Towards Bidirectional Engineering of Satellite Control Procedures Using Triple Graph Grammars

Susann Gottmann[1], Frank Hermann[1], Claudia Ermel[2], Thomas Engel[1], and Gianluigi Morelli[3]

[1] Interdisciplinary Centre for Security, Reliability and Trust,
Université du Luxembourg, Luxembourg
`firstname.lastname@uni.lu`
[2] Technische Universität Berlin, Germany
`firstname.lastname@tu-berlin.de`
[3] SES, Luxembourg
`firstname.lastname@ses.com`

Abstract. The development and maintenance of satellite control software are very complex, mission-critical and cost-intensive tasks that require expertise from different domains. In order to adequately address these challenges, we propose to use visual views of the software to provide concise abstractions of the system from different perspectives.
This paper introduces a visual language for process flow models of satellite control procedures that we developed in cooperation with the industrial partner SES for the satellite control language SPELL. Furthermore, we present a general and formal bidirectional engineering approach for automatically translating satellite control procedures into corresponding process flow visualisations. The bidirectional engineering framework is supported by a visual editor based on Eclipse GMF, the transformation tool HenshinTGG, and additional extensions to meet requirements set up by the specific application area of satellite control languages.

Keywords: model transformation, model synchronisation, triple graph grammars, bidirectional engineering, Eclipse Modeling Framework (EMF)

1 Introduction

Development and maintenance of satellite control software are very complex, mission-critical and cost-intensive tasks demanding expertise from different domains. We address these challenges by a general approach we developed in a joint research project with the industrial partner SES (Société Européenne des Satellites, `http://www.ses.com/`). SES is a world-leading satellite operator currently operating a fleet of 53 satellites of different vendors. The satellite control programming language SPELL (Satellite Procedure Execution Language & Library) [23] was initiated by SES to become a new standard. It is an open-source package based on Python for the development and operation of satellite control procedures.

The main goal of the research project is to develop the visual modelling language *SPELLFlow*, which represents the control flow of SPELL procedures. Satellite engineers and operators at SES are currently working with the SPELL source code. In the development of SPELL source code, engineers already work with a visual representation of the desired source code (as printout), but it is completely uncoupled from the SPELL development and execution environment. In order to enhance the daily work and reduce cost-intensive errors, a visualisation, related to the one used by satellite engineers, is desired with further improvements: It abstracts from the source code and highlights important commands for providing a more intuitive way of input. However, it shall not lose detailed information, which will be hidden at the beginning and can be shown, if the user desires. So, we developed a layered concept (c.f. Sec. 3). SPELLFlow is adapted to the following domain specific requirements set up by SES: **(1)** Provide a hierarchical visual model defining different layers of abstraction for highlighting more important information, but without losing detailed information. **(2)** Emphasise certain SPELL statements, e.g., commands for sending and receiving telemetry data. **(3)** The bidirectional engineered model will be used as a concise visual view on the source code. Hence, the engineering process between SPELL and SPELLFlow has to yield correct visual models and has to retain functional behaviour, i.e., the translation terminates and yields the same result for identical inputs.

For the translation we use triple graph grammars (TGGs) [21,22], a bidirectional formal technique in the field of graph transformation. Several correctness properties are ensured due to the usage of TGGs (syntactical correctness of translation results, functional behaviour, completeness of translation, i.e., every input graph can be translated).

In the former successful joint research project PIL2SPELL [16] with SES, we developed an automated translation based on TGGs from satellite procedures written in the proprietary satellite operation language PIL of the satellite manufacturer ASTRIUM into SPELL. In the current project, we will reuse results, like the SPELL grammar for the conversion from concrete syntax of SPELL into the abstract syntax graph. At a later stage, we will provide an extension to a model synchronisation framework based on [15], which will be integrated at SES into the daily work of satellite developers and controllers. The bidirectional engineering framework is supported by the transformation tool HenshinTGG [8,13]. A visual editor *SPELLFlowEditor* based on Eclipse GMF [2] was generated and extended to the specific requirements of SES.

Sec. 2 introduces the running example. Sec. 3 presents the visual language SPELLFlow and the approach for the translation. Sec. 4 summarises the applied formal techniques. In Sec. 5 we discuss related work and conclude in Sec. 6.

