

Specifying Model-Level Aspects with Graph Transformations

Jon Whittle

Information & Software Engineering
George Mason University
4400 University Drive, Fairfax VA
+1-703-9931677

jwhittle@ise.gmu.edu

João Araújo

Department of Informatics, FCT
Universidade Nova de Lisboa
2829-516 Caparica, Portugal
+351 21 294-8536

ja@di.fct.unl.pt

Ana Moreira

Department of Informatics, FCT
Universidade Nova de Lisboa
2829-516 Caparica, Portugal
+351 21 294-8536

amm@di.fct.unl.pt

ABSTRACT

This paper presents a new method for representing and composing aspect models. The method is based on the use of a UML-based aspect modeling language to precisely and graphically specify model-level aspects and the use of graph transformations to define how aspects should be composed. The result is a method for representing and composing aspect-oriented models that is both scalable and expressive. The work is validated on an air traffic control example based on a NASA application.

Categories and Subject Descriptors

D.2.1 [Software]: Requirements/Specifications – *languages, methodologies*. D.2.2 [Software]: Design Tools and Techniques – *object-oriented design methods, state diagrams*

1. INTRODUCTION

In this paper, a general technique for representing and composing aspects at any level of software abstraction is presented. It is important to be able to represent aspects at any stage of software modeling because aspects naturally occur during requirements [12], analysis [21] and design [3]. The modularization of aspects during software modeling leads to a clear separation of concerns and hence to more maintainable, understandable and analyzable models. However, in order to fully understand or analyze a model with aspects, the aspect modules must be composed with the base models so that any conflicts, ambiguities or omissions can be identified. Therefore, specification mechanisms for aspects at the modeling level should be complemented with composition mechanisms that weave the aspect models into the base models. Research on such mechanisms to-date suffers from one of two major problems – lack of expressiveness and lack of scalability. Compositions at the modeling level can be extremely rich in nature. Existing research does not offer the level of expressiveness needed for specifying these rich compositions. A high-degree of expressiveness, however, can lead to scalability problems because a large effort is required by the modeler to

specify the compositions. The technique presented in this paper addresses both the issues of scalability and expressiveness. The result is a practical technique for defining and composing aspect-oriented models in a way that is compatible with modeling best practices.

The technique uses two key technologies: the role-based metamodeling language (RBML) [10] and graph transformations [9, 14]. RBML (in fact, an extension of it) gives a precise, graphical means of specifying a model-level aspect in a way that is consistent with UML [18], the most common software modeling language. RBML has already been applied to model the structural parts of security aspects [13] and to model behavioral UML aspects [21]. A drawback of these approaches is that they do not scale well since a lot of effort is required to specify how RBML aspects crosscut core models. We will show how graph transformations can be used to reduce the level of effort. Graph transformations have been applied to many problems in software engineering including the merging of different system perspectives [8] but it has not been specifically addressed how to apply them, in a general way, to handle aspects at any stage of UML modeling. This paper combines RBML and graph transformations to achieve the following two aims: (1) general-purpose, UML-based aspect modeling and composition at any level of abstraction; (2) scalability of aspect composition.

The paper illustrates the approach with an air traffic control example based closely on an existing application developed at NASA [4]. Section 2 defines the notion of a model-level aspect and analyzes shortcomings of the closest related research, namely that of France et al. [5] and Whittle and Araújo [21]. Section 3 presents a new method for composing aspect models. Section 4 validates the approach and related work and conclusions follow.

2. WHAT IS A MODEL-LEVEL ASPECT?

We define an aspect-oriented *model* to be a model that crosscuts other models *at the same level of abstraction*. The last part of the sentence is important. It means, for example, that a requirements model is an aspect if it crosscuts other requirements models (or simply, a requirements artifact is an aspect if it cuts across other requirements artifacts); a design model is an aspect if it crosscuts other design models. In particular, a use case (defining a set of requirements) is not necessarily an aspect. Although a use case always cuts across multiple implementation modules, it is only considered to be an aspect if it cuts across other use cases.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

Submitted EA @ ICSE 2006

In this paper, we restrict the definition of an aspect-oriented model further and say that a model is an aspect only if it crosscuts other models written from the same perspective (in UML, the same kind of diagrams). For example, a model showing global component interactions does not, according to our definition, crosscut a model showing internal component behavior. Although the models might be defined at the same level of abstraction, they are written from different perspectives – a global and a local perspective. This is not considered, in this paper, to be crosscutting. In terms of UML, this means that we are only interested in crosscuts defined over diagrams of the same type.

