exclusive branches, see Fig. 5. A merge is introduced, followed by a single stop node. This yields the desired BPEL block structure, from which also the correct join condition for link-style BPEL code can easily be computed.

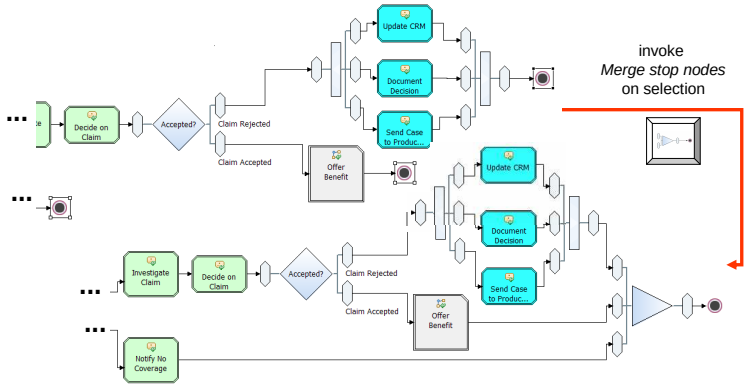


Fig. 5. Merging the remaining stop nodes.

If the user had applied a join to a larger selection of stop nodes, an incorrect process model would result that does not terminate correctly. Ideally, a tool should warn or prevent the user from applying transformations that lead to an incorrect model. In the following section, we take a closer look at the structural analysis methods that we use to ensure that users obtain feedback about the correctness of models resulting from a transformation.

4 Ensuring the Quality of Business Process Models

Business process models were traditionally used mainly for documenting and communicating a business process. As they were used only by humans, lack of quality of a model was tolerable to some extent. Today, with the proliferation of business process management systems, many process models are executed by machines. Errors in those models can incur substantial costs. A faithful and error-free model is also important

when one tries to obtain realistic business measures from a process model through simulation or analysis, which is also supported by many tools today.

Obtaining a faithful and error-free executable model can be a difficult and painful task. Business process models can be quite complex, often comprising a hundred or more activities with complex interactions between various business partners. Applying transformations to such a complex model can easily give rise to additional errors when done manually. It is thus important that a transformation framework can evaluate the quality, in particular, the correctness of a model before a transformation is applied. Furthermore, there should be something like a look-ahead: If applying a transformation to a correct model yields an incorrect model, the user must be alerted of this issue.

Possible errors in business process models include control-flow and data-flow errors. An example of a control-flow error is a *deadlock*, i.e., a situation where some part of the process is waiting indefinitely for another part of the process. A data-flow error occurs, e.g., when a piece of data is not available when needed. Many of these errors can be avoided by applying a rigorous modeling discipline, i.e., by using correct modeling patterns and by avoiding modeling anti-patterns [28].

Control-flow and data-flow errors can also be detected automatically by dedicated analysis algorithms. Detection of deadlocks or a wider class of errors can be done using techniques from the area of model checking which can be applied to business process models [30, 31]. In the worst case, these techniques have to build the entire state space of the process model, the size of which can be exponential in the size of the process model, a problem that is widely known as *state-space explosion*. To mitigate the state-space explosion problem, we use a technique that is known from compiler theory: We decompose the process model into a hierarchy of *single-entry-single-exit (SESE) fragments* [32].

Figure 6 shows a process model and its SESE fragments, which are indicated by dashed boxes. Suppose that this model was derived from the model in Fig. 1 by applying a stop node transformation to the four topmost stop nodes, which were combined by a join, then followed by a second transformation that added a merge. The first of the two transformations introduced a deadlock. For example, if the claim is accepted, the join in fragment F waits in vain for the other three activities in fragment F to finish.

