Rule Execution in Graph-Based Incremental Interactive Integration Tools

Simon M. Becker, Sebastian Lohmann, and Bernhard Westfechtel

Department of Computer Science III, RWTH Aachen University
Ahornstraße 55, D-52074 Aachen, Germany
{sbecker,slohmann,bernhard}@i3.informatik.rwth-aachen.de

Abstract. Development processes in engineering disciplines are inherently complex. Throughout the development process, different kinds of inter-dependent design documents are created which have to be kept consistent with each other. Graph transformations are well suited for modeling the operations provided for maintaining inter-document consistency. In this paper, we describe a novel approach to rule execution for graph-based integration tools operating interactively and incrementally. Rather than executing a rule in atomic way, we break rule execution up into multiple phases. In this way, the user of an integration tool may be informed about all potential rule applications and their mutual conflicts so that he may take a judicious decision how to proceed.

1 Introduction

Development processes in engineering disciplines are inherently complex. Throughout the development process, different kinds of inter-dependent *documents* are created which have to be kept consistent with each other. For example, in software engineering there are requirements definitions, software architectures, module bodies, etc. which describe a software system from different perspectives and at different levels of abstraction and granularity. Documents are connected by manifold dependencies and need to be kept consistent with each other. For example, the source code of a software system must match its high-level description in the software architecture.

Development processes may be viewed as multi-stage transformation processes from the initial problem statement to the final solution. Throughout the transformation process, many interacting decisions have to be performed. These decisions can be automated only to a limited extent; in many settings, human *interactions* are required. Moreover, transformation rarely proceeds stage-wise according to some waterfall model. Rather, *incremental* and iterative processes have been proposed, which require to propagate changes throughout a set of inter-dependent documents.

In such a setting, there is a need for incremental and interactive *integration tools* for supporting inter-document consistency maintenance. An integration tool has to manage *links* between parts — called *increments* in the sequel — of inter-dependent documents. It assists the user in *browsing* (traversing the links in order to navigate between related increments in different documents), *consistency analysis* (concerning the relationships between the documents' contents), and *transformations* (of the increments contained in one document into corresponding increments of the related document).

Graphs and graph transformations have been used successfully for the specification and realization of integration tools [1, 2]. However, in the case of incremental and interactive integration tools specific requirements have to be met concerning the execution of integration rules. In this paper, we describe a novel approach to rule execution for graph-based integration tools operating incrementally and interactively. We have realized this approach, which is based on triple graph grammars [3, 4], in a research prototype called *IREEN*, an *I*ntegration *R*ule *E*valuation *EN*vinronment [5]. Rather than executing a rule in atomic way, IREEN breaks rule execution up into multiple phases. In this way, the user of an integration tool may be informed about all potential rule applications and their mutual conflicts so that (s)he may take a judicious decision how to proceed.

The rest of this paper is structured as follows: Section 2 presents a scenario which motivates our work by a practical example. Section 3 is devoted to the graph-based specification of integration tools. Section 4, the core part of this paper, presents our novel approach to rule execution. Section 5 discusses related work, and Section 6 presents a short conclusion.

2 Scenario

The research reported in this paper is carried out within the IMPROVE project [6], which is concerned with models and tools for design processes in *chemical engineering*. In this section, we present a small example which illustrates key features of incremental and interactive integration tools. This example is drawn from chemical engineering, but we could also have chosen an example from another engineering discipline (e.g., software engineering).

In chemical engineering, the *flow sheet* acts as a central document for describing the chemical process. The flow sheet is refined iteratively so that it eventually describes the chemical plant to be built. Simulations are performed in order to evaluate design alternatives. Simulation results are fed back to the flow sheet designer, who annotates the flow sheet with flow rates, temperatures, pressures, etc. Thus, information is propagated back and forth between flow sheets and *simulation models*. Although the flow sheet plays the role of a master document, it may also happen that a simulation model is created first and the flow sheet is derived from the simulation model (reverse engineering).

Unfortunately, the relationships between flow sheets and simulation models are not always straightforward. Different kinds of simulation models are created for different purposes. Often, simulation models have to be composed from pre-defined blocks which in general need not correspond 1:1 to structural elements of the flow sheet. Thus, maintaining consistency between flow sheets and simulation models is a demanding task requiring sophisticated tool support.

Figure 1 illustrates how an incremental integration tool assists in maintaining consistency between flow sheets and simulation models. In general, flow sheets and simulation models are created by different users at different times with the help of respective tools; an integration tool is used to establish mutual consistency on demand. In a cooperation with an industrial partner, we studied the coupling of COMOS [7], an environment

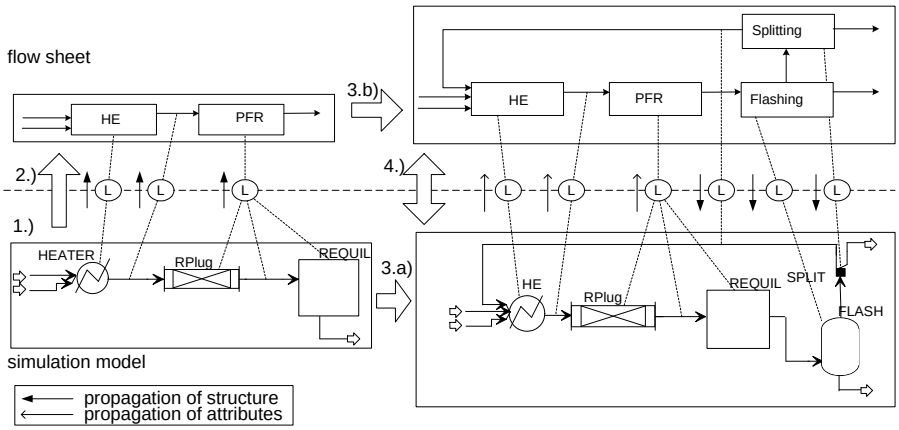

Fig. 1. Integration between flow sheet and simulation model

for chemical engineering which in particular offers a flow sheet editor, and Aspen Plus [8], an environment for performing steady-state and dynamic simulations.

