MATA: A Tool for Aspect-Oriented Modeling based on Graph Transformation

Jon Whittle and Praveen Jayaraman

Dept. of Information & Software Engineering,
George Mason University, Fairfax VA 22030
jwhittle@gmu.edu, praveenjayaraman@yahoo.com

Abstract. This paper describes MATA (Modeling Aspects Using a Transformation Approach), a UML aspect-oriented modeling tool that uses graph transformations to specify and compose aspects. Graph transformations provide a unified approach for aspect modeling. The methods presented here can be applied to any modeling language with a well-defined metamodel. This paper, however, focuses on UML class diagrams, sequence diagrams and state diagrams. MATA takes a different approach to aspect-oriented modeling since there are no explicit joinpoints. Rather, any model element can be a joinpoint and composition is a special case of model transformation. We illustrate MATA on structural and behavioral models for a cellphone example.

1. Introduction

Broadly speaking, there have been, to date, two approaches for modeling aspects in UML. The essence of the first approach is that two models are composed by identifying common elements and applying a generic merge algorithm. This is a symmetric form of aspect modeling and common elements are found according to some matching criteria, e.g., two classes with the same name are merged. Examples of this approach include Theme/UML [1] as well as work by France et al. [2]. The essence of the second approach is to reuse, at the modeling level, mechanisms for specifying and weaving aspects from aspect-oriented programming. There has been a significant amount of research, for example, that identifies a join point model for a modeling language and then uses the AspectJ advices of before, after, and around for weaving. Examples of this type include [3, 4].

These two kinds of approaches are not always sufficient. In the first approach, a merge algorithm based on general matching criteria will never be expressive enough to handle all model compositions. Matching by name, for example, may not work for state diagrams. Given two states with the same name, the states may need to be merged in one of a variety of ways depending on the application being modeled: (1) the two states represent the same thing, which implies making the states equal; (2) the two states represent orthogonal behaviors of the same object, which implies enclosing the states by a new orthogonal region; (3) one state is really a sub-mode of the other, which implies making one state a substate of the other; (4) the behaviors of the two states must be interleaved in a complex way, which implies weaving the actions and transitions in a very application-specific way to achieve the desired result.

Only the first of these can be accomplished based on merge-by-name. Furthermore, these are only four of many possible options so it is not generally sufficient to provide a number of pre-defined merge strategies.

In the second approach, specific elements are allowed to be defined as joinpoints and others are not. For example, in state diagrams, some approaches define events as joinpoints. Others, however, define states as joinpoints. One could even imagine more complex joinpoints, such as the pointcut of all orthogonal regions. (This pointcut might be used, for example, by an aspect that sequentializes parallel behaviors.) Defining only a subset of a model's elements as joinpoints seems to be overly restrictive. In addition, limiting advices to before, after, and around is also rather restrictive since it may be desired to weave behavior in parallel, or as a sub-behavior of a behavior in the base, for example.

MATA is an aspect-oriented modeling tool that tackles these problems by viewing aspect composition as a special case of model transformation. In MATA, composition of a base and aspect model is specified by a graph rule. Given a base model, M_B, crosscut by an aspect model, M_A, a MATA composition rule merges M_A and M_B to produce a composed model M_{AB}. The graph rule, r: LHS → RHS defines a pattern on the left-hand side (LHS). This pattern captures the set of pointcuts, i.e., the points in M_B where new model elements should be added. The right-hand side (RHS) defines the new elements to be added and specifies how they should be added to M_B. MATA graph rules are defined over the concrete syntax of the modeling language. This is in contrast to almost all known approaches to model transformation which typically define transformations at the meta-level, that is, over the abstract syntax of the modeling language. The restriction to concrete syntax is important for aspect modeling because a modeler is unlikely to have enough detailed knowledge of the UML metamodel to specify transformations over abstract syntax.