2.1 Representing Aspects in RBML

We represent aspect-oriented models in the Role-Based Metamodeling Language (RBML) first introduced by [10] and later complemented by France et al. [5]. A RBML model is a specialization of the UML metamodel wherein each element in a RBML model is a role. A role is a restriction of a UML metaclass with (optional) properties that any element playing the role must possess. Since RBML specializes UML, each UML diagram has a corresponding RBML diagram in which model elements are roles. For example, a RBML state diagram has state roles and transition roles. Each state role represents a generic state that can be made concrete by assigning a concrete model element to play the role. Only model elements that satisfy the properties of the role can play a role. This means, for example, that transitions cannot play the role of a state. In essence, a RBML model defines a generic model that can be instantiated in many ways by assigning elements to all of its roles. A UML model is said to *conform* to a RBML model if there is a valid assignment of elements in the UML model to the roles of the RBML model.

RBML has been used to formalize design patterns [10] and to represent model-level aspects [7]. This was extended to behavioral aspects (i.e., aspect state diagrams and aspect sequence diagrams) in [21] and [1]. In the original definition [5], all RBML model elements must be roles, that is, they are meta-level elements. It was noted in [21] that for representing aspects, it is useful to allow object-level elements in RBML as well. The result is an extension to RBML, eRBML, in which an element in eRBML may be either a meta-level element (that is, a role) or an object-level element.

Figure 1 is a simple example of an aspect sequence diagram in eRBML. The aspect says that whenever a message playing the `|failure_ack` role is sent by the `|Client` role to the `|Server` role, a record is made of the message and the client is closed. Note that the aspect in eRBML mixes meta-level roles and object-level modeling elements. This mixing of meta-level and object-level syntax is similar to meta-programming techniques such as the LISP quote/anti-quote. It is preferable to allow object-level modeling elements such as `Status` in the aspect because the object of type `Status` remains unchanged at every joinpoint of the aspect. Only the role elements change for each application of the aspect.

2.1.1 Instantiation

eRBML models can be instantiated by assigning concrete model elements to play the roles. Instantiation is defined by a one-to-many mapping from roles to model elements. An instantiated eRBML model is an eRBML model with all its role elements mapped to concrete model elements. In terms of aspects, an

eRBML aspect model must be instantiated before it can be composed with a base model. The instantiations define what the aspect should look like in the context of a particular application – i.e., they specialize an aspect to a context. Figure 2 is an example base model which the aspect in Figure 1 crosscuts. The sequence diagram in Figure 2 is taken from our case study and will be presented fully in Section 4. For now, it is enough to know that the diagram shows a central controller, `CM`, accessing data and sending it to a client. The controller additionally stores information about the transaction and finally enables a GUI panel. The diagram in Figure 2 does not take care of failure handling.

Instantiation prepares the aspect for composition with the base. In this example, the following instantiations are specified by the modeler: `|Client` \rightarrow `WAClient`, `|Server` \rightarrow `CM` and `|failure_ack` \rightarrow `cannot_accept_wthr_data`. Figure 3 shows the result of composing the instantiated aspect with the base model. The messages to deal with failure have been inserted after `Send_wthr_data(x)` as an alternative execution path using a UML2.0 `alt` interaction fragment. The rationale for deciding on an `alt` fragment and the positioning of the aspect messages is deliberately left unexplained here. Current methods do not provide good ways of specifying such information – see section 2.2 for details.

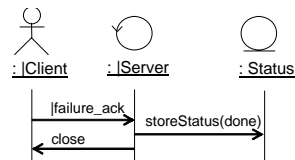


Figure 1: A simple failure-handling aspect.

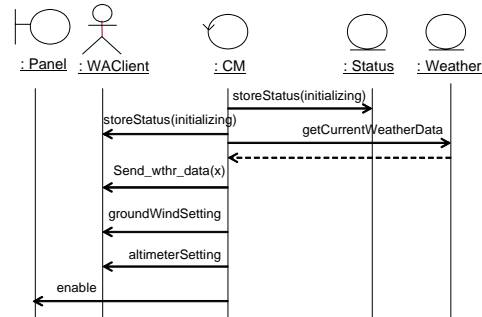


Figure 2: A base sequence diagram for data transferal.

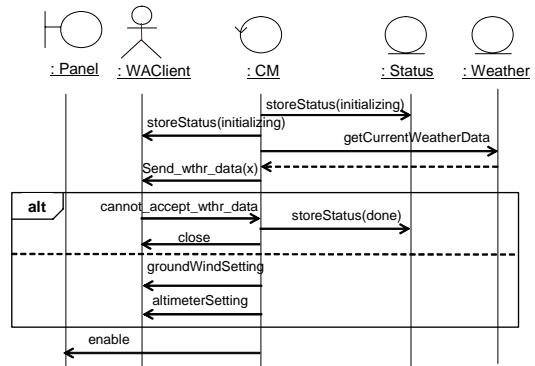


Figure 3: Composition of Figure 1 and Figure 2.