The chemical process taken as example produces ethanol from ethen and water. Flow sheet and simulation model are shown above and below the dashed line, respectively. The *integration document* for connecting them contains links which are drawn on the dashed line. The figure illustrates a design process consisting of four steps:

1. The simulation expert has already created a simulation model for a part of the chemical process (heating and reaction). The simulation model is composed of three blocks according to the capabilities of the respective simulation tool.
2. The simulation model is transformed into a flow sheet. This is achieved with the help of an integration tool. Multiple alternatives are available for this transformation. It turns out that the simplest one — a 1:1 transformation — does not result in an adequate flow sheet because the blocks do not correspond 1:1 to devices in the flow sheet. Rather, the user decides to group two blocks and their connecting stream into a single device (a plug flow reactor) in the flow sheet. The link between the PFR and the respective parts of the simulation model is established by firing a corresponding integration rule. In addition, another rule is available which just transforms the block called RPlug into a PFR. This 1:1 rule stands in conflict with the rule selected here. The integration tool presents conflicting rules to the user who may select the rule to be applied.
3. Steps 3a and 3b are carried out in parallel, using different tools. Using the simulation model created so far, a simulation is performed in the simulation tool. The simulation results comprise flow rates, temperatures, etc. In parallel, a flow sheet editor is used to extend the flow sheet with the chemical process steps that have not been specified so far (flashing and splitting).
4. Finally, the integration tool is used to synchronize the parallel work performed in the previous step. This involves information flow in both directions. First, the simulation results are propagated from the simulation model back to the flow sheet. Sec-

ond, the extensions are propagated from the flow sheet to the simulation model. After these propagations have been performed, mutual consistency is re-established.

From this example, we may derive several features of the kinds of integration tools that we are addressing. Concerning the mode of operation, our focus lies on incremental integration tools rather than on tools which operate in a batch-wise fashion. Rather than transforming documents as a whole, incremental changes are propagated — in general in both directions — between inter-dependent documents. Often, the integration tool cannot operate automatically; rather, the user has to perform decisions interactively. In general, the user also maintains control on the time of activation, i.e., the integration tool is invoked to re-establish consistency whenever appropriate. Finally, it should be noted that integration tools do not merely support transformations. In addition, they are used for analyzing inter-document consistency or browsing along the links between inter-dependent documents.

3 Graph-Based Specification of Integration Tools

In complex scenarios as described in the previous section, an integration tool needs to maintain a data structure storing links between inter-dependent documents. This data structure has been called integration document. Altogether, there are three documents involved: the *source document*, the *target document*, and the *integration document*. Please note that the terms "source" and "target" denote distinct ends of the integration relationship between the documents, but it does not necessarily imply a unique direction of transformation (in fact, transformations are performed in both directions in our sample scenario).

All involved documents may be modeled as graphs, which are called *source graph*, *target graph*, and *correspondence graph*, respectively[1]. Moreover, the operations performed by the respective tools may be modeled by graph transformations. *Triple graph grammars* [3] were developed for the high-level specification of graph-based integration tools. The core idea behind triple graph grammars is to specify the relationships between source, target, and correspondence graphs by *triple rules*. A triple rule defines a coupling of three rules operating on source, target, and correspondence graph, respectively. By applying triple rules, we may modify coupled graphs synchronously, taking their mutual relationships into account.

An example of a triple rule is given in Figure 2 in PROGRES [9] syntax. The rule refers to the running example to be used throughout the rest of this paper, namely the creation of *connections* (appearing in both flow sheets and simulation models). In a flow sheet, a connection is used to relate structural elements such as *devices* and *streams*. An example of a device is a reactor, a stream is used to represent the flow of chemical substances between devices. In Figure 1, devices are represented as rectangles, streams are shown as directed lines. Connections are not represented explicitly (rather, they may be derived from the layout), but they are part of the internal data model. Each device or stream has a set of ports; connections establish relationships between these ports.

[1] If the tools operating on source and target document are not graph-based, the integration tool requires *wrappers* which establish corresponding graph views.

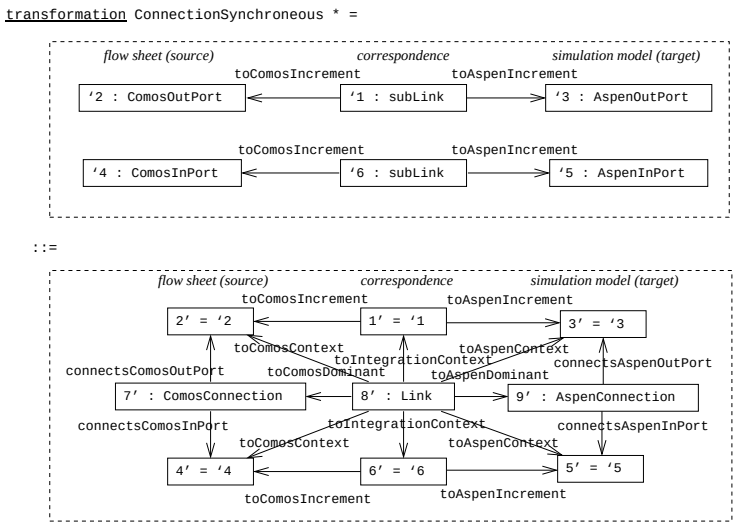

Fig. 2. Triple rule for a connection

The triple rule `ConnectionSynchronous` has a *left-hand side* (shown above the right-hand side) which spans all participating subgraphs: the source graph (representing the flow sheet) on the left, the correspondence graph in the middle, and the target graph (for the simulation model) on the right. The left-hand side is composed of port nodes in source and target graph, distinguishing between output ports and input ports[2]. Furthermore, it is required that the port nodes in both graphs correspond to each other. This requirement is expressed by the nodes of type `subLink` in the correspondence graph and their outgoing edges which point to nodes of the source and target graph, respectively. Port correspondences are established by other triple rules which transform the blocks the ports belong to, e.g. streams or devices. Correspondences between source and target patterns are represented by *links* and can be further structured by *sublinks*, e.g. to express port correspondences.