MATA currently supports composition for UML class, sequence and state diagrams. In principle, however, it is easy to extend MATA to other UML models (or, indeed, other modeling languages as long as a metamodel for the language exists) because the idea of using graph rules is broadly applicable. MATA makes no decisions on join point models, for example, that would limit the approach to specific diagram types. The focus in this paper is on the MATA tool and so the presentation will be by example. Technical details can be found elsewhere. [5] describes MATA's expressive pointcut mechanisms for state diagrams. [6] describes an application of MATA to composition in software product lines.

2. The MATA Language

MATA considers aspect composition as a special case of graph transformation. In general, a graph consists of a set of nodes and a set of edges. A typed graph is a graph in which each node and edge belongs to a type. Types are defined in a type graph. An attributed graph is a graph in which each node and edge may be labeled with attributes where each label is a (value, type) pair giving the value of the attribute and its type. The UML metamodel can naturally be represented as a type graph. Each metaclass becomes a node in the type graph and each meta-association becomes an

edge in the type graph. A UML model, therefore, can be represented as an instance of this type graph and a graph rule, defined over the type graph, will transform the UML model into another model that also conforms to the type graph (i.e., another UML model). In this way, existing graph theory can be used to transform UML models.

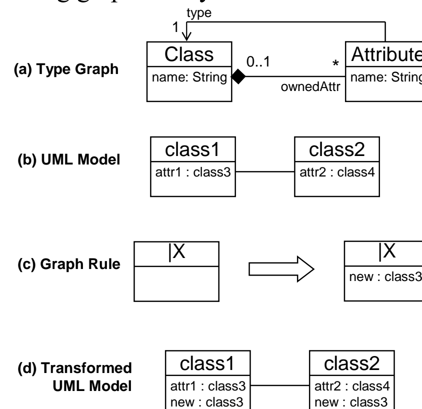


Figure 1: Graph Rules for UML

Figure 1 illustrates these concepts, using a simple example that adds an attribute to an existing class. (a) is a type graph defining a very simple metamodel for a subset of UML class diagrams. (b) is a UML class diagram that will undergo a transformation. The result of applying the rule in (c) to this class diagram is given in (d). Note that the left-hand side (LHS) of the rule in (c) defines a pattern over which the rule applies. Variables are prefixed by '|'. Hence, '|X' matches any class. The right-hand side (RHS) describes elements that should be added or removed. Any element that matches a variable on the LHS and does not appear on the RHS is removed. New elements on the RHS are added. The effect of applying the rule in (c) twice is to add two new attributes since '|X' matches against both class1 and class2.

MATA uses graph rules to define aspects. The pattern on the LHS essentially defines the pointcuts to match against. The RHS defines new model elements that should be added at these pointcuts (or it defines existing model elements that should be removed). There are two points of note about the graph rules in MATA.

Firstly, graph transformations are typically defined over the type graph. For UML models, this means that approaches such as FUJABA [7] and others define rules over UML's metaclasses. Whilst this is the most powerful approach, it is very inconvenient for a modeler because s/he must have a detailed knowledge of the UML metamodel. MATA instead represents graph rules in UML's concrete syntax, as given in the example in Figure 1(c), although minor extensions to the concrete syntax are introduced to allow for powerful pointcut expressions and variable expressions.

Secondly, it is also rather inconvenient to write graph rules using both a LHS and a RHS because elements that are unchanged must be repeated on both sides. Hence, MATA follows approaches such as VIATRA2 [8] in that the rule is given on one diagram. This is done by using a MATA profile which defines three new stereotypes:

- <<create>>, which can be applied to any model element, states that an element should be created by a graph rule.
- <<delete>>, which can be applied to any model element, states that an element should be removed by a graph rule.
- <<context>> is used with container elements to avoid elements being affected by <<create>> or <<delete>> (see below).

