Multi-Domain Integration with MOF and extended Triple Graph Grammars

Alexander Königs, Andy Schürr
Real-Time Systems Lab
University of Technology Darmstadt
64283 Darmstadt, Germany
koenigs@es.tu-darmstadt.de, schuerr@es.tu-darmstadt.de

Abstract

One aim of tool integration is designing an integrated development environment that accesses the data/models of different tools and keeps them consistent throughout a project being considered. Present approaches that aim for data integration by specifying (graphically denoted) consistency checking constraints or consistency preserving transformations are restricted to pairs of documents. We present an example that motivates the need for a more general data/model integration approach which is able to integrate an arbitrary number of MOF-compliant models. From a formal point of view this approach is a generalization of the triple graph grammar document integration approach. From a practical point of view it is a proposal how to specify multi-directional declarative model transformations in the context of OMG's model-driven architecture (MDA) development efforts and its request for proposals for a MOF-compliant "query, view, and transformation" (QVT) approach.

1 Introduction

Software development projects are subdivided into a number of phases. There are lots of different tools specialised in each of these phases. Thus, the data of a project as a whole is distributed over the tools being adopted. Tool integration tries to design an integrated development environment which offers uniform access to the data of the different tools and keeps them consistent.

Present approaches describe dependencies between data of different tools by specifying consistency rules. These rules are often written in a graphical form using a UML-like notation [UML]. All approaches have in common that they fulfil the task of keeping the data consistent throughout a project as a whole by considering only pairs of documents at one time.

The *Request for Proposal: MOF 2.0 Query / View / Transformations RFP* of the OMG deals with queries, views and transformations on models of two documents. It demands a number of features which can be used for classifying approaches [QVT]. Each response to the *RFP* must:

- offer a language for specifying queries for selection and filtering of model elements.
- provide a language for model transformation definitions. These definitions can be used to generate a target model from a source model.
- have a MOF 2.0-compliant abstract syntax of each language.
- have an expressive transformation language allowing automatic transformations.
- support the creation of views.
- support incremental change propagation between source and target model.

Additionally, a response may:

- offer transformations which can be executed bidirectional.
- provide traceability information.
- use generic transformation definitions for reuseability purposes.
- provide some sort of transactional mechanism.
- support the use of additional data which is not contained in the source model.
- allow transformations for the case that the source and the target model coincide.

Dagstuhl Seminar Proceedings 04101
http://drops.dagstuhl.de/opus/volltexte/2005/22

Furthermore, approaches can be classified from a more technical point of view covering the following issues [CH03]:

- Features of transformation rules (e.g. syntactically separated left-hand and right-hand sides, parameterization)
- Features of rule application scoping
- Source-Target Relationship
- Rule application strategies (e.g. deterministic, non-deterministic)
- Transformation rule scheduling
- Rule organization (e.g. source-oriented, target-oriented)
- Traceability support
- Directionality

The QVT response from the QVT-partners is an approach that allows the specification of consistency and transformation rules [QVTP03]. The consistency rules can be used to check whether two linked data objects are consistent or not. They cannot be used to recover consistency, propagate data changes, or for traceability purposes. The transformation rules can be used to transform one document of a source domain into a consistent document of a target domain in a unidirectional manner. Both rule types must be specified separately and are not generated from one declarative rule. The rules are written down in a textual format but can be visualised in a UML-like diagram.

The GReAT approach only defines transformation rules between source and target domains [AKS03]. This transformation happens in a unidirectional way, too. The rules are denoted in a UML-like format and cannot be used for consistency recovery, data change propagation, or traceability due to their transformational character.

In the BOTL approach consistency checking and transformation rules are specified in a declarative way and provide bidirectional transformations [Braun03]. Again, these rules cannot be used for change propagation or traceability issues. The notation is UML-like.

Finally, the IMPROVE framework allows the declarative specification of consistency and transformation rules [BW03]. These rules provide bidirectional transformations, consistency checks, traceability, and incremental change propagation. This framework uses a UML-like notation for specification.

As stated before the main common disadvantage of the presented approaches is the limitation to pairs of documents at one time. *Fig. 1* summarizes the properties of the presented approaches.