All elements of the left-hand side re-appear on the *right-hand side*. New nodes are created for the connections in source and target graph, respectively, as well as for the link between them in the correspondence graph. The connection nodes are embedded locally by edges to the respective port nodes. For the link node, three types of adjacent edges are distinguished. `toDominant` edges are used to connect the link to exactly one *dominant increment* in the source and target graph, respectively. In general, the source and target pattern related through the triple rule may consist of more than one increment in each participating graph. Then, there are additional edges to *normal increments* (not needed in our running example)[3]. Finally, `toContext` edges point to nodes which are

[2] Only ports of different orientation may be connected.

[3] The distinction between dominant and normal increments is not vital, but helpful for pragmatic reasons; see next section.

not themselves part of the transformation but are required as a context condition. These nodes are called *context increments*.

Figure 2 describes a *synchronous graph transformation*. As already explained earlier, we cannot assume in general that all participating documents may be modified synchronously. In case of asynchronous modifications, the triple rule shown above is not ready for use. However, we may derive *asynchronous rules* from the synchronous rule in the following ways:

- A *forward rule* assumes that the source graph has been extended, and extends the correspondence graph and the target graph accordingly. Thus, the forward rule derived from our sample rule would contain node 7 on the left-hand side.
- Analogously, a *backward rule* is used to describe a transformation in the reverse direction. In our example, node 9 would be part of the left-hand side.
- Finally, a *consistency analysis* rule is used when both documents have been modified in parallel. In our running example, this means that connections have been inserted into both the flow sheet and the simulation model and a link is created a posteriori. Thus, the consistency analysis rule would include nodes 7 and 9 on the left-hand side.

Unfortunately, even these rules are not ready for use in an integration tool as described in the previous section. In the case of non-deterministic transformations between inter-dependent documents, it is crucial that the user is made aware of conflicts between applicable rules. Thus, we have to consider all applicable rules and their mutual conflicts before selecting a rule for execution. To this achieve this, we have to give up *atomic rule execution*, i.e., we have to decouple pattern matching from graph transformation.

4 Rule Execution

4.1 Overview

As explained in the previous section, an integration rule cannot be executed by means of a single graph transformation. To ensure the correct sequence of rule executions, to detect all conflicts between rule applications, and to allow the user to resolve conflicts, each integration rule is automatically translated to a set of graph transformations. These rule specific transformations are executed together with some generic ones following an *integration algorithm*. In this subsection, we will present the overall algorithm, while in the following subsections the phases of the algorithm are explained in detail, showing some of the rule specific and generic graph transformations involved. The simplified example in Figure 4 (to be explained later) is used to illustrate the algorithm.

While the algorithm in general supports the concurrent execution of forward, backward, and consistency analysis rules, we focus on forward transformations only, using the forward transformation rule for a connection as running example. Some aspects of the algorithm are omitted, as the treatment of existing links that have become inconsistent due to modifications in the integrated documents.

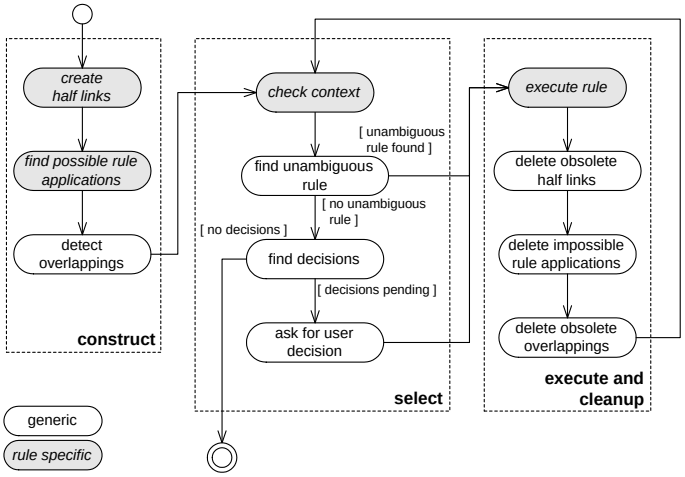

Fig. 3. Integration algorithm

Figure 3 shows a UML activity diagram depicting the integration algorithm. To perform each activity, one ore more graph transformations are executed. Activities that require the execution of rule specific transformations are marked grey and italic. The overall algorithm is divided into three phases.

During the first phase (construct), all possible rule applications and conflicts between them are determined and stored in the graph. First, for each increment in the source document that has a type compatible to the dominant increment's type of any rule, a half link is created that references this increment. Then, for each half link the possible rule applications are determined. The last step of this phase is a generic transformation marking overlappings between possible rule applications.

In the next phase (select), for all rule applications the context is checked. If one rule application, whose context is present, is unambiguous, it is automatically selected for execution. Otherwise, the user is asked to select one rule among the rules with existing context. If there are no executable rules, the algorithm ends.

In the last phase (execute and cleanup), the selected rule is executed and some operations are performed to adapt the information that was collected in the construct phase to the new situation.