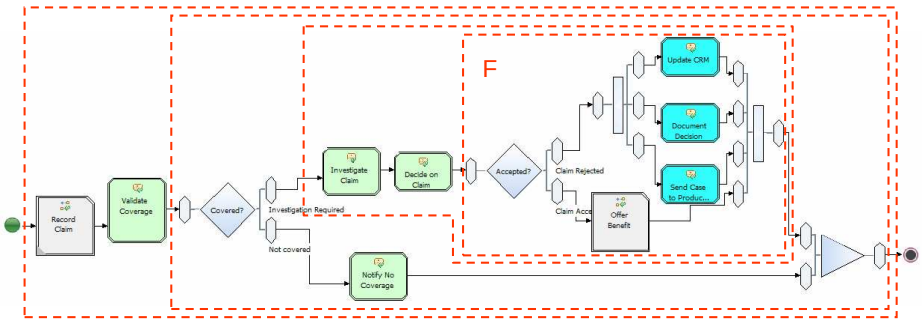


Fig. 6. An erroneous process model and its decomposition into SESE fragments.

To check for control-flow errors in the overall process model, it is sufficient to check each fragment in isolation, i.e., each error is local to some SESE fragment. For example, the deadlock in Fig. 6 is local to fragment F.

A SESE fragment is usually much smaller than the overall process. Its size is measured as the number of edges between its direct subfragments. As the decomposition into SESE fragments can be computed in linear time and there are at most twice as many fragments as there are atomic activities in the process model, the time used for the control-flow analysis of all the fragments mainly depends on the size of the largest fragment in the process. In a case study with more than 340 real-world business process models which had an average size of 75 edges with the maximum being 342 edges, we measured that the largest fragment of a process on average had size 25 with a maximum of 82 [32].

As a second technique to mitigate the state-space explosion problem, we use heuristics that can be applied in linear time to sort out many of the error-free and a fair percentage of the erroneous fragments before any state-space generation is applied [32]. This is based on the observation that many error-free and some erroneous fragments in practice have a simple structure that can easily be recognized. For example, the deadlock in fragment F in Fig. 6 can be detected by recognizing that the fragment includes a decision, but no merge [28, 32].

Modeling errors are reported to the user, who can then take steps to correct the model by manually editing the model or applying automatic transformations. When interleaving the analysis with model transformations, the user can be warned that the selected transformation is not applicable to the set of selected stop nodes without introducing a deadlock into the model. The decomposition into SESE fragments can also be used to speed up an automatic computation of a correct stop-node merging based on model-checking techniques.

5 In-Place Transformation Framework Architecture

IBM WebSphere Business Modeler is built on top of the Eclipse platform, making it relatively straightforward to plug in custom extensions providing advanced functionality. Whereas a detailed discussion concerning the tool's extension points would exceed the scope of this paper, suffice it to say that the tool has been designed using the model-view-controller pattern and that it is possible to manipulate the model elements using the command pattern [33]. Unfortunately, the command pattern does not support easy programmatic access of a model. For every change, a command object has to be set up with the correct parameters, options and references to the model elements to be modified. With this approach, most of the transformation code would be dedicated to setting up commands and pushing them onto the command execution stack, and the logic of the transformation would become very hard to follow.

Thus, an abstraction layer is needed to enable programmatic access to the in-memory model so that it can be modified with minimal amount of coding, but still without breaking the model-view-controller design of the tool. In this way, the results of a transformation become immediately visible to the user, whereas for the developer the elements of a model are exposed in such a way that it becomes easy to implement transformations using an ordinary programming language, i.e., Java in our case. In this

approach, transformations are natively executed because no interpretation is required and the Eclipse infrastructure is reused to package and ship transformation plug-ins as extensions to the product.

The main purpose of our transformation framework is to provide such an abstraction layer. It supports the execution of automatic refactoring, refinement and abstraction transformations, and enables their full integration with the existing modeling environment and the quality-assurance functionality. As shown in Fig. 7, the transformation framework extends the IBM WebSphere Business Modeler environment, acting as a container of plug-ins that package the actual transformation code so that the modeling tool can be customized by activating and deactivating the appropriate transformation plug-ins.