	QVT	GReAT	BOTL	IMPROVE
Declarative Specification	-	-	+	(+)
Consistency Checks	+	-	(+)	+
Transforma-tions	unidirec-tional	unidirec-tional	bidirectional	bidirectional
Notation	textually	graphically	graphically	graphically
MF-based	+	+	+	+
MOF-compliant	+	-	-	-
Traceability	-	-	-	+
Change Propagation	-	-	-	incremental
Multi document support	-	-	-	-

Fig. 1: *Properties of related approaches*

In this paper we present an example which motivates the need for a more general data/model integration approach which is able to simultaneously integrate an arbitrary number of documents. Our new approach will be implemented in the *ToolNet* framework provided by our industrial partner from the automotive sector. [ADDK03].

Sections 2-4 introduce this example. Section 2 presents concrete example data for our project. In section 3 we derive objects diagrams from them. These object diagrams are compliant to project-specific data meta-models as shown in section 4. In section 5 we will explain which disadvantage approaches that only consider pairs of documents suffer from. Section 6 covers triple graph grammars which form the theoretical background for our proposal. Section 7 shows the advantage of declarative multi-domain integration (MDI) rules. Finally, section 8 concludes this paper and discusses open issues that arise from our new approach..

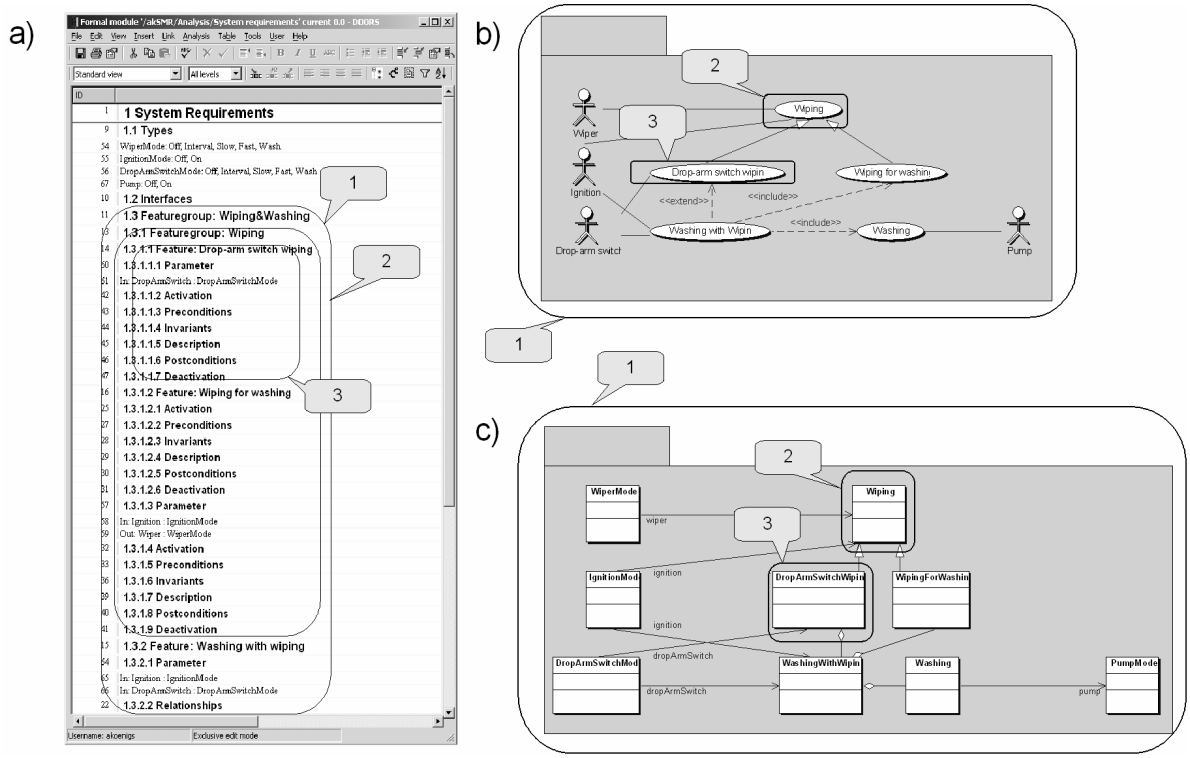

Fig. 2: *Concrete project's data with Requirements, Use Case Diagram and Class Diagram documents*