4.2 Construction Phase

In the construction phase, it is determined which rules can be possibly applied to which subgraphs in the source document. Conflicts between these rules are marked. This information is collected once in this phase and is updated later incrementally during the repeated executions of the other phases.

In the first step of the construction phase (create half links), for each increment, the type of which is the type of a *dominant increment* of at least one rule, a link is created that references only this increment (*half link*). Dominant increments are used as

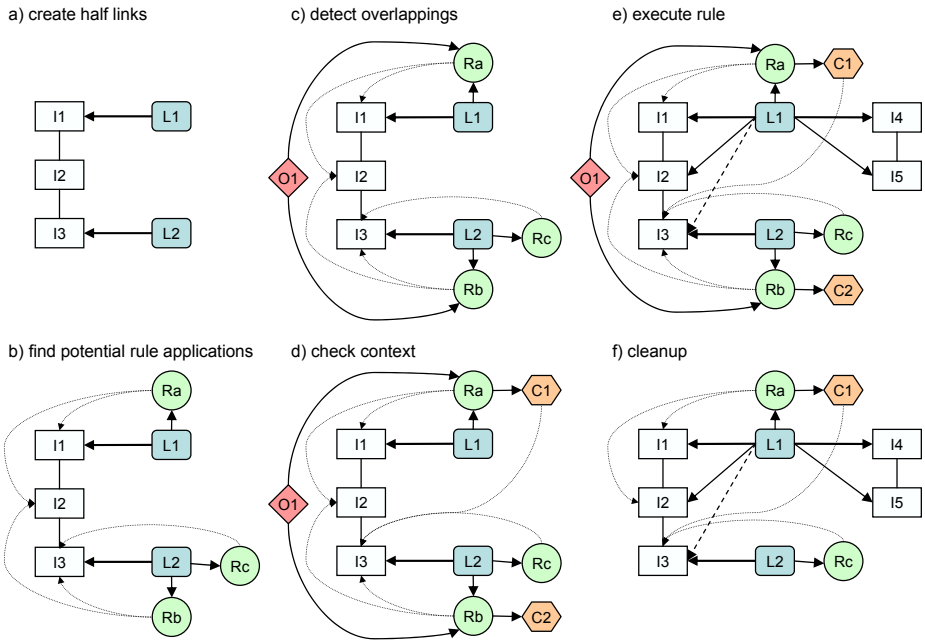

Fig. 4. Simplified example integration

anchor for links and to group decisions for user interaction. Half links are the anchors for information about possible rule applications and are transformed to consistent links after one of the rules has been applied.

In the example, half links are created for the increments I1and I3, named L1 and L2, respectively (c.f. Figure 4 a).

To achieve this, for each rule a PROGRES production is derived that matches an increment with the same type as the rule's dominant increment in its left-hand side, with the negative application condition that there is no half link attached to the increment, yet. Then on its right-hand side the half link node is created and connected to the increment with an edge. All these productions are executed repeatedly, until no more left-hand sides are matched, i.e., half links have been created for all possibly dominant increments.

The second step (find possible rule applications) determines the integration rules that are possibly applicable for each half link. A rule is *possibly applicable* for a given half link if the source document part of the left-hand side of the synchronous rule without the context increments is matched in the source document graph. The dominant increment of the rule has to be matched to the one belonging to the half link. For the possible applicability, context increments are not taken into account because missing context increments could be created later by the execution of other integration rules. For this reason, the context increments are matched in the selection phase before selecting a rule for execution.

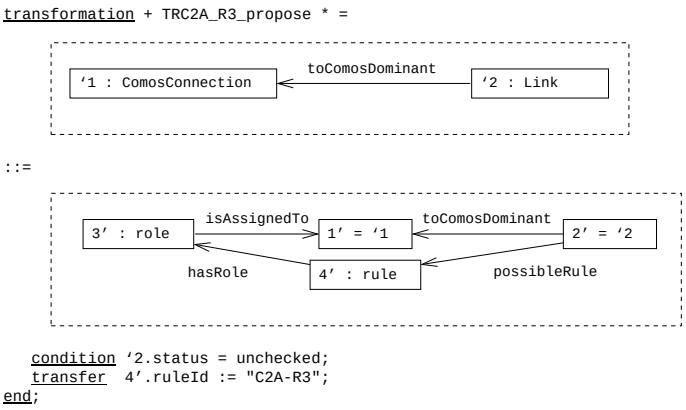

Fig. 5. Find possible rule applications

Figure 5 shows the PROGRES transformation for the example rule. The left-hand side consists of the half link and the respective dominant increment only because all other increments of this rule are context increments. In general, all non-context increments and their connecting edges are part of the left-hand side. On the right-hand side, a rule node is created to identify the possible rule application (4'). This node carries the id of the rule and is connected to the half link. A role node is inserted to explicitly store the result of the pattern matching (3'). If there are more increments matched, role nodes can be distinguished by an id attribute. The asterisk (*) behind the production name tells PROGRES to apply this production for each possible matching of its left-hand side. When executed together with the corresponding productions for the other rules, as a result all possibly applicable rules are stored at each half link. Please note that if a rule is applicable for a half link with different matchings of its source increments, multiple rule nodes with the corresponding role nodes are added to the half link.

In the simplified example (Figure 4 b), three possible rule applications were found, e.g., Ra at the link L1 would transform the increments I1 and I2. Please note that the role nodes are omitted in the figure.

Each increment can be referenced by one link only as non-context increment. This leads to the fact that there can be *conflicts* between possible applications of integration rules. In the case of a conflict, the user has to choose one of the conflicting rules in the selection phase. There are two types of conflicts: First, there can be multiple rule nodes at one half link. These share at least the dominant increment, so only one of the corresponding rules can be executed. This is the case for link L2 in the example in Figure 4 c): Rb and Rc are conflicting. Second, an increment can be referenced by role nodes belonging to rule applications of different links. In the example, the increment I2 is referenced by Ra and Rb.

