

Uniform Support for Modeling Crosscutting Structure

Maria Tkatchenko Gregor Kiczales
 University of British Columbia
 {tkatch, gregor}@cs.ubc.ca

ABSTRACT

A simple join point model supports crosscutting among different perspectives of a model, including class diagrams, sequence diagrams, advice, inter-type declarations and role-bindings.

A simple weaver works to coordinate the crosscutting structure across the model elements. Coordination involves resolving the separate crosscutting structures to find all interactions between join points. This woven structure is represented as a simple extension to the UML meta-model, which makes it easily accessible to the modeling environment and other tools. Simple and uniform access to the woven structure enables aggregate analysis and reasoning about the crosscutting in the model. This can range from simple visual hyperlinks between crosscutting model elements to more elaborate structures such as net sequence diagrams. The woven structure can also be opened to direct queries from the user, to aid in model development.

1. INTRODUCTION

UML provides support for modeling a system from various different perspectives [17]. Some of these perspectives have a hierarchical relationship to each other, such as package and class diagrams. Others have a crosscutting relationship, including sequence and class diagrams [8], and collaboration and class diagrams [23].

There have also been proposals to extend UML with newer crosscutting modeling constructs such as advice, inter-type declarations and role binding [25], [10], [26], [2], [21].

Our work aims to support crosscutting modeling relationships by adding a simple uniform join point model¹ (JPM) to the core of UML. This JPM supports existing class/sequence diagram crosscutting, as well as simple UML extensions for inter-type declarations, advice and role binding. So rather than using modeling to support AOP, we are using aspects to enhance modeling.

Weaving in our system consists of collecting the crosscutting at each model element. Our design provides access to the explicit crosscutting structure in the form of a list of elements that crosscut any given element. A variety of tools can use the explicit crosscutting structure our weaver collects to present the information to the modeler; but issues of user interface and ideal presentation to the modeler are beyond the scope of this paper.

¹ In the context of this paper, the term join point *model* can cause confusion with the models written using the modeling language. The join point model is really a meta-model construct. To help avoid confusion we use the JPM abbreviation throughout the paper.

The contributions of this work are to show that (i) the crosscutting structure of several traditional modeling relationships, as well as newer aspect modeling relationships can be supported by a uniform JPM, (ii) weaving these elements together to make the crosscutting structure explicit is straightforward, (iii) the combination of (i) and (ii) makes it easy for modeling tool implementers and end-modelers to access, analyze and display crosscutting relationships of interest.

2. ARCHITECTURE

Our system has three parts. First, we extend a subset of the UML meta-model with a simple JPM. Second, we extend a subset of the UML modeling language with advice, inter-type declarations and role bindings. Finally, we provide a weaver to process our extensions. Taken together, these three allow native support for both existing and new kinds of crosscutting model elements. The existing crosscutting can be seen between class and sequence diagrams, multiple sequence diagrams, or class and collaboration diagrams. The new crosscutting comes from advice, inter-type declarations [11] and role bindings [14].

A model in our system consists of model elements written using one of three sub-languages: class diagrams with inter-type declarations and advice, sequence diagrams, and role bindings.

Class Diagrams. We currently support class and interface elements with generalization (inheritance), each of which can also have attributes (fields) and operations (methods). We do not currently support collaborations, dependencies, aggregation, realizations, etc., due to the fact that our work is only in its early stages. We allow methods and fields to be *intertype declarations* (ITDs) in that they can be defined in one class, but actually define a member of another class.

Sequence Diagrams. A sequence consists of a series of method calls, each of which in turn leads to a (possibly empty) sequence which models the execution of the method in that sequence diagram. We represent the focus of control of each object as the sub-sequence associated with a call.

Advice declarations are an extension to UML and consist of a pointcut and a body. The body is currently just a string. The pointcut is an AspectJ-style pointcut that can match simple patterns. An advice declares that join points matched by the pointcut are advised (by the string). And advice can be either before or after type.

Role bindings are an extension to UML and specify the binding of a class to a role or of a class-method to a role-method. Roles and role-methods are modeled by ordinary classes and methods. Note that the binding of roles is not the same as the standard concept of binding in UML, where binding refers to the creation of an element from a template [5].