2 Concrete project's data

In this section we will introduce our example by presenting concrete data stored in the documents we want to integrate with each other. The example comes from the automotive sector. Due to lack of space we had to simplify the example in this paper. It covers the following domains:

1. *Requirements Engineering:* We keep the data of this domain in the tool *DOORS* because it is widely used by our industrial partner and we do have already an adapter for this tool implemented in our *ToolNet* framework [DOORS, ADDK03].
2. *Use Case Diagrams:* We use *Together* to draw our use case diagrams for the same reasons. [Together].
3. *Class Diagrams:* Again, we use *Together* to draw these diagrams.

The example deals with the development of a rain sensor-controlled windscreen wiper. As one simplification we integrate one functional requirement with one use case as well as one use case with one class. This does not apply in practice. *Fig. 2 a)* shows a screenshot of the data of the requirements kept in *DOORS.*

For our purposes it is important to remember that our requirements are stored as a structured text in a tree-like manner which offers the opportunity of nesting elements. We make use of this opportunity to nest feature groups. A feature group is either a collection of single features describing different aspects of one system function, or a collection of system functions that have similar characteristics (e.g. pre- and postconditions, interfaces). From this document we derive a use case diagram as shown in Fig. 2 b). We also derive a corresponding class diagram (see Fig. 2 c)).

We learn from the use case and the class diagram that we have two possibilities to represent nested structures from the requirements document. Either we can represent nested feature groups as (nested) packages (see bubble 1 in Fig. 2). We will use this for collections of features. Or we can represent nested feature groups as a generalization between use cases and accordingly classes (see

bubbles 2 and 3 in Fig. 2). We will use this for collections of system functions with similar characteristics.

3 Object diagrams

From the concrete data introduced in section 2 we can infer object diagrams for each considered document. *Fig. 3* shows a part of the object diagram of the use case document as an example.

This figure shows an object which represents a package. This package contains a diagram which contains three use cases. The use cases are associated by generalization relationships.

Accordingly, we can draw object diagrams for the requirements and the class documents as well.

Our approach as well as related approaches uses such object diagrams for the specification of graph-) rules on them.

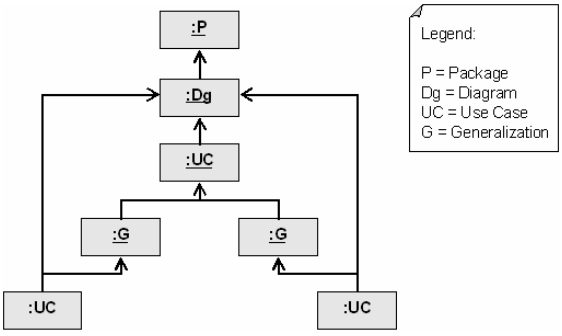

Fig. 3: *Part of the object diagram of the use case document*

4 Project specific data meta-models

In order to be able to draw object diagrams as in section 3 we have to introduce (MOF-compliant) meta-models which declare the used classes and the allowed connections between them.

Fig. 4 shows a part of the data meta-model corresponding to the object diagram from *Fig. 3*. This data meta-model specifies that we have abstract containers which can be packages or diagrams. Packages can be nested. Diagrams contain use cases. Use cases can be the source and the target of abstract relationships. This can be a generalization relationship for instance. We can specify similar data meta-models for the requirements and the class diagram documents as well.

It is important to remember that this data meta-model is not the tool's internal data meta-model. Usually, tool internal data meta-models are too generic for our purposes and must be refined to project specific data meta-models in order to have type information rich enough to allow the specification of (graph-) rules. To be able to access the tool's data using the project-specific data meta-model we must provide a mapping from the project-specific to the tool internal data meta-model. In our framework this mapping is realised by the tool adapters.