2.1.2 Conformance

A UML model is said to *conform* to an eRBML model if there exists an instantiation of the eRBML model such that all model elements of the instantiated eRBML model are present in the UML model and all constraints in the eRBML model hold in the UML model. The constraints include message ordering in the case of eRBML sequence diagrams, transition ordering in the case of eRBML state machines etc., as well as any additional properties specified for eRBML roles. As an example, Figure 3 conforms to the eRBML model in Figure 1 because the message ordering from Figure 1 is present. Note that there may be other model elements present in the UML model but the UML model still conforms as long as the original message ordering from the eRBML model is maintained. In particular, there could be additional messages between *storeStatus(done)* and *close*, for example, although this is not the case in this example. Conformance is a type of refinement. In general, it is a much more complex notion than might be suggested by the examples in this paper - *Client* in Figure 1, for example, could be instantiated to multiple actors at the UML model level. We refer the interested reader to [9] for more details on conformance.

2.2 Aspect Composition

eRBML is a precise way to specify model-level aspects and can be used to describe aspects for any UML diagram. Just as it is important to represent model aspects in a modular fashion, it is also important to compose model aspects with base models. In this section, we compare the composition approaches of France et al. [6] and Whittle & Araújo [21] to illustrate shortcomings in existing research. France et al. use templates to represent aspects which are a restricted form of eRBML. In this paper, we ignore this difference and explain the composition approach of France et al. as if it used eRBML. This is to make the comparison easier. In both approaches, an eRBML aspect is instantiated before it is composed with a base model. The approaches differ in how composition is specified.

The example of aspect composition given so far (Figure 3) is a rather simple one. Even in this simple case, however, there are many choices in how the composition should be done. In Figure 3, the failure handling messages were considered to be an alternative execution sequence and were inserted after the *Send_wthr_data(x)* message. Although this is a common composition strategy, it will not suffice in many cases. Consider, for example, how to specify the fact that the aspect messages should be interleaved with the base model messages *groundWindSetting*, *altimeterSetting* and *enable*. Or to specify that the aspect messages define a sequence executed in parallel with the base model messages. In general, there are many possible ways that composition could occur. The challenge is to find a technique for specifying composition that admits a high degree of expressiveness, but does not require a high degree of modeling effort from the user. We briefly compare the two approaches to composition by France et al. and Whittle & Araújo to see how they perform on expressiveness versus scalability.

2.2.1 Composition Directives (France et al.)

The approach by France et al. is to allow the modeler to specify composition directives that tailor the composition algorithm. These directives allow the user to explicitly specify whether the

aspect messages should be interleaved with the base, should be an alternative or should run in parallel with it etc. The directives take the form of “add”, “delete”, “replace” and “move” statements that modify the model elements in the composed model. The base and instantiated aspect model are first merged by merging model elements with the same name. Hence, if an aspect model has a role instantiated to an element with the same name as an element in the base model, then the two elements are merged. After merging, the directives are applied to tailor the exact form of composition.

This is a very expressive way to tailor composition but involves a lot of work from the modeler. It amounts to a manual composition process because the modeler has to apply a directive to each and every model element in the aspect and possibly model elements in the base as well. For example, to interleave the messages of the aspect of Figure 1 with the base model in Figure 2, the modeler has to instantiate *|failure_ack* to *cannot_accept_wthr_data* and then give a “move” directive for each aspect message, e.g. *move cannot_accept_wthr_data* to after *Send_wthr_data(x)*. This is a very low-level way to specify aspect compositions and the effort involved quickly becomes excessive since the directives must be given for each aspect and for each base model crosscut by an aspect. Clearly, the approach will not scale.

2.2.2 Composition Operators (Whittle & Araújo)

The approach by Whittle & Araújo is at a higher level of abstraction. Instead of composition directives, the modeler specifies composition operators that are high-level composition strategies. For example, [21] defines three composition operators for sequence diagrams – OR, AND, and IN. OR makes the aspect messages an alternative sequence, AND interleaves the aspect messages with the base model messages, and IN inserts the aspect messages in the base sequence. The OR operator produces the result in Figure 3. AND would have a **par** fragment instead of **alt**. IN would insert the aspect messages as a sequence after *Send_wthr_data(x)*.

Composition operators offer a high-level way of composing aspect models. The approach is more scalable than that of France et al. because the modeler need not explicitly specify a composition strategy for each model element in the aspect. On the other hand, the approach is less expressive because there are only a pre-defined set of composition operators which may handle many but not all cases of composition.