2 Running Example

```
def fib(n):
  a = 0
  b = 1
  for i in range(n):
    sum = a + b
    b = a
    a = sum
  #ENDFOR
  Prompt('Result: ' + a, OK)
  return
#ENDDEF

Step('1', 'User Input')
nr = Prompt('Number: ', NUM)
fib(nr)
if(Prompt('Restart?',YES_NO) == YES):
  Goto('1')
#ENDIF
```

Fig. 1. Example Procedure in *SPELL* syntax

Throughout the paper, we use the SPELL source code in Fig. 1 as running example. It is not satellite-specific but well-known and complex enough to explain all details of the approach. The program prints the Fibonacci number of a given user input. Lines 1 - 11 depict the subroutine `fib(n)`, which determines the Fibonacci number for the given parameter `n` and outputs the result. In line 13, the main program starts with the SPELL-specific command `Step`. This command indicates a label (first parameter) used for jumps and provides a description (second parameter). Line 14 asks for user input to be given as parameter to the subroutine in line 15. Afterwards, the user is able to decide, whether she wants to restart the procedure. If the user answers the prompt with `YES`, then the application jumps back to line 13 using the Goto command.

3 Methodology

This section describes the general approach for the bidirectional engineering of satellite control procedures written in SPELL into its flow visualisation and vice versa, We introduce the desired visualisation, which is developed in cooperation with SES and present the approach for implementing the bidirectional translation.

Desired Visualisation

In Sec. 1, we introduced requirements for the visualisation. To fulfil requirement 1 (hierarchical visual model), we developed a layered model containing different abstractions. In practice, it represents the call-hierarchy of a diagram out of another diagram, i.e., we provide the possibility to go from one more abstract layer to an underlying one which contains more fine-grained details. The first layer shows only relevant control structure parts of the main procedure - with the industrial partner SES, we elaborated special rules for defining the first layer out of the source code: the first branch of `if`, `for`, `while` or `try` statements, function calls, `Goto` and `Step` commands shall be situated on the first layer. The second layer will contain more detailed information, e.g., further branches, body of functions called on the previous layer. In general, the richness of detail is increasing with a growing layer depth. In the visual representation, shapes for statements of the same type which directly follow each other, are merged (see Ex. 1).

Requirement 2 (focus on telemetry data) is realised by specific shapes for very important SPELL statements (e.g., send and receive telemetry commands, steps, prompts).

Example 1 (SPELLFlow - concrete syntax). Fig. 2 illustrates the visualisation of the SPELL procedure (Fig. 1). Note, that we use the diagram number on the right top of each box in the following description. The first layer (diagram 1) contains the main procedure according to the rules mentioned before. The box following `Step 1 : User Input` belongs to the `Step` command and links to another diagram (number 2) on second layer, which contains a statement (assignment) that is allocated to this step but should not occur on first layer. If there are further statements, which should not occur on the first layer, boxes with + icons are shown signalling links to further layers. The `Goto` is visualised by a pentagon shape which links to the target statement - the step statement. In diagram 1, the box `Call fib(nr)` indicates a function call. It links to

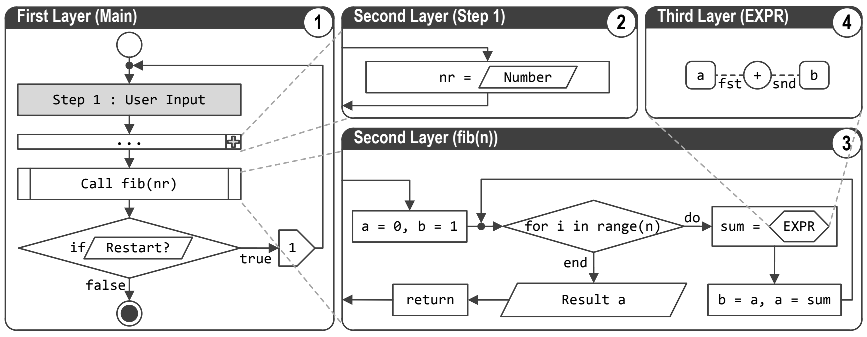


Fig. 2. Desired visualisation for the example procedure in *SPELL*

diagram 3, which is situated on second layer. The content of the function is represented by diagram 3. The shapes of statements of the same type are merged, e.g., in box `b = a, a = sum`. The `Prompt` statement gets a special shape (rhomboid), and expressions (hexagon), that are at least binary. Expressions are depicted explicitly in separate diagrams on the next layer. Consequently, hexagon `a + b` links to diagram 4, which shows the expression in full detail on third layer.