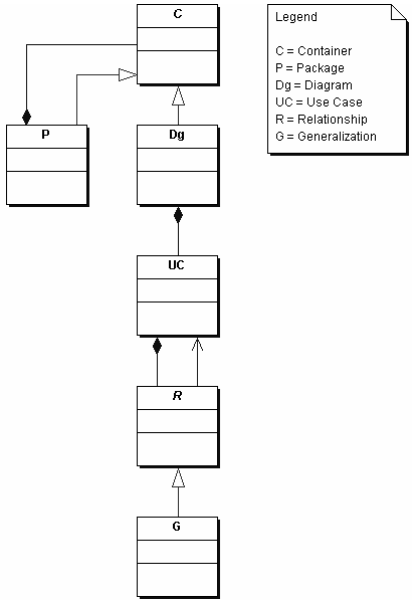

Fig. 4: *Part of the data meta-model of the use case document*

5 Considering pairs of documents

We will now investigate if we can keep the three documents consistent with each other by only considering pairs of documents at one time. In *Fig. 5* we see the alternatives we have to do so.

Fig. 5 a) states that we can try to achieve our goal by integrating the requirements document with the use case diagram document and integrating the use case diagram document with the class diagram document. This approach fails because the requirements document contains information on parameter types which is needed in the class diagram document but not provided by the use case document.

In *Fig. 5 b)* we are trying to integrate the requirements document with the class diagram document and the class diagram document with the use case document. This attempt fails, too. The

requirements document contains information on *includes* and *extends* relationships which is needed by the use case document. In the class diagram document we represent both relationships as aggregations. Thus, the class diagram document does not provide this information.

We avoid the problems from *Fig. 5 a)* and *b)* in *Fig. 5 c)* by integrating the requirements document with the use case diagram document and the requirements document with the class diagram document. The requirements document provides all information needed in the use case document as well as in the class diagram document. Unfortunately, we experience another kind of problem. As mentioned in section 2 we have two possibilities to represent nested structures in the requirements documents in the use case and class diagram documents. *Fig. 5 c)* does not demand that the choice which possibility to use must be made accordingly. This permits representing a nested structure in the requirements document as nested packages in the use case diagram document and as a generalization in the class diagram document and vice versa. Although this solution does not lead to a data inconsistency between the use case diagram and the class diagram document it blurs the correspondence between both documents. We can see this as a kind of structural inconsistency which we want to avoid.

We avoid this by using the last alternative shown in *Fig. 5 d)*. This means that we can integrate three documents by considering only pairs of documents at one time. We will now take a look at the number of sets of rules we need to specify the integration.

We can easily see that the non-declarative, unidirectional approaches (e.g. QVT, GReAT) need three specifications (sets of rules) to specify consistency checking (if they provide this feature at all) and additional six sets of specifications to cover all directions of transformations between the involved three types of documents. In total these are nine specifications for integrating three documents. Declarative and bidirectional approaches (e.g. BOTL, IMPROVE) still need three specifications.

The more the number of document types we want to simultaneously integrate increases the more the situation gets worse. Thus, considering only pairs of documents at one time to achieve data integration is inappropriate for real projects where the number of to be integrated document types is large.

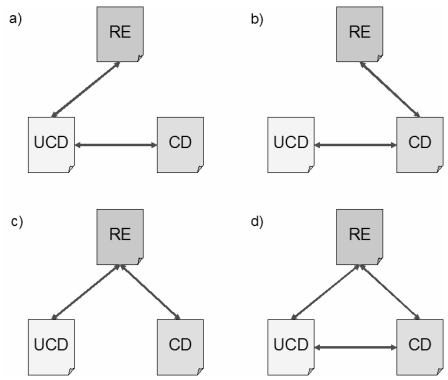

Fig. 5: *Alternatives to achieve data integration considering pairs of documents*

6 Triple Graph Grammars

In order to cope with the situation described in section 5 we want to propose a new approach for specifying data integration rules on the basis of graph transformations. The theoretical background for our approach is formed by triple graph grammars [Sch94].

Triple graph grammars are an extension of pair graph grammars [Pra71]. They allow the (graphical) specification of data integration rules considering a pair of documents. The idea is that the models of each document can be interpreted as a graph as we did in section 3. A triple graph grammar specifies the simultaneous construction of the graphs of both documents. Additionally, it builds up a third graph which contains the information on correspondence of objects of the first document to objects of the second one. In particular this correspondence graph can be used for traceability purposes.