Furthermore, both approaches require the modeler to specify the instantiations of the aspect and these instantiations will in general be different for each base model crosscut by the aspect. For large problems, the amount of effort involved in specifying instantiations may become too great.

3. A NEW METHOD OF COMPOSITION

The composition techniques presented in Section 2 suffer from a lack of scalability. The modeler must give a set of role instantiations for each aspect and for each base model that each aspect crosscuts. For large models with many aspects, there are a large number of instantiations to supply. Moreover, maintenance of the models becomes problematic because the instantiations are given in a non-graphical, low-level format that can be somewhat time-consuming to read. On the other hand, the representation method provides a clean separation of aspects and the base

models. This section presents a new way of representing and composing model aspects in a way that maintains aspect modularity but is also scalable. It is based on representing composition as a graph transformation rule in which the left-hand side of the rule captures the points where the aspect should be applied and the right-hand side captures the aspect itself.

3.1 Aspects as Graph Transformations

A graph transformation [14] is a rule $r: L \rightarrow R$ from a left-hand side (LHS) graph L to a right-hand side (RHS) graph R . The process of applying r to a graph G involves finding a graph monomorphism, h , from L to G and replacing $h(L)$ in G with $h(R)$. To avoid “dangling edges” – i.e., edges with a missing source or target node – $h(R)$ must be pasted into G in such a way that all edges connected to a removed node in $h(L)$ are reconnected to a replacement node in $h(R)$.

Any UML diagram can be represented as a graph because it is defined by the UML metamodel which is a graph where the nodes are meta-classes and the edges are meta-relationships. Hence, transformations over UML models can be given as graph transformations. In particular, we view composition of an aspect-oriented model with a base UML model as a graph transformation in which the LHS and RHS are eRBML models. The LHS defines the points where the aspect should be applied and the RHS defines the crosscutting structure/behavior that should be inserted at those points.

We give an example of how the aspect from Figure 1 can be given as a graph transformation – see Figure 4. There are two parts to the definition of the aspect. The first defines the aspect itself – this is the same as given in Figure 1. The second defines the composition strategy – given in Figure 4 as a graph transformation. The effect of applying the transformation is that messages for dealing with failure will be inserted as an alternative sequence after all instances of `|send_data`, a message sent from `|Server` to `|Client`. This approach to defining aspect composition addresses the issues of expressiveness and scalability in the following ways.

There are a number of points to make about Figure 4. Firstly, the method maintains a complete separation of the aspect and its composition strategy – Figure 1 defines the aspect; Figure 4 is the strategy. This promotes reuse of aspects and application of the same aspect but with a different composition strategy. Section 4 will give an even more convincing example of this by showing how a generic two-phase commit protocol can be reused as an aspect. Secondly, the technique is a fully expressive way of defining composition strategies – Figure 4 illustrates just one strategy but others could easily be defined by modifying the LHS or RHS. Thirdly, the method reduces the number of instantiations that must be provided. In this example, only one instantiation must be provided by the modeler – for `|failure_ack`. All other roles can be instantiated automatically by graph matching against a base model – the LHS of the graph transformation is matched against the base model thus instantiating `|Client`, `|Server`, `|send_data` and `|Other` automatically; only `|failure_ack` remains to be instantiated. In this example, the LHS of Figure 4 matches the base model (Figure 2). The match results in the following instantiations being made automatically: `|send_data` \rightarrow `Send_wthr_data(x)`, `|Client` \rightarrow `WAClient`, `|Server` \rightarrow `CM` and `|Other` matches against all messages following `Send_wthr_data(x)`. Note that a UML2.0 **ref** fragment is used to

define a placeholder for a sequence of messages in the base. This is an easy way to match against a message sequence whose position in the composed model can then be specified exactly on the RHS of the transformation.

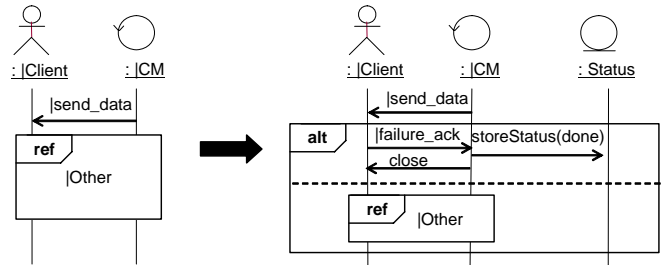


Figure 4: Failure handling aspect composition as a graph transformation.

We must extend the definition of graph transformation when dealing with eRBML. When matching against the LHS of the transformation rule, instantiations for the role elements must be discovered. We therefore modify the definition such that a graph transformation applies to a UML model if and only if the LHS of the transformation can be graph matched *modulo conformance*. This notion is defined below. Also below, we analyze our new composition method with respect to expressiveness and scalability.