General Bidirectional Engineering Approach

The approach for bidirectional engineering of SPELL (Fig. 3) from source code to its visualisation and vice versa will be integrated in two SES applications: the SPELL execution environment , which is used for operating satellites and the SPELL development environment, in which the SPELL programmer gets the possibility to implement SPELL procedures in the source code view and also in using the visualisation of SPELLFlow models for creating a skeleton as a way of code generation. Both SPELL environments are represented by the rounded rectangle on top of Fig. 3. Currently, SES uses SPELL source code - the concrete syntax of SPELL. For the bidirectional engineering process from SPELL source code to visual SPELL models (SPELLFlow), we use the Eclipse tool HenshinTGG, which is based on EMF [3]. The concrete syntax of SPELL will be imported in HenshinTGG using Xtext [4] which results in an abstract syntax graph (ASG) of the SPELL source code. We use HenshinTGG to define triple rules and generate forward translation rules (for the translation from SPELL ASG to SPELLFlow ASG) and generate backward translation rules (for the translation from the SPELLFlow ASG to the SPELL ASG). Using EMF, the abstract syntax graph of SPELLFlow is transferred to the concrete syntax, an XMI file, which can be imported into the SPELLFlowEditor to visualise the flow diagram. The SPELLFlowEditor for

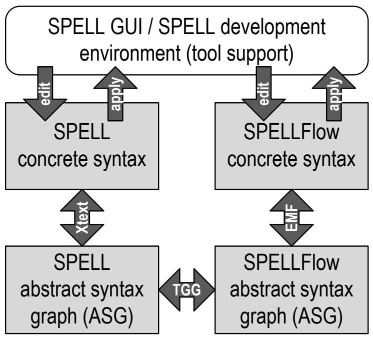

SPELL GUI / SPELL development environment (tool support)
edit
apply
edit
apply
SPELL concrete syntax
SPELLFlow concrete syntax
Xtext
EMF
SPELL abstract syntax graph (ASG)
TGG
SPELLFlow abstract syntax graph (ASG)

Fig. 3. Bidirectional engineering from *SPELL* to *SPELLFlow* and vice versa

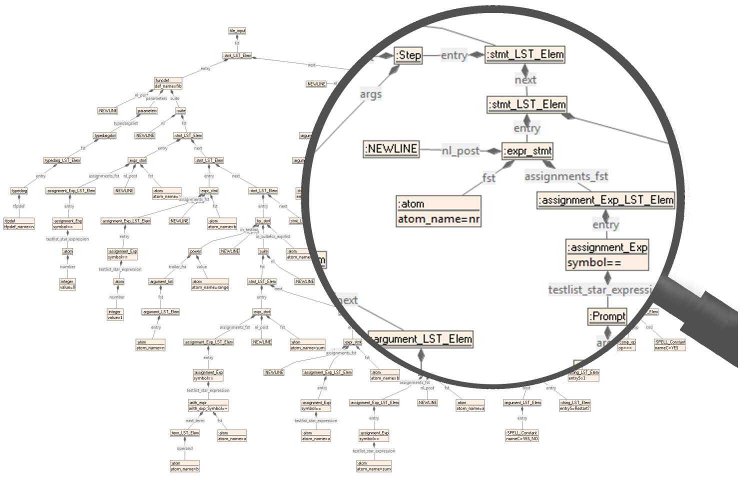


Fig. 4. Abstract syntax graph of the running example

the visualisation currently exists as a prototype and will be integrated into both SPELL environments.

Example 2. In Fig. 4 the abstract syntax graph (ASG) of the running example is illustrated. Due to the complexity, we highlight a detail of the ASG which represents parts of source code lines 13 and 14 (step and assignment statements, see Fig. 1). The ASG is a graph typed over the source part of the type graph. The types are indicated by "':'", e.g., `stmt_LST_elem` is the type of the second node from top.

4 Formal Framework and Application

In the following section, we briefly introduce the main concepts for model transformations based on TGGs [5] on the basis of the running example and the synchronisation framework [15] that we use in the project.

A triple graph is an integrated model, i.e., a model which is composed of a source model, a target model and correspondences between these models. It consists of three graphs: the source, correspondence, and target graphs, and two graph morphisms (mappings) specifying the correspondences between elements of the source and target model. In Fig. 7 an excerpt of the triple graph for our running example is given. A triple graph morphism defines mappings between triple graphs which preserve the correspondences.