At first triple graph grammars were used in the IPSEN project to build tightly-integrated software development environments [LS96]. There a triple graph grammar specification was manually translated into code. The IMPROVE approach [BW03] mentioned in section 1 is a continuation of the IPSEN approach. Later on triple graph grammars were used to specify the migration of relational to object-oriented database systems using the PROGRES environment [JSZ96, Sch91]. Finally, the FUJABA environment adopted triple graph grammars to realize a consistent management system for UML-specifications [NNZ00, Wag01].

7 Proposing MDI-rules

We want to propose a new approach for specifying data integration rules on the basis of graph transformations extending the triple graph grammar approach from section 6. The new rules are not limited to objects from only two documents but can use objects of an arbitrary number of documents. *Fig. 6* illustrates this. We call these rules M*ulti-*D*omain* I*ntegration rules*.

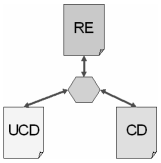

Fig. 6: *Concept of MDI-rules*

The aim is that we only need one set of rules for specifying simultaneous data integration for an arbitrary number of documents. Our approach is declarative and multi-directional.

To clarify our idea we give one example for MDI-rules that solve the problem which came up in section 5 using the alternative from *Fig. 5 d)*. *Fig. 7* presents our solution.

The left part of *Fig. 7 a)* represents the situation which is searched for in the given documents. It searches for a feature group *FG* in the requirements document *RE* which is related to a package *P* in the use case diagram document *UCD* and to a package *P* in the class diagram document *CD* and fulfils the given OCL-expression [OCL]. If this pattern is found and a new feature or feature group *Fs* is inserted in the selected feature group a new use case *UC* is simultaneously inserted in the associated package *P* as well as a new class *Cl* in the corresponding package.

This rule represents the choice to represent a nested structure in the requirements document as packages in the use case and the class diagram documents. In the same way the rule from *Fig. 7 b)* represents the choice to represent a nested structure from the requirements document as generalization in the use case and the class diagram document.
As these rules use objects from all three considered documents at the same time we ensure that choices how to represent nested structures are made correspondingly.

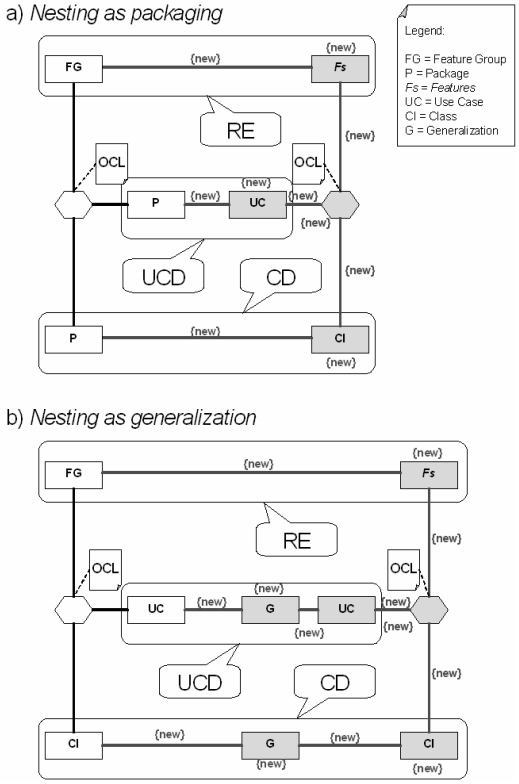

Fig. 7: *Examples for MDI-rules*

From a formal point of view our rules are generalizations of the triple-graph grammar approach. In the example we used a quadruple-graph grammar consisting of the following four graph grammars:

1. The first grammar describes the composition of the requirements document.
2. The second grammar describes the composition of the use case diagram document.
3. The third grammar describes the composition of the class diagram document.
4. Finally, the fourth grammar describes the composition of the integration document formed by the integration relationship objects.

For an arbitrary number of documents we can call our formal approach multi-graph grammars.

Using our proposal we only need to specify one set of rules that describes the data integration for an arbitrary number of considered documents. The price for this is that each rule will be more complicated.

As our approach is declarative we can derive several operational rules from each MDI-rule by omitting *{new}*-tags. *Fig. 8* gives examples for such rules.