Figure 2 gives an example MATA graph rule to add parallel behavior in a sequence diagram. (a) is the MATA rule itself and (b) shows the application of the rule to a particular example. (a) has two parts to it—the pattern to match against and elements to add. In this case, the pattern is defined in two ways. Elements without a stereotype are matched against. In addition, any elements stereotyped as <<context>> are also matched against. This is because the semantics of <<create>> and <<delete>> are defined such that if these stereotypes apply to an element, then they also apply to all of the element's *immediate neighbors*. This is done simply to avoid having to write a <<create>> or <<delete>> stereotype for all neighbor elements. For example, in Figure 2(a), the immediate neighbors of the **par** fragment include any messages inside the fragment. Hence, <<create>> also applies to messages *r* and *s*. To avoid <<create>> being applied to *p*, it is marked with <<context>>. Therefore, the match defined in 2(a) is any pair of lifelines with a message *p* from one lifeline to the other. Note that not specifying an instance and type for a lifeline is equivalent to using two variables, */x:/X*. The effect of applying the rule in Figure 2(a) is to introduce a new **par** fragment around all instances of message *p*, and this new fragment will have messages *r* and *s* occur in parallel with *p*.

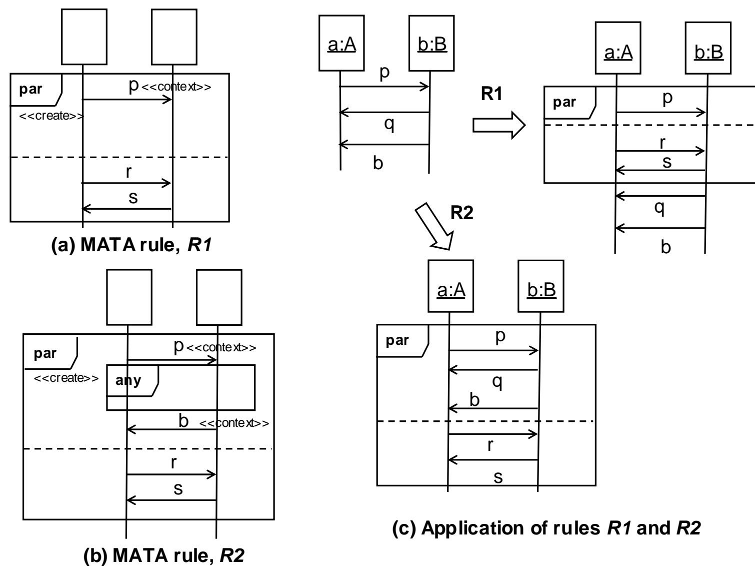


Figure 2: MATA Rules.

As stated earlier, the notion of immediate neighbor is used to reduce the number of elements that must be stereotyped with <<create>> or <<delete>>. This is purely a convenience for the modeler. The definition of immediate neighbor is specific to each

model element. For a class, its immediate neighbors are all its associations. For a state, its immediate neighbors are its transitions and its actions. For a combined fragment in a sequence diagram, its immediate neighbors are all model elements contained inside the fragment.

Figure 2 also shows an example how sequence pointcuts can be defined in MATA. A sequence pointcut is a match against any number of consecutive model elements—for example, a sequence of messages, or a sequence of transitions. Sequence pointcuts are handled naturally since the approach is based on patterns. In the example, the MATA combined fragment **any** is used to match against a sequence of messages of unknown length. So the rule R2 in Figure 2(b) will match any two lifelines with messages *p* and *b* with any number of messages between *p* and *b*. The result of applying the rule is shown in Figure 2(c). Note how the result is different than if rule R1 is applied. For R2, the pointcut is the sequence of messages *p, q, b*, and so, these messages all appear in the first operand of the **par** fragment.

These examples show that expressive pointcut expressions can be naturally specified using MATA. Figure 2 only shows examples for sequence diagrams. Similar rules for state diagrams can also be specified in the current implementation of MATA.