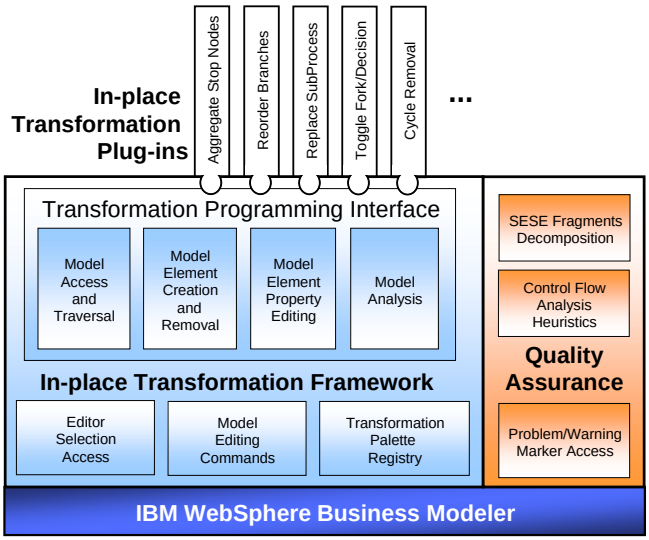


Fig. 7. Architecture of the transformation framework.

The challenge of this approach lies in the design of the "transformation programming interface" (TPI) visible to the developer. It is especially important to add methods to the TPI that make the model efficiently accessible so that it can be traversed, analyzed, and modified by the transformation code.

Table 1 summarizes the main features of the TPI that help in the rapid development of new transformations. Transformations may use the interface to edit models by *creating new elements* and *removing existing ones*. Element *properties can be directly modified*, e.g., to rename an element or to reposition an element in the diagram. Furthermore, the programming interface has been designed to support *different patterns of model traversal*. *Simple transformations* are independently applied once to each target model element and thus do not require the transformation code to deal with model traversal issues. *Complex transformations* may require to filter the elements of a model based on some criteria. In the simplest case, the filter checks the meta-model element type, for example to distinguish stop nodes from start nodes. However, also non-trivial

conditions may be required, such as checking whether elements are connected or belong to a SESE fragment. In general, transformations may *traverse* model elements in some specific order, for example, by drilling down the element containment structure or by navigating through the predecessor/successors elements as linked by the control flow. To support complex transformations that do not scan elements in a predefined order, the framework offers a direct look-up of elements. Finally, transformations can be registered with a palette or menu of macro-editing buttons displayed to the user, see also Section 6.

Table 1. Excerpt of the Transformation Programming Interface.

TPI Feature	**Example**
Creation of new model elements	`addStopNode()`
	`addStartNode()`
	`addTask()`
	`addGateway(Type)`
	`addControlEdge()`
	`addDataEdge(Type)`
Removal of existing model elements	`remove(Element)`
Editing of model element properties	`move(Position)`
	`rename(String)`
Random access to model elements	`find(ElementID)`
Access to selected model elements	`selection.getEdges()`
	`selection.getNodes()`
	`selection.getStopNodes()`
Traversal of related model elements	`getInBranch()`
	`getOutBranch()`
	`getPredecessor()`
	`getSuccessor()`
	`getParent()`
	`getChildren()`
Analysis of model elements	`isDisconnected()`
m	`isSESE(Fragment)`
Transformation palette registration	`register(Transformation)`
	`unregister(Transformation)`

To illustrate how the TPI can be used, we show below how to implement the “stop node aggregation” transformation mentioned in Section 1.

```
transformation aggregateSelectedStopNodes(gatewayType) (
    predecessors = [];
    nodes = TPI.selection.getStopNodes();
    if (nodes.length > 1) (
      foreach (node in nodes) (
        predecessors.append(TPI.getPredecessor(node));
        TPI.remove(node);
      )
      gateway = TPI.addGateway(gatewayType, predecessors.length);
      stopNode = TPI.addStopNode();
      TPI.addControlEdge(TPI.getOutBranch(gateway,0), stopNode);
      i = 0;
      foreach (pred in predecessors) (
        TPI.addControlEdge(pred, TPI.getInBranch(gateway,i));
        i++;
      )))
```