3.1.1 Expressiveness

The RHS of the graph transformation rule defines the manner in which the aspect crosscuts a base model. Because the RHS is a model itself, there is complete expressiveness in how the crosscutting is defined. The aspect messages, for example, can be defined as alternatives to a base message or messages, as interleaved with base messages, as occurring in parallel with base messages, or any combination. The composition operators defined in Section 2.2.2 can be defined as special cases but the graph transformation allows any combination of these operators to be specified, or indeed for new operators to be specified. The composition directives in Section 2.2.1 are subsumed by the graph transformation approach because there is no longer any need to tailor the aspect composition algorithm to add, delete, or remove elements – these modifications are, in essence, defined explicitly in the RHS of the transformation.

3.1.2 Scalability

The biggest barrier to scalability of aspect approaches based on RBML is that the modeler must instantiate the role elements for each base model crosscut by the aspect. Graph transformations reduce this effort because instantiating the role elements can be automated to some extent. Instantiation amounts to finding a base model over which the graph transformation can be applied – i.e., finding a match for the LHS of the transformation rule. We use *matching modulo conformance* when applying an aspect – that is, an eRBML model, R , matches a UML model, U , modulo conformance if and only if there is an instantiation of the role elements, θ , such that U conforms to $\theta(R)$. Consider again the example in Figure 4. In this case, the base model in Figure 2 matches the LHS of the aspect transformation modulo conformance because the roles in the LHS can be instantiated in a way such that the base model conforms to the LHS – `|send_data` is instantiated to `Send_wthr_data(x)` and `|Other` is instantiated to the

sequence of messages following *Send_wthr_data(x)*. Note that matching modulo conformance looks for a maximal match if the LHS contains a **ref** fragment. In other words, all messages following *Send_wthr_data(x)* are matched against *|Other*, not any subsequence of those messages.

4. CASE STUDY

CTAS (Center-TRACON Automation System) [4] is a component-based, distributed system whose goal is to maximize throughput of aircraft through an airport. One of the most crucial aspects of CTAS is a weather subsystem that broadcasts new weather data to all relevant processes and ensures integrity of this weather data. The CM (communications manager) is responsible for distributing new weather-related data to all CTAS clients that need it (weather-aware clients). New data arrives as a result of new forecasts or new weather data sources, and can be triggered automatically or manually. A weather update procedure is invoked by CM every time new data arrives or a new weather-aware client connects to the system.

Data integrity is clearly an important issue in the design of this system that will affect several of its components. All weather-aware clients rely on a model of the current weather data and must use the same model at all times. The original design deals with data integrity using a two-phase commit protocol. However, in the design, the details of this protocol are intertwined with the core functionality of CM. In what follows, we show how to modularize the two-phase commit protocol as an aspect and show how to compose this aspect with the core functionality. This results in a more readable design in which changes to the data integrity design are localized and therefore easier to make. We focus in this paper on the dynamic aspects of the design and model the design using UML sequence diagrams.

There are two situations in which weather data integrity becomes important. Firstly, when a new weather-aware client connects, current weather data must be sent to it. Secondly, when the user or system invokes an update of the weather data, the new data must be broadcast to all weather-aware clients. In both cases, the requirements state that the two-phase commit protocol should be applied. Figures 5 and 6 show the two base sequence diagram models corresponding to the connection of a new client and a weather update, respectively. The two core functionalities are similar but not the same – different information is stored in each case, and in the second case, there are additional messages for data persistency¹.

Figure 6 introduces some additional notations not currently part of UML2.0. `<<multiobject>>` is used to denote the fact that there are multiple participants of a given type. `{all}` means that a message is sent to or received from all of those participants. `{exists}` (not shown in the figure) means that a message is sent to or received from at least one of those participants.

¹ It would have been possible to also treat persistency as an aspect but this is not done in this paper.

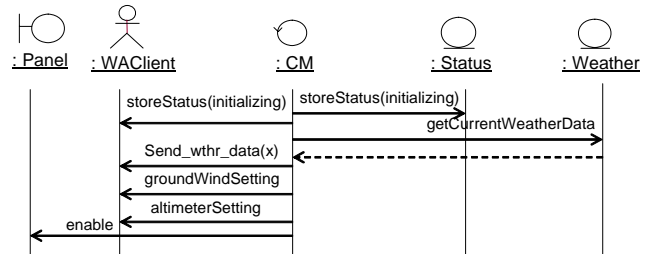


Figure 5: New weather-aware client connects. Update clients.