The conflicts of the first type can be easily determined by counting the rule nodes belonging to a link. The conflicts of the second type are less obvious, so to prepare the user interaction in the selection phase, all of them have to be found and marked. This is done with the help of the generic PROGRES production in Figure 6. The pattern on the left-hand side describes an increment ('7) that is referenced by two roles

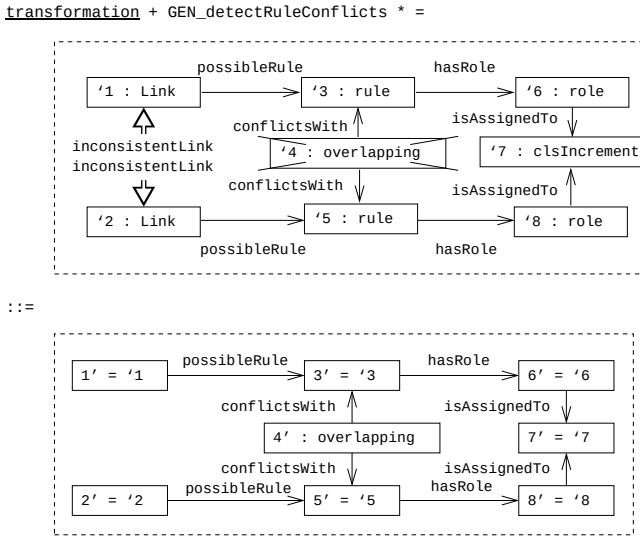

Fig. 6. Detect overlappings

belonging to different rule nodes which belong to different links. The negative node '4 prevents multiple markings of the same conflict. On the right-hand side, an overlap node is inserted between the two rule nodes (O1 in the example). Again, this production is marked with an asterisk, so it is executed until all conflicts are detected. Besides detecting conflicts between different forward transformation rules, the depicted production also detects conflicts between forward, backward, and correspondency analysis rules generated from the same synchronous rule. As a result of that, it is not necessary to check whether the non-context increments of the right-hand side of the synchronous rule are already present in the target document when determining possible rule applications in the second step of this phase.

In the example in Figure 4 c), the overlap node O1 is created between Ra and Rb because they both reference I1. The conflict between Rb and Rc is not explicitly marked because it can be seen from the fact that they both belong to the same half link.

4.3 Selection Phase

The goal of the selection phase is to select one possible rule application for execution in the next phase. If there is a rule that can be executed without conflicts, the selection is performed automatically, otherwise the user is asked for his decision. Before a rule is selected, the contexts of all rules are checked because only a rule whose context has been found can be executed.

The context check is performed in the first step of this phase. The context is formed by all context elements from the synchronous rule. It may consist of increments of source and target documents and of links contained in the integration document.

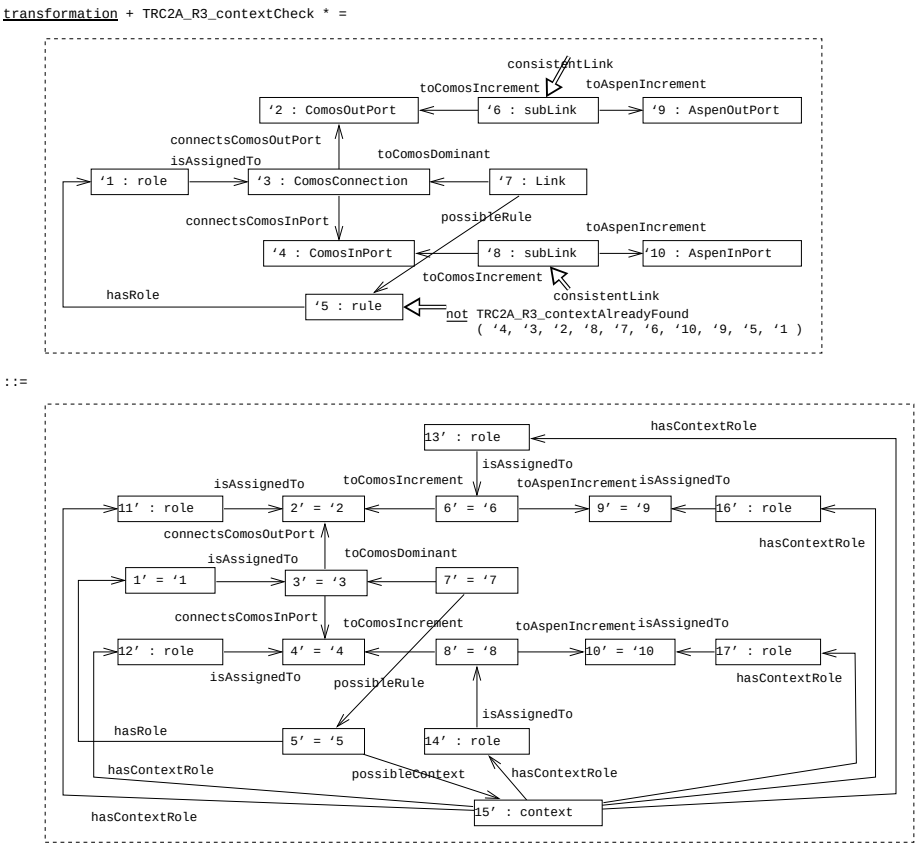

Fig. 7. Check context

In the example in Figure 4 d), the context for Ra consisting of increment I3 in the source document was found (C1). The context for Rb is empty (C2), the context for Rc is still missing.