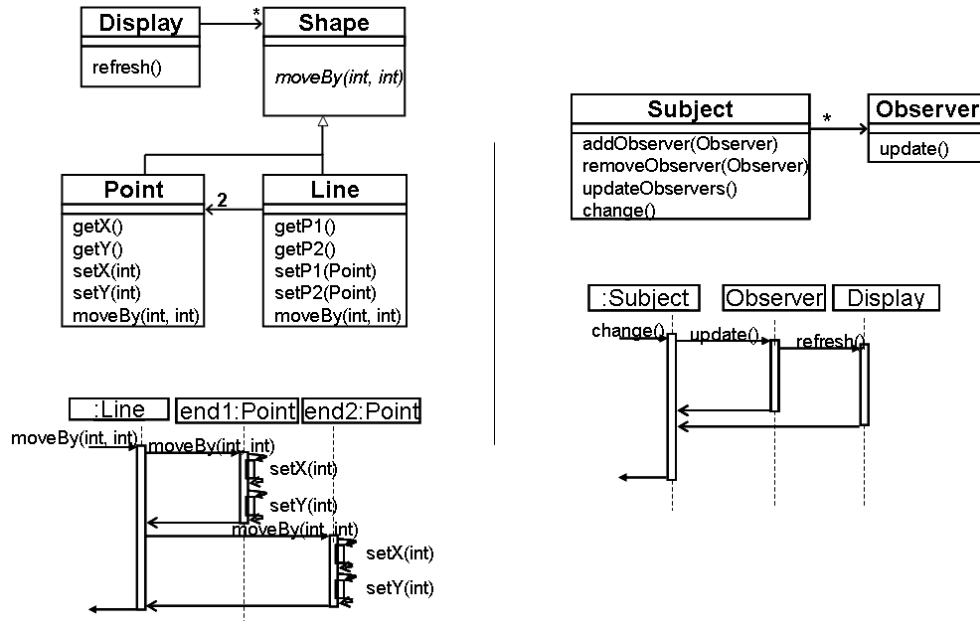


Figure 1: Shape Example with the Observer pattern

2.1 EXAMPLE

The example we use in this paper is shown in Figure 1. It is an adaptation of the familiar AspectJ figures/shapes example [11], [12]. Two simple model fragments are shown, each of which consists of a class and a sequence diagram. One models the main functionality of Point and Line shapes. The other models the Observer design pattern. In addition, advice can be written to advise, for example, the change operations (the execution of set* and moveBy methods in Shape, Point, and Line). We have omitted the exact syntax of the advice from the figure as it is not essential to the following discussion.

2.2 THE JPM

A JPM can be described in terms of the nature of the join points, a means of identifying join points, and a means of affecting semantics at join points [13]. The discussion below uses this ontology.

2.2.1 JOIN POINTS

The join points of our JPM are declarations in the model.

From class diagrams the join points are class, method, and field declarations, as well as any inter-type declarations and advice.

From sequence diagrams the join points are sequence and method call declarations. We do not yet include method return or method call reception join points.

Each role binding declaration is a join point.

Because our join points are all declarations, we will often refer to a join point as a declaration.

2.2.2 MEANS OF IDENTIFYING JOIN POINTS

Our system supports several mechanisms for identifying join points. All constructs that include a declaration signature of some form are essentially considered as identifying join points. For example, a method call declaration in a sequence diagram

identifies method declarations with the same signature. Similarly method declarations identify method calls and sequences with matching signatures.

Advice declarations include a pointcut construct similar to that in AspectJ. We support several primitive pointcuts, including: call, sequence, method, field, class, advice, each of which retrieves the corresponding method element. We also support pointcuts such as within, cflow and cflowbelow which correspond in meaning to the pointcuts with the same name in AspectJ.

Note that in our model a method declaration and a method call declaration are both considered join points. So, when their signatures match, we end up with two join points that crosscut each other, rather than two model elements that crosscut at a single join point.

2.2.3 SEMANTIC EFFECT AT JOIN POINTS

In our JPM, semantic effect simply means that two model elements crosscut each other, and during weaving this fact is recorded for easy retrieval during analysis. Thus, each join point now contains all information relevant to its structure and behavior. The semantics of each model element is defined by that element.

2.3 THE WOVEN MODEL

The weaving process takes the model elements, and computes the crosscutting among them. We use a simple meta-model for this woven structure. For each join point the weaver records a list of all other join points it crosscuts, we call this the *crosscut by list* of a join point. The woven model can then be output in any format, depending on the view desired by the user.