In Figure 6, all messages concerning communication between CM and its clients are sent to all clients. Both diagrams take care of broadcasting weather data by simply sending (one or more) *Send_wthr_data* messages. The two-phase commit protocol is not modeled as part of the core functionality. Instead, it will be modeled separately as an aspect, which means that the nature of the protocol can easily be modified if necessary. Note that it is not the aim of this paper to provide a mechanism to identify aspects. To do that, we can make use of some aspect mining techniques such as EA-Miner [16].

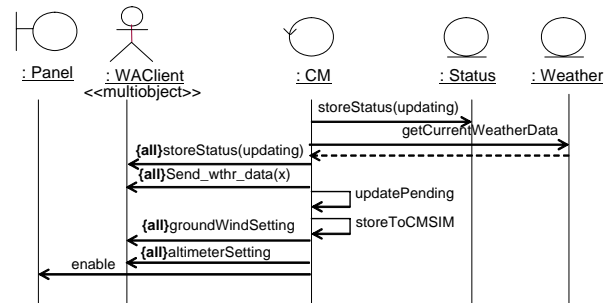


Figure 6: New weather data arrives. Update clients.

We give a brief description of the behavior specified in Figures 5 and 6. In Figure 5, the behavior is triggered by a connection request from a new client (not shown). CM responds by starting an initialization procedure for the client and storing data related to that procedure. It then queries a database for current weather data and sends this data to the client along with two other settings. It finishes by enabling a GUI panel that had been enabled when the new client requested connection. In Figure 6, the behavior is similar except that new weather data has arrived (either triggered automatically or manually) – this is not shown in the figure. The CM again stores data related to the procedure, queries the weather database and broadcasts the data to all weather aware clients. It carries out some additional storage operations and re-enables the GUI panel.

The two-phase commit protocol is modeled as an aspectual sequence diagram in eRBML in Figure 7. The aspect defines a general pattern of communication for the protocol that can be reused across many applications. In the figure, there are two classifier roles - *|Participant* and *|Commit Server*. An instance of this aspect would instantiate actual classifiers to play those roles. Similarly, the interactions in the figure are given as message roles that can be instantiated to specific message names. The figure only commits a transaction if all participants acknowledge and agree to the transaction. For this case study, the aspect is first refined because the logic of the CTAS application requires that information about the progress of the protocol is stored. This information becomes part of a refined aspect shown in Figure 8.

Figures 9 and 10 respectively give the LHS and RHS of a graph transformation that composes the refined aspect (Figure 8) with the base functionality in Figure 6. Note that the LHS says that the aspect will apply at all points in which there is an instance of a `|prepareToCommit` message, followed by a sequence of messages (given as a role representing a UML2.0 `ref` fragment), followed by an `enable` message. This application condition is true for both of the core scenarios. A graph matching algorithm can be used to search for a match of the LHS with base scenarios. In this case, `|prepareToCommit` can only be instantiated to `Send_wthr_data`, `|CommitServer` can only be instantiated to `CM`, but `|Participant` can be instantiated to either a single weather-aware client or a multiobject of weather-aware clients. `|OtherMessages` matches with any sequence of messages between the messages `Send_wthr_data` and `enable`.

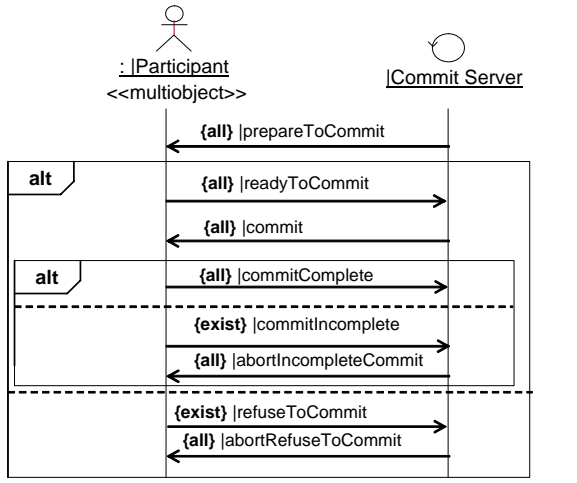


Figure 7: General Two-Phase Commit Protocol.