3. Extended Example

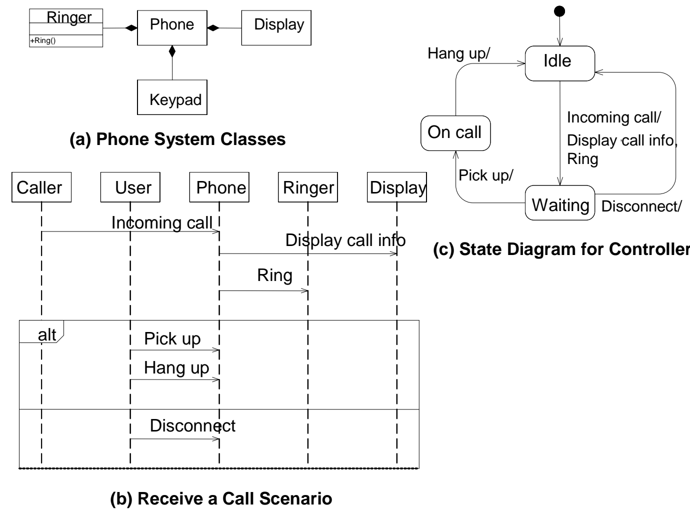
This section provides an extended example of MATA that includes both static and dynamic models. A cellphone application is used to illustrate how aspects can be specified and composed in MATA.

We will model three use cases for a simple cellphone—Receive a Call, Take a Message, and Notify Call Waiting. Following Jacobson and Ng's use case slice approach [4], we consider each use case as an aspect. We use MATA to maintain the use case separation throughout the modeling process. This avoids the traditional problem in OOAD of multiple dimensions of decomposition—that is, the requirements are decomposed in terms of use cases but the design models are captured in terms of objects. Maintaining multiple dimensions of decomposition can lead to difficulties when updating the requirements and design models.

We will consider Receive a Call to be the base use case since all cellphones will have this functionality. In contrast, Take a Message and Notify Call Waiting might not be available for all phones and so are modeled as aspects. (Even if they are available for all phones, it is still useful to model them as aspects since this will maintain a clear separation of each use case from other use cases.) The base use case is modeled in UML whereas the aspect use cases are modeled as MATA models, that is, as increments of the base models. Note that the models for the aspect use cases refer only to those elements in the base that are needed for the modifications to take place. Also, each aspect is modeled independently from the other aspects, that is, it is modeled only in terms of the base (although this is not a limitation of MATA).

Figure 3 shows (simplified) static and dynamic models for the base use case, Receive a Call. The phone contains a ringer, a phone component, a display unit and a keypad. Upon receiving an incoming call, the phone notifies the user by displaying the caller information on the display unit and sending a ring message to the ringer.

The user is allowed to either accept the call (then hang up later) or not accept (i.e., disconnect) the call.



(a) Phone System Classes

(b) Receive a Call Scenario

(c) State Diagram for Controller

Figure 3: Models for the Base Use Case.

Figure 4 gives the behavior models for the two aspects, Take a Message and Notify Call Waiting. 4(a) is a sequence diagram for Take a Message. If the phone rings for a specified amount of time, the call goes to a messaging system. In MATA, this is specified by creating a new **alt** fragment since forwarding to voice mail is an alternative scenario to the case where the callee accepts the call. Note that an **any** fragment is used to match against all messages coming after Ring in the base. This is needed since once a message is taken, the user should not be able to pick up the call or disconnect it. Hence, the **alt** fragment must be wrapped around all messages in the base concerned with call pick up or disconnect. A <<context>> stereotype is used on the **any** fragment to specify that it is to be matched against in the base.

In Figure 4(b), the aspect rule matches any two states which have a transition between them with an event named Incoming call. The effect of the aspect is to add an additional transition capturing the voicemail behavior. When this rule is applied, the two states will match against Idle and Waiting in Figure 3(c). The effect is to add a transition from Waiting back to Idle.

Figure 4(c) introduces messages for putting an incoming call on hold when a call is already underway. These new messages are only relevant when a call is taking place, that is, in between messages Pick Up and Hang Up in the base. Hence, the **loop** fragment is marked with a <<create>> stereotype and this fragment is inserted in between Pick Up and Hang Up. Note that, in this case, it would be sufficient to leave out the Hang Up message in 4(c), which, in effect, would insert the new behavior *after* Pick Up. However, we include Hang Up because there may eventually be other occurrences of Pick Up which should not be affected by the aspect.