Figure 7 shows the PROGRES production checking the context of the example integration rule. The left-hand side contains the half link (‘7), the non-context increments (here, only ‘3), the rule node (‘5), and the role nodes (‘1). The non-context increments and their roles are needed to embed the context and to prevent unwanted folding between context and non-context increments. For the example rule, the context consists of the two ports connected in the source document (‘2, ‘4), the related ports in the Aspen document (‘9, ‘10), and the relating sublinks (‘6, ‘8).

On the right-hand side, a new context node is created (‘15). It is connected to all nodes belonging to the context by role nodes (11’, 12’, 13’, 14’, 16’, 17’) and appropriate edges. If the matching of the context is ambiguous, multiple context nodes with their roles are created as the production is executed for all matches.

Because the selection phase is executed repeatedly, it has to be made sure that each context match (context node and role nodes) is added to the graph only once. The context match cannot be included directly as negative nodes on the left-hand side because edges between negative nodes are prohibited in PROGRES. Therefore, this is checked using an additional graph test which is called in the restriction on the rule node. The graph test is not presented here because it is rather similar to the right-hand side of this production.

The context is checked for all possible rule applications. To make sure, that the context belonging to the right rule is checked, the rule id is checked in the condition part of the productions. After the context of a possible rule application has been found, the rule can be applied.

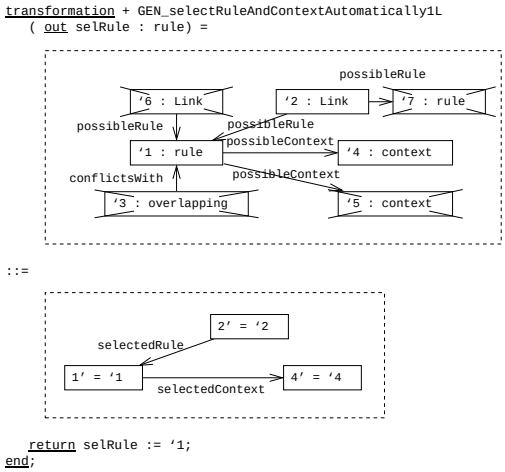

Fig. 8. Select unambiguous rule

After the context has been checked for all possible rule applications, some rules can be applied, others still have to wait for their context. The next step of the algorithm (find unambiguous rule) tries to find a rule application that is not involved in any conflict. The conflicts have already been determined in the construction phase. Because any increment may be referenced by an arbitrary number of links as context, no new conflicts are induced by the context part of the integration rules. The generic PROGRES production in Figure 8 finds rule applications that are not part of a conflict. On the left-hand side a rule node is searched (`'1`) that has only one context node and is not related to any overlapping node. It has to be related to exactly one half link (`'2`) that does not have another rule node. For forward transformation rules, a rule node belongs to one link only, while nodes of consistency analysis rules are referenced by two half links. Therefore for consistency analysis rules, another production is used which is not shown here. A rule node is not selected for execution if there are conflicting rules, even if their context is still missing. As the context may be created later, the user has to decide whether to execute this rule and thereby making the execution of the other rules impossible.

If a match is found in the host graph, the rule node and the context node are selected for execution by substituting their referencing edges by `selectedRule` and `selectedContext` edges, respectively. The rule node is returned in the output parameter `selRule`. The corresponding rule can be applied in the execution phase.

In the example in Figure 4 d), no rule can be automatically selected for execution. The context of Rc is not yet available and Ra and Rb as well as Rb and Rc are conflicting.

If no rule could be selected automatically, the user has to decide which rule has to be executed. Therefore, in the next step (find decisions), all conflicts are collected and presented to the user. For each half link, all possible rule applications are presented. If a rule application conflicts with another rule of a different half link, this is presented as annotation at both half links. Rules that are not executable due to a missing context are included in this presentation but cannot be selected for execution. This information allows the user to select a rule manually, knowing which other rule applications will be made impossible by his decision. If there are no decisions left, the algorithm terminates. If there are still half links left at the end of the algorithm, the user has to perform the rest of the integration manually. If there are decisions, the result of the user interaction is stored in the graph (ask for user decision) and the selected rule is executed in the execution phase. In the example, the user selects rule Ra.

4.4 Execution Phase

The rule that was selected in the selection phase is executed in the execution phase. Afterwards, the information collected during the construction phase has to be updated.

In the example (Figure 4 e), the corresponding rule of the rule node Ra is executed. As a result, the increments I4 and I5 are created and references to all increments are added to the link L1.

Rule execution is performed by a rule specific PROGRES production, see Figure 9. The left-hand side of the production is nearly identical to the right-hand side of the context check production in Figure 7. The main difference is that the edge from the link (`'10`) to the rule node (`'7`) is now a `selectedRule` edge and the edge from the rule node to the context node (`'13`) is a `selectedContext` edge. The `possibleRule` and `possibleContext` edges are replaced when a rule together with a context is selected for execution either by the user or automatically.

On the right-hand side, the new increments in the target document are created and embedded by edges. In this case, the connection (`18'`) is inserted and connected to the two Aspen ports (`14'`, `15'`). The half link (`10'`) is extended to a full link, referencing all context and non-context increments in source and target document. The information about the applied rule and roles etc. is kept to be able to detect inconsistencies occurring later due to modifications in source and target documents.