For example, a method in a class diagram will have on its *crosscut by list* all sequence or call declarations with a matching method signature. Also on the *crosscut by list* will be any advice declarations with a pointcut that matches the join point.

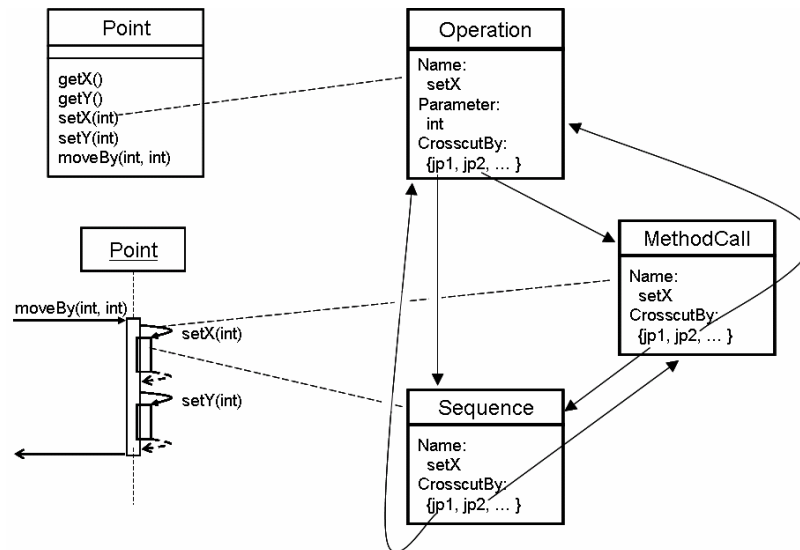


Figure 2: The corresponding internal representation of model elements in the left part of the figure is shown on the right.

In our approach crosscutting is treated symmetrically. The *crosscut by list* of a sequence declaration includes the method declaration it crosscuts and vice versa. Our model has multiple join points in cases where a different model might have one. For example, the right side of Figure 2 shows the *crosscut by list* of the `setX` method declaration, which includes the join points corresponding to a call to `setX` and the corresponding execution sequence.

A distinction should be made between the internal and external representations of the crosscutting information. Figure 2 (left side) shows the external representation, in standard UML format, which would be seen by the user. On the contrary, Figure 2 (right side) shows an internal representation, which is used by the weaver. As can be seen, the correspondence between these two representations is what would make it easy for a tool to use the internal representation used by the weaver to create the UML picture that is more familiar to the user. Figure 3 shows a more precise description of our changes to the meta-model.

We believe that this simple and uniform model will support easy extension. When new crosscutting modeling language constructs are added, no complex changes are required to the woven meta-model. The woven model can also be queried, with each of the queries being defined by a pointcut expression and returning a subset of elements whose join points match the pointcut.

2.4 THE WEAVING PROCESS

One issue we are working on has to do with semantic dependencies between different kinds of model elements.

For example, should advice be able to be defined in terms of the effect of a role binding? In concrete terms, should advice on `Subject.change` be applicable to `Point.setX` given the role binding in our example? We believe it clearly should. We also believe that advice should be able to depend on ITDs.

But if role binding needs to be able to depend on the effect of ITDs, then we may have to adopt a fixed-point approach in our weaver. So far, we have been unable to come up with a convincing example that would require this change. One issue we

hope to get feedback on at the workshop is which approach we should adopt.

Initially, we are using a simple ordered approach as follows:

- 1) Role binding. In this step, all the role bindings used in the model are processed. These relationships are stored in a way that affects later operation and property lookup (through inheritance). In addition, each of the role bindings is added to the *crosscut by list* of both elements participating in the binding, and vice versa.
- 2) Inter-type declarations. New structural elements are introduced in the model as specified by inter-type declarations. These relationships are stored in a way that affects later operation and property lookup (through inheritance). The inter-type declaration elements themselves are also added pair-wise to the *crosscut by lists* of participating elements, and vice versa.
- 3) Sequence diagrams. Crosscutting relationships between elements of a class and sequence diagram, or two sequence diagrams, are captured in this step. The crosscutting elements are added to *crosscut by lists* pair-wise. For example, if a method and a method call share the same signature, the method call will be added to the method's *crosscut by list*, and vice versa. This is shown in Figure 2.
- 4) Advice. For each advice, all the join points matching its pointcut are added to the advice's *crosscut by list*, and vice versa.