Figure 4(d) introduces a new state, Waiting for hold prompt, into the base to capture the new behavior for the call waiting use case. Note that the two transitions in

4(d) implicitly have <<create>> stereotypes because they are immediate neighbors of the newly created state.

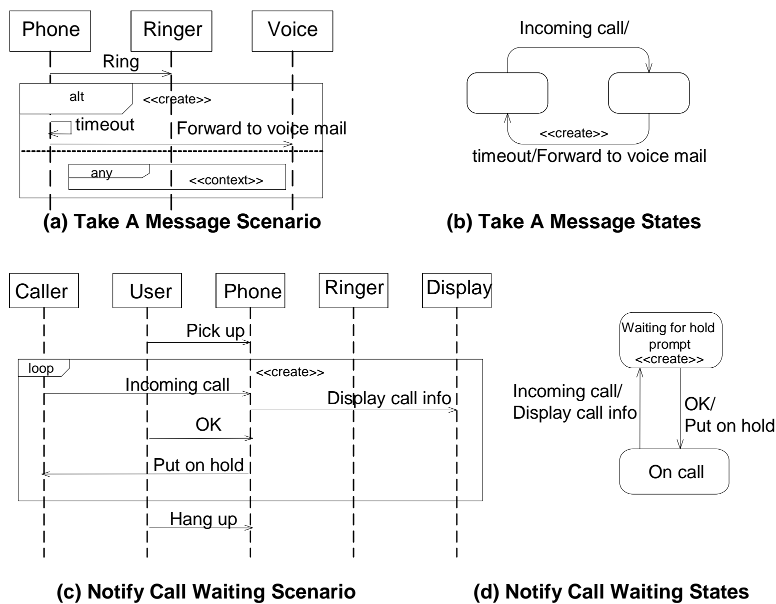


Figure 4: Aspect Models for Take a Message and Notify Call Waiting

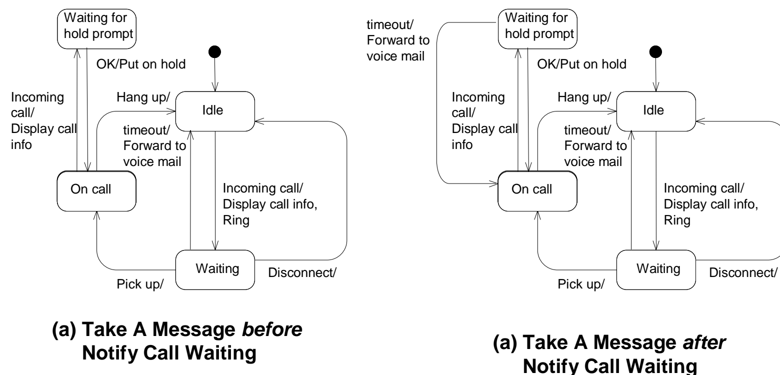
3.1. Interactions between Aspects

MATA comes with some support for automatically detecting interactions between aspects. Since aspects in MATA are graph rules, the technique of critical pair analysis (CPA) [9] can be used to detect dependencies and conflicts between rules. CPA examines rules in a pair wise fashion and declares a dependency if one rule requires a model element introduced by another rule. A conflict is declared if one rule modifies the base in such a way that another rule can no longer be applied. Conflicts and dependencies usually imply that the rules should be applied in a particular order since the result may be different depending on the order. A conflict may also mean that two rules that should both be applied cannot be, and therefore, the rules themselves should be modified. In any case, CPA provides a degree of automatic feedback to the modeler that both provides assistance in ordering rules and provides some assurance that the composition is correct.