Triple graphs are typed over a type triple graph *TG* via a triple graph morphism. Triple graph morphisms between triple graphs have to preserve the typing. *TG* can be seen as the meta-model. Triple graphs can have attributes and node type inheritance. For this, we use the formal notation presented in [5,6].

$$\begin{tikzcd} L \arrow[r, hook, "tr"] \arrow[d, "m"'] \arrow[dr, phantom, "(PO)"] & R \arrow[d, "n"] \\ G \arrow[r, hook, "t"'] & H \end{tikzcd}$$

Fig. 5. Tripe rule

A triple rule *tr* as shown on the first row of Fig. 5 is an inclusion of triple graphs $L \hookrightarrow R$, i.e., all elements in L are uniquely mapped

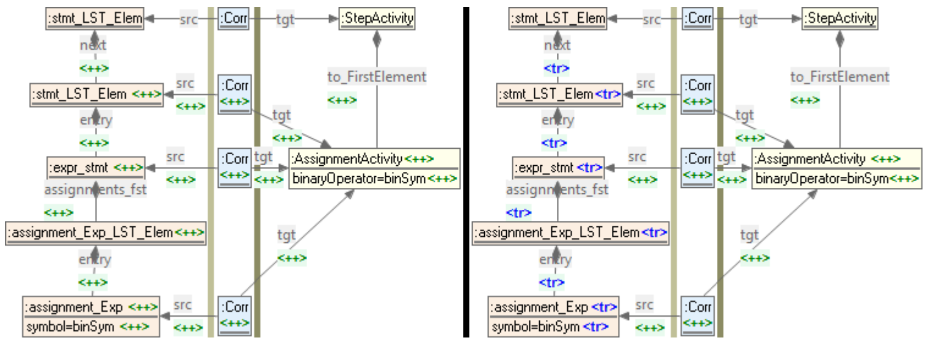


Fig. 6. Triple rule *T_Step_assignment_Expr-2-AssignmentActivity* (left) and derived FT rule (right)

to elements in R. Consequently, triple rules are non-deleting. They specify, how a consistent triple graph can be extended on all three parts simultaneously resulting again in a consistent triple graph. The rule application is illustrated in Fig. 5. The triple rule tr is applied to triple graph G via a graph morphism m called match. The result is triple graph H, where L is replaced by R in G [6]. Triple rules can be extended by negative application conditions (NACs) defining forbidden context in order to restrict the rule application [5].

Example 3 (Triple Rule). In Fig. 6, a triple rule is illustrated. Elements marked with <++> are created by this triple rule. Unmarked elements are called context elements. This triple rule creates correspondences between an assignment expression following a `Step` command in the SPELL ASG with an assignment activity in the SPELLFlow ASG. The latter element is situated on a new layer in the target graph. The new layer is indicated by containment edges, i.e., the assignment activity is contained by the step activity.

A TGG is a tuple $TGG = (TG, S, TR)$ containing a type triple graph TG, a start graph S, which is usually the empty triple graph, and a set of triple rules TR. A TGG generates all consistent triple graphs. For $TG = (TG_S \leftarrow TG_C \rightarrow TG_T)$, we use $\mathcal{L}_{TG}$, $\mathcal{L}_S$, $\mathcal{L}_T$ to denote the language, i.e., the classes of all graphs typed over TG, TG_S, or TG_T, respectively.

For the translation from the source into the target model, we use a set of operational forward translation rules (FT rules) that are generated automatically out of the set of triple rules [14]. Each FT rule tr_{FT} differs only on the source part from the corresponding triple rule tr: Each <++> is replaced by a Boolean valued marker `<tr>`. In order to translate a source model to an integrated model, all elements in the source model are initially marked with false. When applying an FT rule, the `<tr>`-marker is set to true, so that the specific rule cannot be applied again on the same elements. So, the source model will be translated stepwise into an integrated model, without modifying the source model. Similar to the generation of FT rules, operational backward translation rules (BT rules) can be created in order to translate backward a target model into the integrated model.