The addition of support for new kinds of crosscutting to the model will require a new phase of processing if the elements aren't recognized by any of the above steps. In addition, the position of the new step in the ordering will also have to be determined. However, none of the original steps will need to be modified to accommodate this change.

After the whole process is complete, the woven structure contains information about all the crosscutting that is present in the model and supported by our prototype. In this fashion, the woven model can be used by other tools, which then have uniform easy access to the crosscutting information that has now been made explicit. The correspondence between the internal and external representations is discussed in Section 2.3, and supports this view

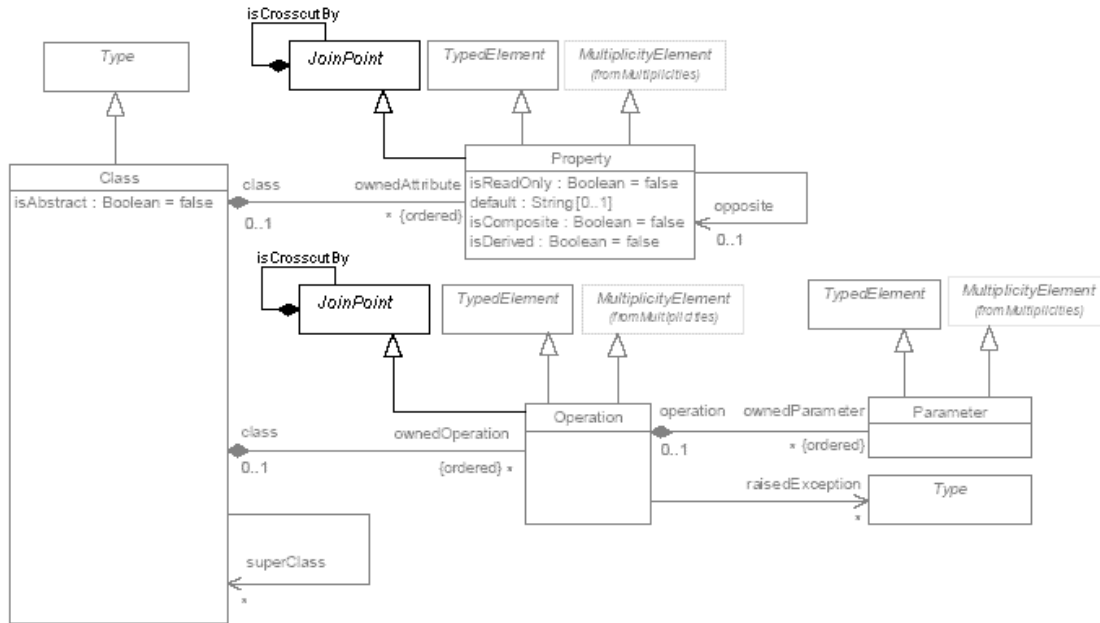


Figure 3: Extended class diagram meta-model.

of the model. Since XML [29] is becoming a standard for model interchange between tools, we expect that many modeling tool can be modified to present this information in the desired format.

3. EVALUATION

The hypothesis we set out to test was that the addition of a simple JPM to UML could provide uniform support for modeling crosscutting concerns, that this would simplify display and analysis of existing crosscutting structure in UML, and make the addition of new kinds of crosscutting elements easier. In this section we use the example of Section 2.1 to assess the degree to which our current system supports the initial hypothesis.

Our system provides a simple way of dealing with existing crosscutting in UML models. All kinds of crosscutting are treated uniformly, ensuring that all the existing crosscutting elements are captured in the same framework. The underlying JPM simplifies display operations and reasoning about the model. One might want the IDE to mark declarations that have advice. The following method implements the underlying test based on our model:

```
boolean hasAdvice(Decl decl) {
    for (Decl otherDecl: decl.getCrosscutByList())
        if (decl instanceof Advice)
            return true;
    return false;
}
```

In terms of the weaver, for each such display/query operation, a single visitor needs to be written that would check the appropriate conditions. Since all the existing elements are classified in terms of their structure and *crosscut by lists*, no changes need to be made to the representation of elements, or the part of the weaver that builds up the *crosscut by lists*.