Considering the example again, we can see that there is a dependency between the two state diagram rules for Take a Message and Notify Call Waiting. This dependency arises because Notify Call Waiting creates a transition with event Incoming Call (Figure 4(d)) whereas Take a Message matches against the event Incoming Call (Figure 4(b)). Hence, if Take a Message is applied to the base before

Notify Call Waiting then any incoming call that is received during an existing call cannot be sent to voicemail. Figure 5 gives the results of composing the two aspects with the base in either order. In 5(a), Take a Message is applied to the base before Notify Call Waiting. In 5(b), it is applied after. The difference is that there is an extra transition from Waiting for hold prompt to On call in 5(b) which captures the fact that an incoming call may be sent to voice mail even when there is currently an active call taking place. The difference in the composed state diagrams arises because the rule for Notify Call Waiting introduces a new transition with event Incoming call. Hence, when the Take a Message rule is applied in 5(b), there are two transitions with event Incoming call and so the rule applies twice.

MATA detects these kinds of dependencies automatically. Ultimately, the modeler must decide which order is the correct one, but MATA can at least provide some assistance in flagging cases that must be considered more carefully. If there are no conflicts or dependencies found by CPA, then the rules can be applied in any order. CPA is particularly important when aspects are reused in a different context than originally intended since new conflicts and dependencies may then arise inadvertently.



(a) Take A Message *before* Notify Call Waiting

(a) Take A Message *after* Notify Call Waiting

Figure 5: Base and Aspect State Diagrams Composed

4. Tool Implementation

This section describes the implementation of MATA. MATA is designed as a vendor-independent tool but currently works on top of IBM's Rational Software Modeler (RSM). Each aspect is modeled as a package. Within this package, the class diagrams, sequence diagrams and state diagrams for the aspect are kept. Users may select a subset of the aspects and the tool generates the composed model for all of these aspects and the base. The user may also define an ordering of aspect composition in the case that one aspect needs to be composed before another. If an ordering is not specified, the tool selects an order nondeterministically.

Since MATA uses graph transformations as the underlying theory, it relies on an existing graph rule execution tool to apply graph rules. The graph rule execution tool

used is AGG [10]. MATA converts a UML base model, captured as an instance of the UML2 metamodel by RSM, into an instance of a type graph, where the type graph represents a simplified form of the UML2 metamodel. MATA composition rules are converted into AGG graph rules and are executed on the base graph automatically. The results are converted back into a UML2 compliant model and are displayed in RSM. Critical pair analysis is done by AGG and the results are converted into RSM so that detected dependencies and conflicts can be understood by the user.

In principle, MATA could use any existing graph rule execution tool (e.g., VIATRA2 or FUJABA) as its underlying engine, but AGG was chosen because of its support for critical pair analysis. Although built on top of an existing engine, MATA provides some unique features that make it very suitable for aspect modeling and composition, namely: (1) graph rules are defined graphically using the concrete syntax of UML rather than using metaclasses; (2) MATA supports sequence pointcuts, that is, an aspect may match against a sequence of messages or a sequence of transitions. This is supported directly in the MATA rule syntax; (3) the stereotype <<context>> is unique to MATA; (4) dependencies and conflicts between aspects can be detected automatically using critical pair analysis.

5. Related Work

Most research on aspect-oriented modeling has focused on static models (e.g., [1, 2]). Some work has addressed behavioral models (e.g., [3, 11-13]). This paper is different from previous work in three key respects. Firstly, there are no explicit join points but instead composition is viewed as a special case of model transformation. Secondly, graph transformations provide a formal foundation for aspect composition. Finally, there is support for statically analyzing aspect interactions via critical pair analysis.

Related work that is closest to ours includes join point designation diagrams (JPDDs) [14]. JPDDs are similar to defining patterns using graph rules. The advantage of using graph rules is the existence of formal analysis techniques. In addition, JPDDs focus on defining join points and are not so much concerned with composition. MATA provides a full composition tool in which very expressive composition relationships can be specified. This is not possible with JPDDs.

More generally, model composition has been addressed outside of the AOSD community. In particular, [15] investigates how to merge state machines using composition relationships and category theory. This is similar in many respects to our work but has a different goal in that it addresses how to reconcile state machines produced by different development teams. Model composition is also important in feature-based approaches to software product lines. Recent work by Lopez has addressed how to adapt feature-oriented programming to UML models.