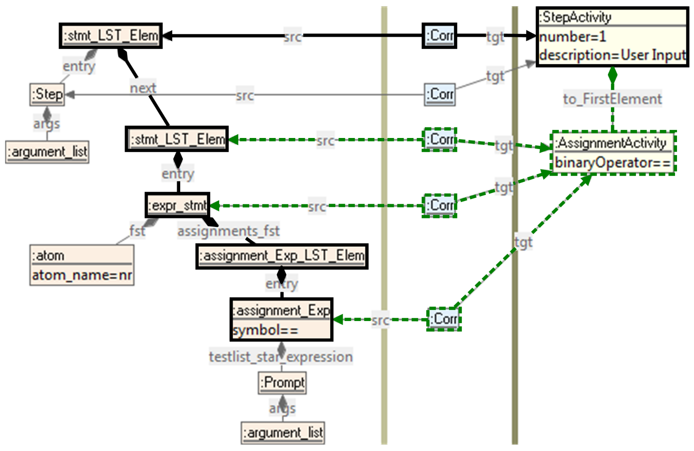


Fig. 7. Excerpt from triple graph (integrated model)

Example 4 (FT/BT Rule and FT-Rule Application). We consider the triple rule in Fig. 6. In the corresponding FT rule, each `<++>` marker is replaced by a `<tr>` in the source part. In the corresponding BT rule, the marker is replaced in the target part.

In general, the translation from a source model into a target model needs the source graph as a basis. In our example, the translation is performed from the SPELL ASG, which is illustrated in Fig. 4. It will be translated stepwise into an integrated model with the set of FT rules. In Fig. 7 we illustrate the application of the FT rule *FT_Step-assignment_Expr-2-AssignmentActivity* generated out of the triple rule in Fig. 6. The Step statement is already translated, so that the rule *FT_Step-assignment_Expr-2-AssignmentActivity* can be applied. The elements marked with a fat border are required context elements which are mapped by the FT rule. After applying the FT rule, the elements marked with a dashed line are created by this FT rule.

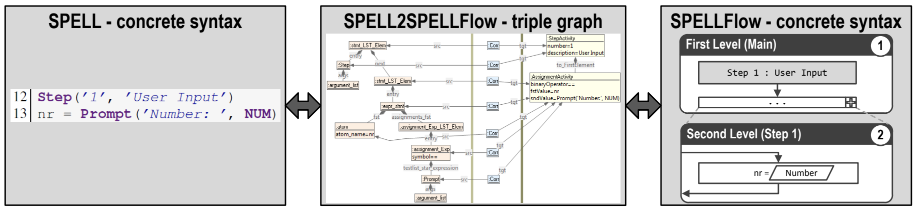
In future work, we will apply the model synchronisation framework based on TGGs [15]. The main idea is to propagate changes from one domain to the other by reusing the operational forward and backward translation rules.

$$\forall\, G'^S \in \mathcal{L}_S : \qquad\qquad \forall\, G'^T \in \mathcal{L}_T :$$

$$\begin{tikzcd}
G^S \arrow[r, leftrightarrow, "r"] \arrow[d, "a"'] \arrow[dr, phantom, "\searrow\!\!\searrow :\mathsf{fPpg}"] & G^T \arrow[d, dotted, "b"] & G^S \arrow[r, leftrightarrow, "r"] \arrow[d, "a"'] \arrow[dr, phantom, "\swarrow\!\!\swarrow :\mathsf{bPpg}"] & G^T \arrow[d, dotted, "b"] \\
G'^S \arrow[r, leftrightarrow, dotted, "r'"'] & G'^T & G'^S \arrow[r, leftrightarrow, dotted, "r'"'] & G'^T
\end{tikzcd}$$

Fig. 8. Synchronisation operations fPpg, bPpg

In Fig. 8, we illustrate the forward propagation operation (fPpg) which applies the model update *a* performed in the source model to the integrated model. On the left side of the figure, we illustrate the fPpg operation and on the right side, we illustrate the symmetric backward propagation operation bPpg. The forward operation consists of three steps: The *forward alignment* step constructs a new correspondence graph by deleting all correspondence elements which became invalid by the source model update *a*. The *deletion* operation creates a consistent integrated model in removing parts which became inconsistent by update *a*. The *forward addition* operation executes the operational forward rules, until all untranslated elements are translated. Due to the definition



```
Step('1', 'User Input')
nr = Prompt('Number: ', NUM)
```

Fig. 9. Excerpt: Summary of bidirectional engineering process

of this operation, the resulting integrated model is consistent. The bPpg operation is symmetric. In [15], we have shown for this model synchronisation framework that also correctness and completeness properties hold.