The second claim we made is that the addition of new kinds of crosscutting modeling constructs is easy. Advice is not built in to UML, and in our implementation, the work required to add advice is modest. A new model element to represent advice must be

defined, that inherits from JoinPoint. A new phase must also be added to the weaver, that implements advice (Section 2.4 step 4). The implementation of pointcut matching is somewhat involved, but all of that is well decoupled from the core meta-model and weaver. So it would have been little work for a tool developer, using our framework, to add advice if we had not already added it.

It is also easy to add support for different patterns, as demonstrated in the example with the Observer pattern and role bindings. In contrast to XDE [7], patterns are base model elements, not meta-level elements.

We argue that the implicit weaving of the behavior and structural features together is a strength of our system. Other proposals [22] require the user to specify which elements they want woven together, allowing them to say which subset of structure or behavior they wish to see. We expand on this by having the weaving happen automatically, and then, in the simplest approach, having the user select the elements they are interested in, displaying the crosscutting effects from the rest of the model. In addition, we intend to include the ability to select only a subset of model elements and show the crosscutting relationships strictly between those elements. The main idea is that all the crosscutting structure is already present in all the elements in our woven model, and by selecting the different views the user can choose to display either part or all of that information. This is in contrast to approaches where the crosscutting information has to be collected for each weave independently [6], or even specified individually for each crosscutting relationship [25]. The main advantage we see to this is scalability, where the addition of new elements to the model will not force major changes to how crosscutting relationships between the elements are handled.

4. IMPLEMENTATION

We have developed a prototype implementation of our tool based on the Eclipse Modeling Framework (EMF) version 2.0.0 [3], which allows for creation and display of class and sequence

diagrams. We extended the EMF implementation of the UML meta-model with join points, inter-type-declarations, role bindings and advice. We also implemented a simple weaver.

To provide information about crosscutting structure, we took advantage of an EMF feature whereby it displays textual properties of every meta-object. Our weaver not only computes *crosscut by lists* for declarations, it also computes a textual summary of that *crosscut by list*. EMF automatically displays that summary, which lets us see whether declarations are advised, participate in role bindings, appear in sequence diagrams, etc.² It should be noted that the work discussed here is only a prototype, and thus we do not currently make it widely available.

Discussion of performance and other aspects of implementation are beyond the scope of this workshop paper.

5. RELATED WORK

There have been a number of proposals for extending UML to allow for design with aspects, and a number of them relate more or less directly to our work. Work on tool support for crosscutting and model composition can also be found.

The closest to our work is probably Katara, who proposes building a refinement hierarchy for a class or sequence diagram [15]. Each concern is viewed as a collection of superposition steps that define it. The authors use the term *aspect* to refer to these concerns. All additional functionality crosscuts the starting model, so each is an aspect, even though it is part of the core functionality of the final model. It is possible to merge sequence diagrams to see the composed behavior of a number of sequences.

Prehofer's work [19] addresses the merging of state chart diagrams in much the same way as we treat sequence diagrams. Just like Katara he proposes a mechanism to expand more abstract diagrams by substituting the more concrete sub-diagram into the larger one for each concern.

Stein et al. [24] take the ideas developed by the previous two authors further, by introducing weaving as a concept in the extended meta-model of UML. They reproduce in UML the idea of weaving in AspectJ. As a result, sequence diagrams that crosscut each other can be merged to display the final expected behavior. The modeler has to specify the crosscutting elements by means of a weaving instruction that specifies the bindings. By contrast, in our system the developer only needs to specify the role bindings, and then the structural and crosscutting relationships between elements are established automatically. In addition, weaving of sequence diagrams implies splitting apart and later composing the sequences with the new elements, at each of the join points that are affected. In this scenario, the set of calls in a sequence may need to be totally ordered to properly complete the weaving.

Kande argues that aspects need to be first-class elements in UML [10]. This is contrary to the approach taken by the above works, which is that aspects can be introduced by using UML's extension mechanisms such as stereotypes [9], [17]. Kande's approach, however, is an indication of the fact that others have successfully

argued for the approach of introducing aspects from the bottom-up, instead of building on top of the existing framework. He argues that the standard UML models, when composed with the aspect model, do not leave the separate concerns well modularized – the elements in the design model are coupled more than they would be coupled in the code. In addition, the model does not communicate the ability to plug and un-plug aspects from the core functionality depending on the semantics desired. The main reason is that since UML doesn't define the concept of weaving, the well-separated concerns in the AO program end up being scattered throughout the design model. Thus, a model element that encapsulates the role of the aspect as well as models the interactions between all crosscutting objects may be needed.