Fig. 8 a) is a consistency checking rule. It searches for the graph pattern as a whole and tests whether the given OCL-expressions hold or not. *Fig. 8 b)* is a rule which links existing objects from all documents by creating a consistency object associated to them. *Fig. 8 c)* is an example for a rule which creates parts of two documents by using the third document. In our case we get three of these rules. Finally, *Fig. 8 d)* is an example for a rule which creates parts of one document by using the others. In our case we are able to generate three of these rules, too.

Thus, we can totally derive eight operational rules by specifying a single MDI-rule if we consider three documents. As the number of documents increases the number of rules we can derive increases correspondingly.

To be honest all declarative approaches only succeed if they can automatically derive attribute assignments from attribute constraints (e.g. OCL-expressions). As we point out in section 8 this is an open issue which is addressed by the ongoing research on constraint solving strategies [BKPT02].

8 Conclusion

In this paper we give an example of a small data integration task considering three development documents. First, we try to enforce data consistency by specifying consistency rules that only consider two of the three documents at one time. We see that we can do it this way.

We learn that the number of specifications rapidly increases with the number of involved document types. To cope with this we present a new approach for specifying consistency rules on basis of a generalization of the triple-graph grammar approach. These rules consider objects of all involved documents at the same time. Thus, we only need one set of rules for an arbitrary number of documents. The price for this is that the rules become more complicated. We are convinced that it is worth the effort due to the number of operational rules which can be derived from each declarative rule.

Compared to the related approaches we presented in section 1 our MDI-approach will provide the same features as the IMPROVE

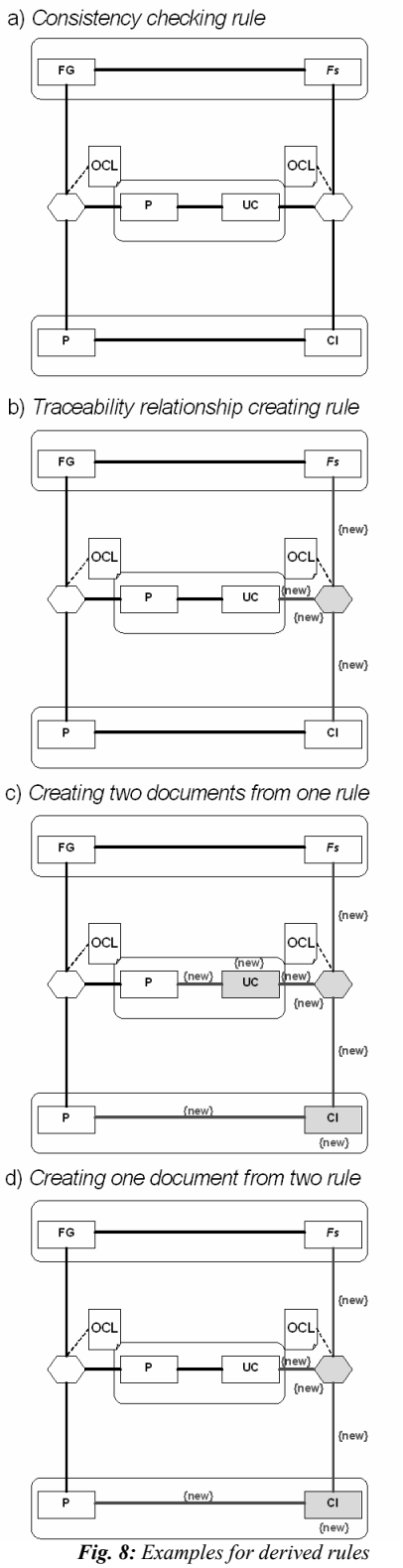

Fig. 8: *Examples for derived rules*

approach and adds MOF-compliance as well as multi document support [BW03]. Our approach will fulfil all mandatory and most of the optional requirements (except for transactional mechanisms) requested by the QVT-RFP of the OMG [QVT].

Among other things we will have to address the following issues when implementing the presented MDI-approach as an extension of FUJABA [NNZ00]:

1. We have to develop new strategies for processing more than two documents for efficiently pattern matching purposes.
2. We have to consider how to deal with the situation when more than one rule is applicable in a given (graph-) situation. Either the user will be asked to interactively resolve this ambiguity, or we allow the specification of priorities on MDI-rules.
3. We have to examine possibilities to automatically derive attribute assignments from consistency checking OCL-expressions to push our declarative approach as far as possible. Simple equality constraints are no problem at all. More complex cases can be handled by reusing constraint solving strategies for translating undirected more complex equations into directed equations. Inequalities or more complex Boolean expressions are out of scope at the moment.
4. We want to integrate our approach with the upcoming MOF 2.0 specification [MOF].
5. Finally, we want to implement incremental consistency checks and change propagation instead of the existing batch approach.