The following steps of the algorithm are performed by generic productions that update the information about possible rule applications and conflicts. First, obsolete half links are deleted. A half link is obsolete if its dominant increment is referenced by another link as non-context increment. In the example this is not the case for any half link (Figure 4 e). Then, possible rule applications that are no longer possible are removed. In Figure 4 f), Rb is deleted because it depends on the availability of I2 which is now

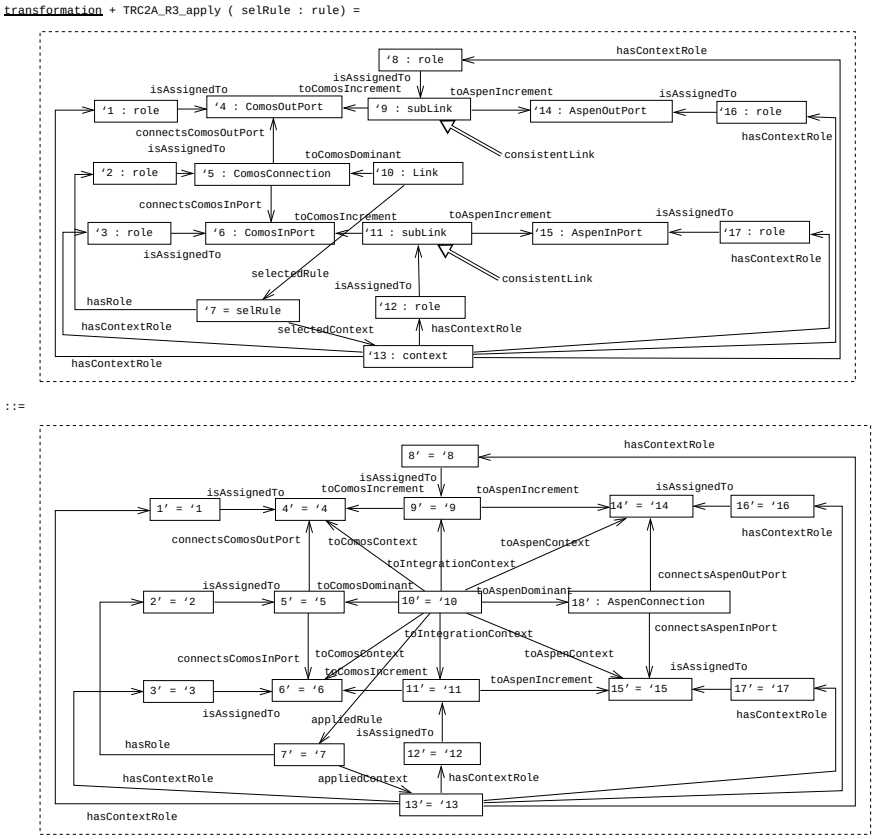

Fig. 9. Execute rule

referenced by L1 as non-context increment. If there were alternative rule applications belonging to L1 they would be removed, as well. Last, obsolete overlappings have to be deleted. In the example, O1 is removed because Rb was deleted. Please note that the cleanup procedure may change depending on how detailed the integration process has to be documented.

5 Related Work

Our approach to incremental integration for development processes is based on the triple graph grammar approach introduced by Schürr [3] and early work at our department in the area of software engineering [10] during the IPSEN project [11]. We adapted the results to the domain of chemical engineering [12] and extended the original approach: now, we are dealing with the problem of *a-posteriori* integration, the rule

definition formalism was modified [13] and the rule execution algorithm was further elaborated to support conflict detection (see Section 4).

Related areas of interest in computer science are (in-) *consistency checking* [14] and *model transformation*. Consistency checkers apply rules to detect inconsistencies between models which then can be resolved manually or by inconsistency repair rules. Model transformation deals with consistent translations between heterogeneous models. In the following a few projects of both areas are presented which are using graph transformations. Our approach contains aspects of both areas but is more closely related to model transformation.

In [15], a consistency management approach for different view points [16] of development processes is presented. The formalism of distributed graph transformations [17] is used to model view points and their interrelations, especially consistency checks and repair actions. To the best of our knowledge, this approach works incrementally but does not support detection of conflicting rules and user interaction.

The consistency management approach of Fujaba [18] supports inter-model consistency checks. The approach is based on triple graph grammars [3] as well. Comparable to our approach, different graph transformations are derived from each triple rule. User interaction is restricted to choosing the repair action for a detected inconsistency. Conflict detection between different inconsistency checking rules is supported only w.r.t. preventing endless loops if repair actions create new inconsistencies.

Model transformation recently gained increasing importance because of the model driven approaches for software development like the model driven architecture (MDA) [19]. In [20] and [21] some approaches are compared and requirements are proposed.

The PLCTools prototype [2] allows the translation between different specification formalisms for programmable controllers. The translation is inspired by the triple graph grammar approach [3] but is restricted to 1:n mappings. The rule base is conflict free so there is no need for conflict detection and user interaction. It can be extended by user defined rules which are restricted to be unambiguous 1:n mappings. Incrementality is not supported.

In the AToM project [1], modelling tools are generated from descriptions of their meta models. Transformations between different formalisms can be defined using graph grammars. The transformations do not work incrementally but support user interaction. Unlike in our approach, the control of the transformation is contained in the user-defined graph grammars.

The QVT Partner's proposal [22] to the QVT RFP of the OMG [23] is a relational approach based on the UML and very similar to the work of Kent [24]. While Kent is using OCL constraints to define detailed rules, the QVT Partners propose a graphical definition of patterns and operational transformation rules. Incrementality and user interaction are not supported.

BOTL [25] is a transformation language based on UML object diagrams. Comparable to graph transformations, BOTL rules consist of an object diagram on the left-hand side and another one on the right-hand side, both describing patterns. Unlike graph transformations, the former one is matched in the source document and the latter one is created in the target document. The transformation process is neither incremental

nor interactive. There are no conflicts because of very restrictive constraints on the rule base.

Transformations between documents are urgently needed (not only) in chemical engineering. They have to be incremental, interactive and bidirectional. Additionally, transformation rules are most likely ambiguous. There are a lot of transformation approaches and consistency checkers with repair actions that can be used for transformation as well, but none of them fulfills all of these requirements. Especially, the detection of conflicts between ambiguous rules is not supported. We address these requirements with the integration algorithm described in this contribution.