This transformation is applied to a set of selected stop nodes and replaces them with a join or merge depending on the user's input, recall Figs. 4 and 5. As shown in the pseudo-code, the transformation first ensures that more than one stop node has been selected. As additional precondition, the transformation could check whether aggregating the selected nodes would not introduce an error, see the discussion in Section 4. Then, the transformation iterates over all selected stop nodes, stores their predecessor element for later use, and subsequently deletes the stop node. Then it adds either a join or a merge to the model and links its outgoing branch with a new stop node. As a last step, it connects each predecessor element to a different incoming branch of the newly added join or merge.

6 Palette-Based Invocation of Transformations

Transformations can be made available to users through a menu or palette. One can imagine that palettes are provided to users with transformations supporting certain development methodologies or industry-specific requirements. Figure 8 shows a possible design of such a palette-based user interface. Users can invoke transformations via a menu or by clicking on the palette button showing a mnemonic picture of the transformation. If no model elements are selected prior to invocation, a transformation is applied to the entire model. An "undo" capability can easily be provided to the user, because transformations are executed as sequences of editor commands. The history of transformed models could be maintained by using version management enhanced with traceability at the model-element level.

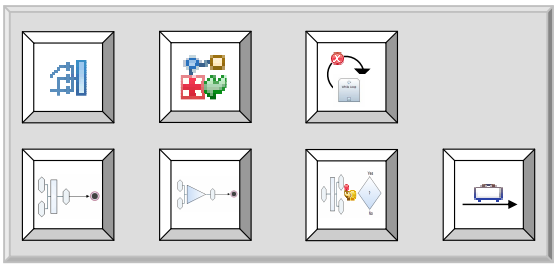


Fig. 8. A palette of model transformations.

The palette above shows some of the model transformations that we implemented. Most of these transformations can exist in a simple form without linking to quality assurance and in a more sophisticated form that links to quality assurance to support the user in correctly applying a transformation. In the upper row of the palette, we find (from left to right) the transformations *automatically reorder branches*, *replace subprocess*, and *cycle removal*. In the lower row, we find the transformations *join stop nodes*, *merge stop nodes*, *toggle fork/decision*, and *assign data container*. In addition to these transformations, many others can be imagined.

Automatically reorder branches is a horizontal, non-destructive, semantics-preserving transformation that simply cleans up clutter in the diagram, which can occur

when branches are connected to a join or merge. The transformation analyzes the graphical layout and eliminates crossing branches.

Replace subprocess is a horizontal, destructive transformation that replaces a user-selected subprocess by another user-selected subprocess. It prompts the user to select the replacing subprocess from a list of subprocesses that the transformation obtains from the workspace. In the current implementation, this transformation connects the new subprocess only with control-flow edges.

Cycle removal is a vertical, destructive, semantics-preserving transformation that takes a process model with unstructured cycles, i.e., backward edges added to the flow, and produces a model with well-structured loops [34]. The transformation leads to a model with a more technical flavor for many business users—therefore, we consider it as a vertical transformation. Cycle removal relies on the SESE analysis described in Section 4. It can happen that it returns an only partially transformed model. In particular, cycles that spawn parallel branches often cannot be removed.

Join stop nodes and *Merge stop nodes* are horizontal and destructive transformations already known to the reader. While Merge stop nodes is semantics-preserving, Join stop nodes is not due to the semantics of these modeling elements. The two transformations are implemented, but do not link to the quality assurance yet. Hence, it is under the full responsibility of the user whether to apply the transformation.

Toggle fork/decision is a horizontal, destructive transformation that simply flips a selected fork into a decision and vice versa. This version is useful during the editing process, e.g., when correcting modeling errors. However, it can easily introduce control-flow errors, as discussed in Section 4. A more sophisticated version would transform process fragments of sequential branching behavior into fragments of parallel behavior and vice versa, which requires a combination with quality assurance.

