

Chapter 1

KEYWORDS, ETC.

(The first two pages only resemble the abstract infos and are needed to ensure proper layout of the rest of the article. Bernhard)

Name(s) and affiliation (s):

Andy Evans

University of York, York, UK

andy@cs.york.ac.uk

Robert France

Colorado State University, Colorado, US

france@cs.colostate.edu

Kevin Lano

Imperial College, London, UK

kcl@doc.ic.ac.uk

Bernhard Rumpe

Software & Systems Engineering

Munich University of Technology, Munich, Germany

rumpe@in.tum.de

Title:

Meta-Modelling Semantics of UML

Index items:

UML, Meta-model, Theory, Formalization, OCL, Semantics,

Generalisation

Abstract:

OO modelling notations such as the UML embody high-quality modelling experiences, but the lack of precise semantics for the notations prohibits rigorous analysis of the models. In this paper we examine how the existing semantics of UML can be strengthened to support the development of rigorous analysis and proof techniques. Our

approach involves developing a strategy for identifying extensions to the UML semantic model. These extensions more precisely capture the semantic meaning of UML modelling elements, while remaining consistent with the existing documentation. We then describe a way in which the notion of proof can be expressed within the existing semantics model. This enables model transformations to be justified as a means of rigorously verifying properties of UML models.

Chapter 2

META-MODELLING SEMANTICS OF UML

Andy Evans

*University of York, UK
andy@cs.york.ac.uk*

Kevin Lano

*Imperial College, UK
kcl@doc.ic.ac.uk*

Robert France

*Colorado State University, US
france@cs.colostate.edu*

Bernhard Rumpe

*Munich University of Technology,
Germany
rumpe@in.tum.de*

Abstract OO modelling notations such as the UML embody high-quality modelling experiences, but the lack of precise semantics for the notations prohibits rigorous analysis of the models. In this chapter we examine how the existing semantics of UML can be strengthened to support the development of rigorous analysis and proof techniques. Our approach involves developing a strategy for identifying extensions to the UML model. These extensions more precisely capture the semantic meaning of UML modelling elements, while remaining consistent with the existing documentation. We then describe a way in which the notion of proof can be expressed within the existing semantics model. This enables model transformations to be justified as a means of rigorously verifying properties of UML models.

1. INTRODUCTION

The Unified Modeling Language (UML) [BRJ98, RJB99] is rapidly becoming a de-facto language for modelling object-oriented systems. An important aspect of the language is the recognition by its authors of the need to provide a precise description of its semantics. Their intention is that this should act as an unambiguous description of the language, whilst also permitting extensibility so that it may adapt to future changes in object-oriented analysis and design. This has resulted in a Semantics Document [OMG99], which is presently being managed by the Object Management Group, and forms an important part of the language's standard definition.

The UML semantics is described using a meta-model that is presented in terms of three views: the abstract syntax, well-formedness rules, and modelling element semantics. The abstract syntax is expressed using a subset of UML static modelling notations. The abstract syntax model is supported by natural language descriptions of the syntactic structure of UML constructs. The well-formedness rules are expressed in the *Object Constraint Language* (OCL) and the semantics of modelling elements are described in natural language. The advantage of using the meta-modelling approach is that it is accessible to anybody who understands UML. Furthermore, the use of object-oriented modelling techniques helps make the model more intuitively understandable.

A potential advantage of providing a precise semantics for UML is that many of the benefits of using a formal language such as Z [S92] or Spectrum [BFG⁺93] might be transferable to UML. Some of the major benefits of having a precise semantics for UML are given below:

Clarity: The formally stated semantics can act as a point of reference to resolve disagreements over intended interpretation and to clear up confusion over the precise meaning of a construct.

Equivalence and Consistency: A precise semantics provides an unambiguous basis from which to compare and contrast the UML with other techniques and notations, and for ensuring consistency between its different components.

Extendibility: The soundness of extensions to the UML can be verified (as encouraged by the UML authors).

Refinement: The correctness of design steps in the UML can be verified and precisely documented. In particular, a properly developed semantics supports the development of design transformations, in which a more abstract model is diagrammatically transformed into an implementation model.

Proof: Justified proofs and rigorous analysis of important properties of a system described in the UML require precise semantics. Proof and rigorous analysis are not currently supported by UML or any of its tools.

Unfortunately, the current UML semantics are not sufficiently formal to realise many of these benefits. Although much of the syntax of the language has been defined, and some static semantics given, dynamic semantics are mostly described using lengthy paragraphs of often ambiguous informal English, or are missing entirely. Furthermore, little consideration has been paid to important issues such as proof, compositionality and rigorous tool support. A further problem is the extensive scope of the language, all of which must be dealt with before the language is completely defined.