6 Conclusion

We have presented a novel approach to the execution of integration tools in incremental and interactive integration tools using graph transformations. Our approach was evaluated in an industrial cooperation with the German software company Innotec with a simplified prototype for the integration of flow sheets and simulation models implemented in C++. In parallel to our work with PROGRES, we developed a light-weight framework [12] for the rapid construction of integration tools. Current work aims at integrating the rule execution approach presented here into this framework.

Acknowledgements

This work was in part funded by the CRC 476 IMPROVE of the Deutsche Forschungsgemeinschaft (DFG). Furthermore, the authors gratefully acknowledge the fruitful cooperation with innotec.

References

1. de Lara, J., Vangheluwe, H.: Computer aided multi-paradigm modelling to process petri-nets and statecharts. In: Proc. of 1st Int. Conf. on Graph Transformations (ICGT 2002). LNCS 2505, Springer (2002) 239–253
2. Baresi, L., Mauri, M., Pezzè, M.: PLCTools: Graph transformation meets PLC design. Electronic Notes in Theoretical Computer Science **72** (2002)
3. Schürr, A.: Specification of graph translators with triple graph grammars. In: Proc. of the 20th Intl. Workshop on Graph-Theoretic Concepts in Computer Science (WG 1994). LNCS 903, Herrsching, Germany, Springer (1995) 151–163
4. Becker, S.M., Westfechtel, B.: Incremental integration tools for chemical engineering: An industrial application of triple graph grammars. In: Proc. of the 29th Workshop on Graph-Theoretic Concepts in Computer Science (WG 2003). LNCS 2880, Springer (2003) 46–57
5. Lohmann, S.: Ausführung von Integrationsregeln mit einem Graphersetzungssystem. Master's thesis, RWTH Aachen University, Germany (2004)
6. Nagl, M., Marquardt, W.: SFB-476 IMPROVE: Informatische Unterstützung übergreifender Entwicklungsprozesse in der Verfahrenstechnik. In: Informatik '97: Informatik als Innovationsmotor. Informatik aktuell, Aachen, Germany, Springer (1997) 143–154
7. innotec GmbH: COMOS PT Documentation, http://www.innotec.de. (2003)

8. Aspen-Technology: Aspen Plus Documentation, http://www.aspentech.com. (2003)
9. Schürr, A., Winter, A., Zündorf, A.: The PROGRES approach: Language and environment. Volume 2. World Scientifi c (1999) 487–550
10. Lefering, M., Schürr, A.: Specifi cation of integration tools. [11] 324–334
11. Nagl, M., ed.: Building Tightly-Integrated Software Development Environments: The IPSEN Approach. LNCS 1170. Springer, Berlin, Germany (1996)
12. Becker, S., Haase, T., Westfechtel, B., Wilhelms, J.: Integration tools supporting cooperative development processes in chemical engineering. In: Proc. of the 6th Biennial World Conf. on Integrated Design and Process Technology (IDPT-2002), Pasadena, California, USA, Society for Design and Process Science (2002) 10 pp.
13. Becker, S.M., Haase, T., Westfechtel, B.: Model-based a-posteriori integration of engineering tools for incremental development processes. Journal of Software and Systems Modeling (2004) to appear.
14. Spanoudakis, G., Zisman, A.: Inconsistency management in software engineering: Survey and open research issues. In: Handbook of Software Engineering and Knowledge Engineering. Volume 1. World Scientifi c (2001) 329–380
15. Enders, B.E., Heverhagen, T., Goedicke, M., Tröpfner, P., Tracht, R.: Towards an integration of different specifi cation methods by using the viewpoint framework. Transactions of the SDPS **6** (2002) 1–23
16. Finkelstein, A., Kramer, J., Goedicke, M.: ViewPoint oriented software development. In: Intl. Workshop on Software Engineering and Its Applications. (1990) 374–384
17. Taentzer, G., Koch, M., Fischer, I., Volle, V.: Distributed graph transformation with application to visual design of distributed systems. In: Handbook on Graph Grammars and Computing by Graph Transformation: Concurrency, Parallelism, and Distribution. Volume 3. World Scientifi c (1999) 269–340
18. Wagner, R., Giese, H., Nickel, U.A.: A plug-in for flexible and incremental consistency mangement. In: Proc. of the Intl. Conf. on the Unifi ed Modeling Language (UML 2003), San Francisco, California, USA, Springer (2003)
19. OMG Architecture Board ORMSC: Model driven architecture (MDA) (2001)
20. Gerber, A., Lawley, M., Raymond, K., Steel, J., Wood, A.: Transformation: The missing link of MDA. In: Proc. of 1st Intl. Conf. on Graph Transformations (ICGT 2002). LNCS 2505, Barcelona, Spain, Springer (2002) 90–105
21. Kent, S., Smith, R.: The Bidirectional Mapping Problem. Electronic Notes in Theoretical Computer Science **82** (2003)
22. Appukuttan, B.K., Clark, T., Reddy, S., Tratt, L., Venkatesh, R.: A model driven approach to model transformations. In: Proc. of the 2003 Model Driven Architecture: Foundations and Applications (MDAFA2003). CTIT Technical Report TR-CTIT-03-27, Univ. of Twente, The Netherlands (2003)
23. OMG: MOF 2.0 query / view / transformations, request for proposal (2002)
24. Akehurst, D., Kent, S., Patrascoiu, O.: A relational approach to defi ning and implementing transformations between metamodels. Journal on Software and Systems Modeling **2** (2003)
25. Braun, P., Marschall, F.: Transforming object oriented models with BOTL. Electronic Notes in Theoretical Computer Science **72** (2003)