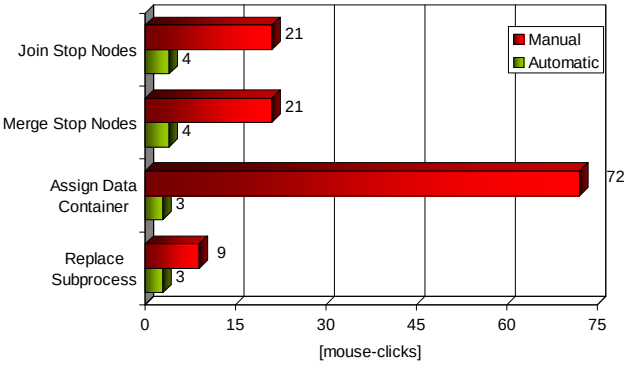
A very interesting challenge is the treatment of data flow in transformations. It can be studied in the *Assign data container* transformation, which is a vertical, destructive transformation that takes a model with control flow and refines it into a model with data flow. It can also be applied to models with mixed control and data flow. The transformation leads to a changed interface of model elements.

Several possible solutions exist for how a transformation can modify the interfaces of activities, e.g., it can add only the newly required inputs/outputs or it can in addition remove those inputs/outputs that are no longer needed. Existing data-flow edges can be restored if the old and the new interface of a model element share the inputs and outputs that are required by the data flow. Otherwise, data maps have to be inserted, which will remain abstract in most cases, because the transformation cannot determine what the exact mapping between mismatched data will be. These interface changes usually affect the consistency of other process models that share the same model elements. The resolution of possible inconsistencies is a challenging problem, which may not be amenable to a fully automatic solution and require other transformations to support the user. In addition, beautifier transformations relying on quality assurance may be required to eliminate control and data flow edges that are no longer needed in the transformed models.

At the moment of writing, it is too early to give a comprehensive evaluation of the framework itself. Concerning the performance of the transformations, following an in-

place approach has shown its benefits in terms of the speed at which transformations are executed. Users running transformations hardly notice the difference between transformations and normal editing commands, because they see the result of the transformation immediately without having the need to persist the transformed models.

In terms of usability, the transformations are easy to apply and significantly reduce the editing effort for the user. Based on the example scenario in this paper, Fig. 9 shows that model transformations reduce lengthy and error-prone manual editing operations to a few clicks. For example, manually performing the join and merge stop nodes transformations in the example scenario takes 42 mouse clicks. Automating the transformation still requires the user to select the set of nodes (twice three clicks), but then the model is updated with a single mouse click. The chart in Fig. 9 shows two more transformations, *assign data container* and *replace subprocess*, in the context of the example scenario.



	Manual	Automatic
Join	$6s+3$	$s+1$
Merge	$6s+3$	$s+1$
Assign	$8e$	3
Replace	$2l+5$	3

s ... No. of stop nodes
e ... No. of edges
l ... No. of links connected to the process

Fig. 9. Usability evaluation of selected in-place model transformations.

7 Conclusion

Model transformations help significantly in addressing challenges in the business-IT gap encountered during business-driven development, which aims at directly taking business process models to the IT level. In this paper, we report on a transformation framework that adds a lightweight infrastructure to IBM WebSphere Business Modeler for the rapid development of model transformations. Using this framework, in-place transformations are developed that are easily applicable by business users to automate complicated editing steps. By linking them to quality-assurance capabilities provided in modeling tools, the transformations can be made "intelligent" and help users to preserve or re-establish the correctness of their models when going through a sequence of refinement and refactoring operations. The set of transformations implemented significantly increases user productivity as they raise the abstraction level of the model editing palette from a "picture-drawing" tool to a level supporting real business-process modeling.

Acknowledgement The work published in this article was partially conducted within the EU project Super (www.ip-super.org) under the EU 6th Framework.