The RHS of the graph transformation is given in Figure 10. Note the placement of `|OtherMessages`. Taking one example of the composition, in Figure 6, `|OtherMessages` matches to `updatePending`, `storeToCMSIM`, `{all}|groundWindSetting`, `{all}|altimeterSetting`. These messages should only be sent if the two-phase commit protocol successfully completes. But the two-phase commit aspect contains cases both for successful commit and unsuccessful commit. The composition of the aspect and base model must be done in such a way that the matches for `|OtherMessages` are inserted only at the point of successful commitment. Using the existing methods by France et al. and Whittle & Araújo, this would be difficult and awkward to specify. In the approach by Whittle & Araújo, there would have to be a predefined composition operator that would allow the weaving in the manner described. Currently, there is no such operator. In the approach by France et al., the modeler would have to give a list of composition directives that instruct the composition algorithm where to place the messages matching with `|OtherMessages`. This would be a time-consuming and error-prone process because there is no easy, graphical way to specify these directives. In essence, the graph transformation is a graphical way to specify the directives. The modeler is free to place `|OtherMessages` wherever s/he likes on the RHS of the graph transformation rule. Note also that the modeler can easily specify a different composition strategy by simply modifying the RHS of the rule.

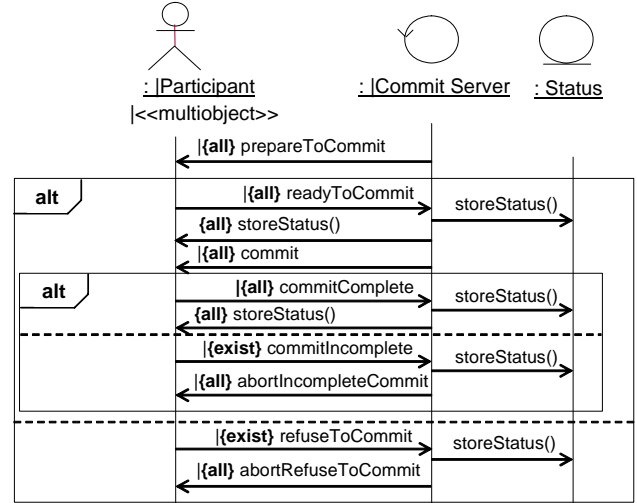


Figure 8: CTAS specific two-phase commit

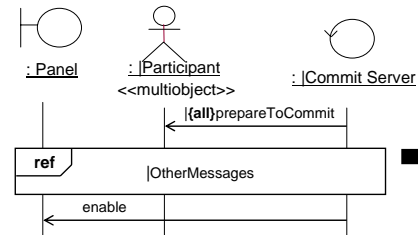


Figure 9: LHS of graph transformation.

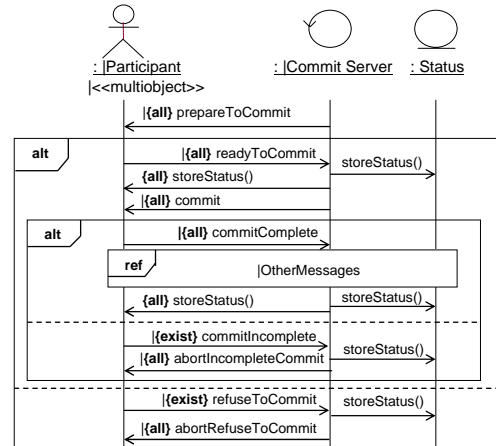


Figure 10: RHS of a graph transformation that composes the refined aspect in Figure 8

We do not show here the results of composing the aspect with each of Figures 5 and 6 but leave it to the reader to apply the graph transformation and hence to generate the compositions. Note that the entire application was modeled using our approach but only a small part can be presented in this paper.

5. RELATED WORK

Graph transformations have a noble history in software engineering. Their use has been suggested for viewpoint integration [8] in requirements engineering, software refactoring [2], and generative programming [17]. Graph transformations are

even being considered as a potential standardized model transformation language as part of the MOF 2.0 Query/Views/Transformations (QVT) effort [15]. However, to the author's knowledge, there has been no in-depth research on composing aspects using graph transformations. The UMLAUT framework [20] is a tool to transform UML models that is not explicitly based on graph transformation. Its use to compose aspects in UML has been investigated by defining transformation rules over a UML model given by an abstract syntax tree. However, UMLAUT has its own definition for transformation rules and does not make use of the intuitive graphical nature of graph transformations. Graph transformations defined over eRBML models resemble the equivalent UML models very closely and hence can be understood easily by practitioners.

A number of approaches have applied template-based UML models for defining aspects. Templates are a poor man's version of eRBML because their instantiation mechanism is very restricted. [3], for example, deals with crosscutting concerns as UML templates at the design level. Template binding is used to compose the pattern with concrete model elements. The compositions, however, are not as flexible as in our approach. The aspect-oriented software development group at Colorado State University has conducted extensive research on using RBML or related languages to represent aspects. We feel that graph transformations offer a more scalable and more expressive way of defining aspect compositions.