However, translating all updates performed synchronously in the source and target model to the integrated model can cause conflicts. In [7], an appropriate conflict resolution is discussed.

Due to the well-defined formal frameworks we use for defining the translation and later synchronisation, requirement 3 (concise and correct visual models and functional behaviour) is fulfilled. The model is concise, because we defined a hierarchical (multi-layer) view on the SPELL source code, especially the main layer of SPELLFlow provides an abstract view on the SPELL source code. The correctness and completeness w.r.t. correspondence patterns between SPELL and SPELLFlow is ensured by Theorem 8.2 in [15]. To show functional behaviour, we use the automatic critical pair analysis provided by HenshinTGG [13,14].

In Fig. 9, we show an overview of the whole bidirectional engineering process for an excerpt of our running example. The SPELL source code is parsed using Xtext yielding the SPELL ASG (left). This ASG is translated into an integrated model (middle) represented by a triple graph in using the set of FT rules. The target part is the SPELLFlow ASG, which is exported as an XMI file. This XMI file is imported into the SPELLFlowEditor which displays the desired SPELLFlow digram (right) in concrete syntax. To generate source code out of the visualisation, we will perform the same process in the backward direction and apply the set of BT rules for the translation. At a later stage, we will apply the presented synchronisation framework.

5 Related Work

TGGs were introduced in [21] and since then refined and extended by several works [19,12,22]. Many works focus on defining and preserving correctness properties and functional behaviour of TGGs [5,14]. Based on the delta-lenses framework [24], TGGs were extended by bidirectional model synchronisation frameworks [10,15]. These results will be reused in the presented approach.

In [17,1], a new type of TGGs was introduced: view triple graph grammars (VTGGs), in order to model domain-specific views of a source model. The authors present different views, e.g., domain-specific views or views presenting different abstraction layers, and describe an appropriate model transformation technique satisfying every type of view. VTGGs are very promising for the approach presented in this paper, though the definition of VTGGs as given in [1] is too restrictive for our approach and needs to be relaxed.

The Atlas Transformation Language (ATL) [18] is a widely-used framework for specifying model transformations in a declarative manner. However, the approach only supports the specification of unidirectional transformations and requires to specify each direction of bidirectional model transformations separately. Therefore, in contrast to TGGs, the approach does not allow to generate operational translation rules for forward and backward model transformations from one consistent specification.

Several works deal with model visualisation and visualisation languages. In [9], a general approach for defining a visualisation language and its simulation is given based on typed algebraic graph transformation. In [11], an overview of different software visualisation approaches and important properties for an appropriate visualisation are discussed. SPELLFlow matches most of these requirements. Koschke [20] presents a tool suite for software visualisation in reverse engineering. There, different visualisations are provided as additional information. In contrast, it is planned that SPELLFlow will replace the source code view completely. Both papers present surveys on software visualisation where the majority of interviewees (more than 80% in each survey) agree that software visualisation is at least important.

6 Conclusion

In this paper we introduced a new visual modelling language (*SPELLFlow*) for the visualisation of procedures written in the satellite control language *SPELL*. The requirements for the syntax and semantics of the visual language SPELLFlow were developed in cooperation with the industrial partner SES. We presented an approach for the automatic generation of SPELLFlow models from SPELL programs, and the generation of SPELL source code from SPELLFlow models. This bidirectional engineering approach is based on the formal framework of TGGs and supported by the tool HenshinTGG and a visual editor based on Eclipse GMF which we developed for SPELLFlow.

According to the requirements set up by SES, we will apply the synchronisation framework presented in [15] using HenshinTGG. Finally, we will evaluate our implementations regarding efficiency and usability in order to integrate the implementations in the daily work of satellite controllers and developers at SES.

Acknowledgements. Supported by the Fonds National de la Recherche, Luxembourg (3968135).