References

1. Newcomer, E., Lomow, G.: Understanding SOA with Web Services. Addison Wesley (2005)
2. Mitra, T.: Business-driven development. IBM developerWorks article, http://www.ibm.com/developerworks/webservices/library/ws-bdd, IBM (2005)
3. Koehler, J., Hauser, R., Küster, J., Ryndina, K., Vanhatalo, J., Wahler, M.: The role of visual modeling and model transformations in business-driven development. In: Proceedings of the 5th International Workshop on Graph Transformation and Visual Modeling Techniques, Elsevier (2006) 1–12
4. Brahe, S., Bordbar, B.: A Pattern-based Approach to Business Process Modeling and Implementation in Web Services. In: In Proceedings of Workshop Modeling the SOA - Business perspective and model mapping, in conjunction with ICSOC. (2006)
5. Christensen, E., Curbera, F., Meredith, G., Weerawarana, S.: Web services description language (WSDL). http://www.w3.org/TR/wsdl (2001)
6. Jordan, D., et al.: Web services business process execution language (WSBPEL) 2.0. htpp://www.oasis-open.org/committees/wsbpel/ (2007)
7. IBM: WebSphere Business Modeler. http://www.ibm.com/software/integration/wbimodeler
8. Object Management Group: Model driven architecture (2001) http://www.omg.org/mda.
9. Mens, T., van Gorp, P., Karsai, G., Varró, D.: Applying a model transformation taxonomy to graph transformation technology. In Karsai, G., Taentzer, G., eds.: GraMot 2005, International Workshop on Graph and Model Transformations. Volume 152 of ENTCS., Elsevier (2006) 143–159
10. Mens, T., Gorp, P.V.: A Taxonomy of Model Transformation. Electr. Notes Theor. Comput. Sci. **152** (2006) 125–142
11. Czarnecki, K., Helsen, S.: Feature-based survey of model transformation approaches. IBM Systems Journal, special issue on Model-Driven Software Development **45**(3) (2006) 621–645
12. Biermann, E., Ehrig, K., Köhler, C., Kuhns, G., Taentzer, G., Weiss, E.: Graphical Definition of In-Place Transformations in the Eclipse Modeling Framework. In Nierstrasz, O., Whittle, J., Harel, D., Reggio, G., eds.: Model Driven Engineering Languages and Systems, 9th International Conference, MoDELS 2006. Volume 4199 of LNCS., Springer (2006) 425–439
13. Ermel, C., Rudolf, M., Taentzer, G.: The AGG-Approach: Language and Tool Environment. In Ehrig, H., Engels, G., Kreowski, H.J., Rozenberg, G., eds.: Handbook of Graph Grammars and Computing by Graph Transformation, volume 2: Applications, Languages and Tools, World Scientific (1999) 551–603
14. Jouault, F., Kurtev, I.: Transforming Models with ATL. In Bruel, J.M., ed.: Satellite Events at the MoDELS 2005 Conference, Revised Selected Papers. Volume 3844 of LNCS., Springer (2005) 128–138
15. Mens, T.: On the use of graph transformations for model refactoring. In: 2005 Summer School on Generative and Transformational Techniques in Software Engineering, Braga, Portugal, Departamento Informatica, Universidade do Minho, Braga, Portugal, Technical Report TR-CCTC/DI-35 (2005) 67–98
16. Stürmer, I., Kreuz, I., Schäfer, W., Schürr, A.: Enhanced simulink/stateflow model transformation: The mate approach. In: Proceedings of MathWorks Automotive Conference (MAC 2007), MathWorks (2007)
17. Balogh, A., Németh, A., Schmidt, A., Rath, I., Vágó, D., Varró, D., Pataricza, A.: The VIATRA2 model transformation framework (2005) Presented at ECMDA 2005 – Tools Track.
18. Karsai, G., Agrawal, A., Shi, F., Sprinkle, J.: On the Use of Graph Transformation in the Formal Specification of Model Interpreters. Journal of Universal Computer Science **9**(11) (2003) 1296–1321