There has also been considerable recent work on defining aspects during requirements engineering. The aspect-oriented requirements engineering (AORE) approach [12] defines detailed composition rules for aspectual requirements in XML. AORE focuses on textual requirements and hence is complementary to our work which focuses on graphical models.

6. CONCLUSIONS & FURTHER WORK

Existing techniques to represent and compose aspects at requirements and design level suffer from several problems, in particular they are not scalable, requiring a lot of detailed work from the modeler. This paper described an approach to represent aspects using a UML-based role modeling language to define the aspect and graph transformations to define aspect compositions. Hierarchies were used to structure aspects and their possible instances. This approach is scalable because graph matching can be applied to automatically compose aspects, and expressive because the modeler defines any kind of composition in the graph transformation rule which is expressed in an intuitive graphical form. To validate the approach, we applied it to part of an air traffic control system developed at NASA.

7. REFERENCES

- [1] J. Araújo, J. Whittle and D-K. Kim, "Modeling and Composing Scenario-Based Requirements with Aspects", RE 2004, Kyoto, Japan, IEEE CS Press, 2004.
- [2] P. Bottoni, F. Parisi-Presicce, G. Taenzer, "Specifying Integrated Refactoring with Distributed Graph Transformations," AGTIVE '03, 227-272, 2003.
- [3] S. Clarke and R. J. Walker, "Composition Patterns: An Approach to Designing Reusable Aspects". ICSE 2001, Toronto, Canada, IEEE CS Press, pp. 5-14, 2001.
- [4] D. Denery, H. Erzberger, T. Davis, S.Green and B. McNally, "Challenges of Air Traffic Management Research: Analysis, Simulation and Field Test". In AIAA Guidance, Navigation and Control Conference, 1997.
- [5] R. France, D-K. Kim, S. Ghosh and E. Song, "A UML-Based Pattern Specification Technique". IEEE Transactions on Software Engineering, Vol. 30(3), pp. 193-206, 2004.
- [6] R. France, I. Ray, G. Georg and S. Ghosh, "An Aspect-Oriented Approach to Design Modeling", IEE Proceedings Software, Vol 151(4), pp. 174-186, August 2004.
- [7] G. Georg and R. France. "UML Aspect Specification using Role Models". OOIS, France, Lecture Notes in Computer Science, Springer, Vol. 2425, pp. 186-191, September 2002.
- [8] M. Goedicke, B. Enders, T. Meyer, G. Taentzer, "ViewPoint-Oriented Software Development: Tool Support for Integrating Multiple Perspectives by Distributed Graph Transformation." TACAS 2000: 43-47
- [9] D-K. Kim, A Metamodeling Approach to Specifying Patterns, PhD Thesis, Colorado State University, 2004.
- [10] Dae-Kyoo Kim, Robert France, Sudipto Ghosh and Eunjee Song, "A Role-Based Metamodeling Approach to Specifying Design Patterns", COMPSAC 2003, Dallas, Texas, 2003.
- [11] A. Moreira, A. Rashid, J. Araújo, A. "Multi-Dimensional Separation of Concerns in Requirements Engineering," RE 2005, IEEE Computer Society, France, 2005.
- [12] A. Rashid, A. Moreira and J. Araújo, "Modularisation and Composition of Aspectual Requirements". AOSD 2003, Boston, USA, ACM Press, pp. 11-20, March 2003.
- [13] I. Ray, N. Li, R. B. France and D-K. Kim, "Using UML to Visualize Role-based Access Control Constraints." SACMAT 2004: 115-124.
- [14] G. Rozenberg, editor. "Handbook of Graph Grammars and Computing by Graph Transformation, Volume 1: Foundations." World Scientific, 1997.
- [15] OMG: Revised submission for MOF 2.0 Query/View/Transformation rfp (ad/2002-04-10) (2005).
- [16] A. Sampaio, R. Chitchyan, A. Rashid, P. Rayson, "EA-Miner: A Tool for Automating Aspect-Oriented Requirements Identification", ASE 2005, IEEE Computer Society, 2005.
- [17] S. Sendall, "Combining Generative and Graph Transformation Techniques for Model Transformation: An Effective Alliance?" OOPSLA '03 Workshop "Generative techniques in the context of MDA", 2003.
- [18] UML, version 2.0. Available from the Object Management Group, 2005, <http://www.omg.org>.
- [19] Meta Object Facility, MOF 2.0 core specification, <http://www.omg.org/docs/ptc/03-10-04.pdf> (2004)
- [20] UMLAUT: Unified Modeling Language All pUrpose Transformer, <http://www.irisa.fr/UMLAUT/>
- [21] J. Whittle and J. Araújo, "Scenario Modeling with Aspects", IEE Proceedings Software, Vol 151(4), pp. 157-172, 2004.