Fonds National de la Recherche Luxembourg

References

1. Anjorin, A., Rose, S., Deckwerth, F., Schürr, A.: Asymmetric delta lenses with view triple graph grammars (to appear). ECEASST pp. 1–15 (2013)
2. Eclipse Consortium: Eclipse Graphical Modeling Framework (GMF) (2013), `http://www.eclipse.org/modeling/gmp/`

3. Eclipse Consortium: Eclipse Modeling Framework (EMF), Version 2.8.3 (2013), `http://www.eclipse.org/emf`
4. The Eclipse Foundation: Xtext, Version 2.3.1 (2013), `http://www.eclipse.org/Xtext/`
5. Ehrig, H., Ermel, C., Hermann, F., Prange, U.: On-the-Fly Construction, Correctness and Completeness of Model Transformations based on Triple Graph Grammars. In: Proc. MODELS'09. LNCS, vol. 5795, pp. 241–255. Springer (2009)
6. Ehrig, H., Ehrig, K., Prange, U., Taentzer, G.: Fundamentals of Algebraic Graph Transformation. EATCS Monographs in Theor. Comp. Science, Springer (2006)
7. Ehrig, H., Ermel, C., Taentzer, G.: A Formal Resolution Strategy for Operation-Based Conflicts in Model Versioning Using Graph Modifications. In: Proc. FASE'11. LNCS, vol. 6603, pp. 202–216. Springer (2011)
8. Ermel, C., Hermann, F., Gall, J., Binanzer, D.: Visual Modeling and Analysis of EMF Model Transformations Based on Triple Graph Grammars. ECEASST 54, 1–14 (2012)
9. Ermel, C.: Simulation and animation of visual languages based on typed algebraic graph transformation. Ph.D. thesis, Technische Universität Berlin (2006)
10. Giese, H., Wagner, R.: From model transformation to incremental bidirectional model synchronization. SoSyM 8, 21–43 (2009)
11. Gracanin, D., Matkovic, K., Eltoweissy, M.: Software visualization. ISSE 1(2), 221–230 (2005)
12. Greenyer, J., Kindler, E.: Comparing relational model transformation technologies: implementing Query/View/Transformation with Triple Graph Grammars. SoSyM 9, 21–46 (2010)
13. EMF Henshin, Version 0.9.6 (2013), `http://www.eclipse.org/henshin/`
14. Hermann, F., Ehrig, H., Golas, U., Orejas, F.: Efficient Analysis and Execution of Correct and Complete Model Transformations Based on Triple Graph Grammars. In: Proc. MDI'10. pp. 22–31. MDI '10, ACM (2010)
15. Hermann, F., Ehrig, H., Orejas, F., Czarnecki, K., Diskin, Z., Xiong, Y., Gottmann, S., Engel, T.: Model synchronization based on triple graph grammars: correctness, completeness and invertibility. SoSyM pp. 1–29 (2013)
16. Hermann, F., Gottmann, S., Nachtigall, N., Braatz, B., Morelli, G., Pierre, A., Engel, T.: On an Automated Translation of Satellite Procedures Using Triple Graph Grammars. In: Proc. ICMT'13, LNCS, vol. 7909, pp. 50–51. Springer (2013)
17. Jakob, J., Königs, A., Schürr, A.: Non-materialized Model View Specification with Triple Graph Grammars. In: Graph Transformations, LNCS, vol. 4178, pp. 321–335. Springer (2006)
18. Jouault, F., Allilaire, F., Bézivin, J., Kurtev, I.: ATL: A model transformation tool. Science of Computer Programming 72, 31–39 (2008)
19. Kindler, E., Wagner, R.: Triple Graph Grammars: Concepts, Extensions, Implementations, and Application Scenarios. Tech. Rep. TR-ri-07-284, Department of Computer Science, University of Paderborn, Germany (2007)
20. Koschke, R.: Software Visualization for Reverse Engineering. In: Revised Lectures on Software Visualization, International Seminar. pp. 138–150. Springer (2002)
21. Schürr, A.: Specification of Graph Translators with Triple Graph Grammars. In: Tinhofer, G. (ed.) Proc. WG'94. LNCS, vol. 903, pp. 151–163. Springer (1994)
22. Schürr, A., Klar, F.: 15 years of triple graph grammars. In: Proc. ICGT'08. pp. 411–425. No. 5214 in LNCS, Springer (2008)
23. SES Engineering: SPELL - Satellite Procedure Execution Language & Library, Version 2.3.13 (2013), `http://code.google.com/p/spell-sat/`
24. Xiong, Y., Song, H., Hu, Z., Takeichi, M.: Supporting Parallel Updates with Bidirectional Model Transformations. In: Proc. ICMT'09. pp. 213–228. Springer (2009)