19. de Lara, J., Vangheluwe, H.: $AToM^3$: A Tool for Multi-Formalism and Meta-Modelling. In Kutsche, R.D., Weber, H., eds.: Proceedings Fundamental Approaches to Software Engineering (FASE 2002). Volume 2306 of LNCS., Springer-Verlag (April 2002) 174–188
20. Braun, P., Marschall, F.: BOTL - The Bidirectional Objekt Oriented Transformation Language. Technical report, Fakultät für Informatik, Technische Universität München, Technical Report TUM-I0307 (2003)
21. Nickel, U., Niere, J., Zündorf, A.: Tool demonstration: The FUJABA environment. In: Proceedings of the 22^{nd} International Conference on Software Engineering (ICSE), Limerick, Ireland, ACM Press (2000) 742–745
22. Akehurst, D.H., Bordbar, B., Evans, M.J., Howells, W.G.J., McDonald-Maier, K.D.: SiTra: Simple Transformations in Java. In Nierstrasz, O., Whittle, J., Harel, D., Reggio, G., eds.: Model Driven Engineering Languages and Systems, 9th International Conference, MoDELS 2006. Volume 4199 of LNCS., Springer (2006) 351–364
23. Object Management Group (OMG): Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification. Final Adopted Specification ptc/05-11-01. (November 2005)
24. Varró, D.: Model Transformation by Example. In Nierstrasz, O., Whittle, J., Harel, D., Reggio, G., eds.: Model Driven Engineering Languages and Systems, 9th International Conference, MoDELS 2006. Volume 4199 of LNCS., Springer (2006) 410–424
25. Küster, J.M.: Definition and validation of model transformations. Software and Systems Modeling (SoSyM) **5**(3) (2006) 233–259
26. Varró, D., Varró-Gyapay, S., Ehrig, H., Prange, U., Taentzer, G.: Termination Analysis of Model Transformations by Petri Nets. In Corradini, A., Ehrig, H., Montanari, U., Ribeiro, L., Rozenberg, G., eds.: Graph Transformations, Third International Conference. Volume 4178 of LNCS., Springer (2006) 260–274
27. Küster, J.M., Abd-El-Razik, M.: Validation of Model Transformations - First Experiences Using a White Box Approach. In: MoDELS Workshops. Volume 4364 of LNCS., Springer (2006) 193–204
28. Koehler, J., Vanhatalo, J.: Process anti-patterns: How to avoid the common traps of business process modeling, part 1 modeling control flow, part 2 modeling data flow. IBM WebSphere Developer Technical Journal **10.2, 10.4** (2007)
29. Giese, H., Wagner, R.: Incremental Model Synchronization with Triple Graph Grammars. In Nierstrasz, O., Whittle, J., Harel, D., Reggio, G., eds.: Model Driven Engineering Languages and Systems, 9th International Conference, MoDELS 2006. Volume 4199 of LNCS., Springer (2006) 543–557
30. van der Aalst, W.M.P.: Workflow verification: Finding control-flow errors using Petri-net-based techniques. In: Business Process Management, Models, Techniques, and Empirical Studies, London, UK, Springer-Verlag (2000) 161–183
31. Mendling, J., Moser, M., Neumann, G., Verbeek, H.M.W., van Dongen, B.F., van der Aalst, W.M.P.: Faulty EPCs in the SAP reference model. In: Proceedings of the 4th International Conference Business Process Management (BPM 2006). Volume 4102 of LNCS., Springer (2006) 451–457
32. Vanhatalo, J., Völzer, H., Leymann, F.: Faster and More Focused Control-Flow Analysis for Business Process Models though SESE Decomposition. In: 5th International Conference on Service-Oriented Computing (ICSOC), Vienna, Austria (September 2007) to appear.
33. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns: Elements of Reusable Object-Oriented Software. Addison Wesley (1994)
34. Koehler, J., Hauser, R., Sendall, S., Wahler, M.: Declarative techniques for model-driven business process integration. IBM Systems Journal **44**(1) (2005) 47–65