One important difference between the approaches surveyed and our work is that we make weaving implicit, or automatic, and allow the modeler to select which *view* to take on the woven structure.

Pawlak presents a notation for designing AO programs [20]. Aspect-classes are a new element, that contains regular methods and aspect methods, which extend the semantics of base classes. The pointcuts are represented as pointcut relations, which link aspect-methods to points in the base class.

Lions also notes the restrictions that arise when using stereotypes and profiles to extend UML, and proposes introducing AOP into a lower level through the use of meta-modeling [16]. The argument is that given a meta-model, it is relatively easy to provide support in a tool for models created based on the meta-model. Since we are essentially changing the meta-model for UML by introducing a number of new AO elements into the standard model, it is helpful to know that the question of tool support has been considered by others. We believe that the ability to provide modeling tool support for a changed meta-model, coupled with the ease of combination of model elements and their relationships with our tool and meta-model, support our view that extending the UML model to include AO concepts should be done from the bottom-up.

Straw et al. look into composing primary and aspect class diagrams [4]. They note that during analysis of the composed model, conflicts and undesirable emergent properties can be identified post-composition. However, the proposed composition directives require the developer to be aware of the interplay and potential conflicts in the model, instead of helping them discover that information through analysis of crosscutting and other relationships in an existing model.

Stein and Hanenberg propose using UML collaborations to specify the details of crosscutting in a given decomposition, either structural or behavioral [28]. The reason for using collaborations is that the authors view UML collaborations as inherently crosscutting elements, since they are not guaranteed to describe the model elements in full. The crosscut and crosscutting elements can be specified independently, as can the composition strategy and the join points. Like some of the above approaches, the modeler is required to explicitly state all the crosscutting relationships and points while designing the system.

Ossher and Tarr present HyperJ [18], a tool which supports multi-dimensional separation of concerns [27]. The main idea is that a program can be decomposed in any number of ways. HyperJ is a

² Ideally, EMF would just display the contents of the crosscut by list itself, but version 2.0.0 does not allow us to do that easily.

tool that allows the developer to use this concept by allowing “identification, encapsulation and integration of multiple dimensions of concern”. They suggest that their approach could be used at any stage of the software development life cycle, but do not describe an implementation of the approach for modeling.

Finally, ThemeUML [1] is a tool that provides support for aspects throughout the lifecycle, from identifying them in requirements to modeling them using a design language. This approach requires explicit identification and binding of aspects by the modeler, although it allows nice continuity from requirements identification through to modeling.

6. SUMMARY

We propose bottom-up support for aspects in UML, by adding a simple JPM to the UML meta-model. This is in contrast to proposals based on stereotypes. We feel that our approach simplifies implementation of tool support for working with crosscutting structure, addition of new kinds of crosscutting structure, and also makes models of crosscutting structure more declarative and scalable.

7. ACKNOWLEDGMENTS

Thanks to Marcellus Mindel, Bran Selic, Paul Elder, Charles Riballe and other participants of the IBM Ottawa CASTLE Poster Session for their feedback on this work.

This work is partially supported by IBM Center for Advanced Studies and the Natural Sciences and Engineering Research Council of Canada (NSERC).