References

[ADDK03] Altheide, Dörfel, Dörr, Kanzleiter, *An Architecture for a Sustainable Tool Integration*, in: Dörr, Schürr (eds.), TIS 2003, Workshop on Tool Integration in System Development, 2003, pp. 29-32

[AKS03] Agrawal, Karsai, Shi, *Graph Transformations on Domain-Specific Models*, Technical report, Institute for Software Integrated Systems, Vanderbilt University, Nashville, TN, 2003.

[BKPT02] Bottoni, Koch, Parisi-Presicce, Taenzer, *Working on OCL with Graph Transformation*, Proc. of APPLIGRAPH Workshop on Applied Graph Transformation (AGT 2002), Grenoble, France, 2002, pp. 1 – 10

[Braun03] Braun, *Metamodellbasierte Kopplung von Werkzeugen in der Softwareentwicklung*, Dissertation, 2003

[BW03] Becker, Westfechtel, *Incremental Integration Tools for Chemical Engineering: An Industrial Application of Triple Graph Grammars*, in Proc. 29th Intl. Workshop Graph-Theoretic Concepts in Computer Science, Elspeet, Netherlands, 2003

[CH03] Czarnecki, Helsen, *Classification of Model Transformation Approaches*, 2nd OOPSLA Workshop on Generative Techniques in the context of Model Driven Architecture, Anaheim, California, 2003

[DOORS] Telelogic, *DOORS*, http://www.telelogic.com/products/doorsers/doors

[JSZ96] Jahnke, Schäfer, Zündorf, *A Design Environment for Migrating Relational to Object Oriented Database Systems*, in: Proc. Of the International Conference on Software Maintenance, IEEE Computer Society Press, 1996, pp. 163-170

[LS96] Lefering, Schürr, *Specification of Integration Tools*, in: Nagl (ed.), Building Tightly-Integrated Software Development Environments: The IPSEN approach, LNCS 1170, Berlin, Springer Verlag, 1996, pp. 440-456

[MOF] OMG, *MOF2.0 Specification*, http://www.omg.org/cgi-bin/doc?ad/2003-04-07

[NNZ00] Nickel, Niere, Zündorf, *The FUJABA Environment*, in: Proc. Of the 22nd Internation Conference on Software Engineering, ACM Press, 2000, pp. 742-745

[OCL] OMG, *OCL Specification*, http://www.omg.org/cgi-bin/doc?ad/2003-01-07

[Pra71] Pratt, *Pair Grammars, Graph Languages and String-to-Graph Translations*, in: Journal of Computer and System Sciences, Vol. 5, Academic Press, 1971, pp. 560-595

[QVT] OMG, *Request for Proposal: MOF 2.0 Query / Views / Transformations RFP*, http://www.omg.org/cgi-bin/doc?ad/2002-04-10

[QVTP03] QVT-partners, *QVT-Partners revised submission to QVT*, http://qvtp.org/downloads/1.1/qvtpartners1.1.pdf

[Sch91] Schürr, *Operationales Spezifizieren mit programmierten Graphersetzungssystemen*, Dissertation (German), Deutscher Universitätsverlag, 1991

[Sch94] Schürr, *Specification of graph translators with tripel graph grammars*, in: Mayr, Schmidt (eds.), Proc. WG'94 Workshop on Graph-Theoretic Concepts in Computer Science, LNCS 903, Springer, Herrsching, 1994, pp. 151-163

[Together] Borland, *Together*, http://www.borland.com/together

[UML] OMG, *UML 2.0 superstructure specification*, http://www.omg.org/cgi-bin/doc?ptc/2003-08-02

[Wag01] Wagner, *Realisierung eines diagrammübergreifenden Konsistenzmanagement-Systems für UML-Spezifikationen*, Diploma Thesis (German), Universität Paderborn, 2001