6. Conclusion

This paper presented a new approach for modeling and composing aspect models written in UML. Base models are specified using UML and aspect models are

specified using UML and MATA stereotypes that define where and how aspect model elements should be added to the base. Tool support for MATA is built on top of IBM's Rational Software Modeler. A prototype has been implemented and a number of case studies have been modeled with this prototype. MATA uses the graph rule execution tool AGG as a back-end for executing graph transformations and performing critical pair analysis.

1. Clarke, S., Baniassad, E.: Aspect-Oriented Analysis and Design: The Theme Approach. Addison Wesley (2005)
2. France, R., Ray, I., Georg, G., Ghosh, S.: Aspect-oriented approach to early design modeling. IEE Proceedings - Software **151** (2004) 173-186
3. Cottenier, T., van den Berg, A., Elrad, T.: Motorola WEAVR: Model Weaving in a Large Industrial Context. Aspect-Oriented Software Development (AOSD), Vancouver, Canada (2007)
4. Jacobson, I., Ng, P.-W.: Aspect Oriented Software Development with Use Cases. Addison-Wesley Professional (2004)
5. Whittle, J., Moreira, A., Araújo, J., Rabbi, R., Jayaraman, P., Elkhodary, A.: An Expressive Aspect Composition Language for UML State Diagrams. In: Engels, G., Opdyke, B., Weil, F. (eds.): International Conference on Model Driven Engineering, Languages and Systems (MODELS). Springer, Nashville, TN (2007)
6. Jayaraman, P., Whittle, J., Elkhodary, A., Gomaa, H.: Model Composition in Product Lines and Feature Interaction Detection using Critical Pair Analysis. In: Engels, G., Opdyke, B., Weil, F. (eds.): International Conference on Model Driven Engineering, Languages and Systems (MODELS). Springer, Nashville, TN (2007)
7. Nickel, U., Niere, J., Zuendorf, A.: The FUJABA Environment. International Conference on Software Engineering, Limerick, Ireland (2000) 742-745
8. Csertan, G., Huszerl, G., Majzik, I., Pap, Z., Pataricza, A., Varro, D.: VIATRA - visual automated transformations for formal verification and validation of UML models. Automated Software Engineering, 2002 (ASE), Edinburgh, UK (2002) 267
9. Heckel, R., Küster, J., Taentzer, G.: Confluence of Typed Attributed Graph Transformation Systems. 1st Int. Conference on Graph Transformation (ICGT 02) (2002)
10. Taentzer, G.: AGG: A Graph Transformation Environment for Modeling and Validation of Software. In: Pfaltz, J., Nagl, M., Boehlen, B. (eds.): AGTIVE: Applications of Graph Transformations with Industrial Relevance, Vol. 3062 LNCS, Charlottesville, VA (2003) 446-453
11. Klein, J., Helouet, L., Jézéquel, J.-M.: Semantic-Based Weaving of Scenarios. Aspect-Oriented Software Development (AOSD), Vancouver, Canada (2006) 27-38
12. Araújo, J., Whittle, J., Kim, D.-K.: Modeling and Composing Scenario-Based Requirements with Aspects. International Conference on Requirements Engineering (RE), Kyoto, Japan (2004) 58-67
13. Mahoney, M., Elrad, T.: Generating Code from Scenario and State Based Models to Address Crosscutting Concerns. Sixth International Workshop on Scenarios and State Machines (2007)
14. Stein, D., Hanenberg, S., Unland, R.: Expressing Different Conceptual Models of Join Point Selections in Aspect-Oriented Design. Aspect-Oriented Software Development (AOSD), Bonn, Germany (2006) 15-26
15. Nejati, S., Sabetzadeh, M., Chechik, M., Easterbrook, S., Zave, P.: Matching and Merging of Statecharts Specifications. International Conference on Software Engineering (2007) 54-64