8. REFERENCES

- Andersen, E.P. and Reenskaug, T. System Design by Composing Structures of Interacting Objects. *Proc. of ECOOP*. 133 - 152.
- Baniassad, E. and Clarke, S., Theme: An Approach for Aspect-Oriented Analysis and Design. *International Conference on Software Engineering*, (2004).
- Booch, G., Rumbaugh, J. and Jacobson, I. *The Unified Modeling Language User Guide*. Addison-Wesley, 1999.
- Dae-Kyoo Kim; France, R.G., S.; Eunjee Song A role-based metamodeling approach to specifying design patterns. *Proceedings of Computer Software and Applications Conference*. 452 – 457.
- EclipseProject Eclipse Modeling Framework. <http://download.eclipse.org/tools/emf/scripts/home.php>.
- Ho, W.-M., Jezequel, J.-M., Pennaneac'h, F. and Plouzeau, N., A toolkit for weaving aspect oriented UML designs. *Proceedings of the 1st international conference on Aspect-oriented software development*, (2002), ACM Press, 99-105.
- IBM Rational Rose XDE Developer. <http://www-306.ibm.com/software/awdtools/developer/rosexde/>.
- Jacobson, I. and Ng, P.-W. *Aspect-Oriented Software Development with Use Cases*. Addison Wesley Professional, 2004.
- Jezequel, J., Plouzeau, N., Weis, T. and Geihs, K., From Contracts to Aspects in UML Designs. *Aspect-Oriented Modeling with UML workshop at AOSD*, (2002).
- Kande, M.M., J. Kienzle and A. Strohmeier From AOP to UML - A Bottom-Up Approach. *Aspect-Oriented Modeling with UML workshop at the 1st International Conference on Aspect-Oriented Software Development*.
- Katara, M. and Mikkonen, T., Refinements and Aspects in UML. *Aspect-Oriented Modeling with UML Workshop at UML Conference*, (2002), UML 2002.
- Kiczales, G., Hilsdale, E., Hugunin, J., Kersten, M., Palm, J. and Griswold, W.G., An Overview of AspectJ. *European Conference on Object-Oriented Programming (ECOOP)*, (2001), Springer, 327-355.
- Kiczales, G. and Mezini, M., Aspect-Oriented Programming and Modular Reasoning. *ACM International Conference on Software Engineering*, (2005 (to appear)).
- Lions, J.M., Simoneau, D., Pitette, G. and Moussa, I., Extending OpenTool/UML Using Metamodeling: An Aspect Oriented Programming Case Study. *Workshop on Aspect-Oriented Modeling with UML at the UML Conference*, (2002).
- Masuhara, H. and Kiczales, G., Modeling crosscutting in aspect-oriented mechanisms. *European Conference on Object-Oriented Programming (ECOOP)*, (2003), Springer, 2-28.
- Mezini, A.M. and Ostermann, A.K., Conquering aspects with Caesar. *International Conference on Aspect-Oriented Software Development (AOSD)*, (2003), ACM Press, 90-100.
- OMG, T. Unified Modeling Language (UML), Version 1.5. www.uml.org.
- Ossher, H. and Tarr, P., Hyper/J: Multi-Dimensional Separation of Concerns for Java. *International Conference on Software Engineering*, (2000), ACM Press, 734-737.
- Pawlak, R., Duchien, L., Florin, G., Legond-Aubry, F., Seinturier, L. and Martelli, L., A UML Notation for Aspect-Oriented Software Design. in *Aspect-Oriented modeling with UML workshop at AOSD*, (2002).
- Prehofer, C., Feature Interactions in Statechart Diagrams or Graphical Composition of Components. *Workshop on Aspect-Oriented Modeling with UML at the UML conference*, (2002).
- Reifer, D. Doubts and hopes for AOP. *COMMUNICATIONS OF THE ACM*, 45 (3). 11-12.
- Selic, B., Using UML for Modeling Complex Real-Time Systems. *Languages, Compilers, and Tools for Embedded Systems: ACM SIGPLAN Workshop LCTES*, (1998).
- Stein, D., Hanenberg, S. and Unland, R., Designing Aspect-Oriented Crosscutting in UML. *Workshop on Aspect-Oriented Modeling with UML at AOSD*, (2002).
- Stein, D., Hanenberg, S. and Unland, R., Position Paper on Aspect-Oriented Modeling: Issues on Representing Crosscutting Features. *Workshop on Aspect-Oriented Modeling at AOSD*, (2003).
- Straw, G., Georg, G., Song, E., Ghosh, S., France, R.B. and Bieman, J.M., Model Composition Directives. *Conference on the Unified Modeling Language*, (2004).
- Tamai, T., Ubayashi, N. and Ichiyama, R. An adaptive object model with dynamic role binding. <http://www.graco.cu-tokyo.ac.jp/~tamai/pub/epsilon/rolemodel.pdf>.
- Tarr, P., Ossher, H., Harrison, W. and Sutton, S.M., N degrees of separation: multi-dimensional separation of concerns. *International Conference on Software Engineering (ICSE)*, (1999), IEEE Computer Society Press, 107-119.
- W3C. Extensible Markup Language, <http://www.w3.org/XML/>, 2003.