This chapter describes work being carried out by the precise UML (pUML) group and documented in [PUM1999, FELR98, EFLR98]. PUM1 is an international group of researchers and practitioners who share the goal of developing UML as a precise (formal) modelling language, thereby enabling it to be used in a formal manner. This chapter reports on work being carried out to strengthen the existing semantics of UML. In Section 2., a formalisation strategy is described (developed through the experiences of the group) that aims to make precise the existing UML semantics. To achieve this

goal, the benefits of concentrating on a core UML semantics model given are examined in Section 3.. Section 4. describes how the formalisation strategy has been applied to the development of a precise understanding of a small yet interesting part of the UML semantics - generalisation/specialisation hierarchies. Finally, in Section 5. it is considered whether the notion of proof can be expressed in an abstract way within this model.

2. FORMALISATION STRATEGY

In developing a formalisation strategy for UML it is necessary to consider the following questions:

1. Does the existing (informal) semantics form a suitable basis for a formal semantics?
2. What is the most appropriate approach to assigning a formal semantics to UML components?
3. Given the large scope of UML, which parts should be formalised first?

In answer to the first question, it is necessary to understand both the practical and theoretical aspects of using the existing standard as a basis for a formal semantics. From a practical point of view, it is important to recognise that UML is a standard, and therefore radically different semantic proposals are unlikely to be incorporated. From a theoretical standpoint, meta-modelling is a commonly adopted technique for assigning a formal semantics to languages. However, it is also possible to argue that the use of OCL is inappropriate for this task, as it does not have a precise semantics. To address both these arguments, we adopt the use of formal notations to elicit and understand modelling concepts, but present the final definition in OCL as a means linking this with the existing meta-model.

The second question recognises that there are many approaches used to assign semantics to languages. One of the best known (and most popular) is the denotational approach (for an in-depth discussion see [S86]). The denotational approach assigns semantics to a language by giving a mapping from its syntactical elements to a meaningful representation. For example: an operation might be denoted as a relation between before and after states; a multiplicity might be denoted by a set of integer values. As UML already adopts the denotational approach to describe aspects of its language, it is logical to adopt it in the formalisation strategy. However, greater emphasis will be placed on obtaining more complete and precise descriptions of denotations than is currently pursued in the UML semantics.

To cope with the large scope of the UML it is natural to concentrate on essential concepts of the language to build a clear and precise foundation as a basis for formalisation. Therefore, the approach taken in the group's work is to concentrate on identifying and formalising a core semantic model for UML before tackling other features of the language.

2.1 STAGES

The complete formalisation strategy consists of the following steps:

1. Identify the core elements of the existing UML semantics;
2. Iteratively examine the core elements, seeking to verify their completeness. Here, completeness is achieved when: (1) the modelling element has a precise syntax, (2) is well-formed, and (3) has a precise denotation in terms of some fundamental aspect of the core semantic model;
3. Use formal techniques to gain better insight into the existing definitions as shown in [FELR98, EFLR98];
4. Where in-completeness is identified, we attempt to address it in a number of ways, depending on the type of omission found.

Model strengthening - this is necessary where the meaning of a model element is not fully described in the meta-model. The omission is fixed by strengthening the relationship between the model element and its denotation;

Model extension - in certain cases it is necessary to extend the meta-model to incorporate new denotational relationships. This occurs when no meaning has been assigned to a particular model element, and it cannot be derived by constraints on existing associations. For example, this is necessary in the case of *Operation* and *Method*, where the meaning of a method is defined in terms of a *procedureExpression* and *Operation* is given no abstract meaning at all;

Model simplification - in some cases, aspects of the model are surplus to needs, in which case we aim to show how they can be omitted or simplified without compromising the existing semantics.

5. Feed the results back into the UML meta-model, with the aim of clarifying the semantics of a core part of the UML;
6. Disseminate to interested parties for feedback;

Finally, it is important to consider how the notion of *proof* can be represented in the semantic model. This is essential if techniques are to be developed for analysing properties of UML models. Such analysis is required to establish the presence of desired properties in models. The need to establish properties can arise out of the need to establish that models adhere to requirements or out of challenges posed by reviewers of the models. Proof is also important in understanding properties of model transformations in which a system is progressively refined to an implementation [BHH⁺97].

3. THE CORE SEMANTICS MODEL

The question of what should form a core precise semantics for UML is already partially answered in the UML semantics document. It identifies a ‘Core Package - Relationships’ package and a number of ‘Common Behaviour’ packages. The Core Relationship package defines a set of modelling elements that are common to all UML diagrams, such as *ModelElement*, *Relationship*, *Classifier*, *Association* and *Generalisation*. However, it only describes their syntax. The Common Behavior (Instances and Links) package gives a partial denotational meaning to the model elements in the core package. For instance, it describes an association between *Classifiers* (classes) and *Instances*. This establishes the connection between the representation of a *Class* and its meaning, which is a collection of instances. The meaning of *Association* (a collection of Object Links) is also given, along with a connection between Association roles and Attribute values.

To illustrate the scope, and to show the potential for realising a compact core semantics, the relevant class diagrams of the two models are shown in the Figures 2.1 and 2.2. Well-formedness rules are omitted for brevity.

An appropriate starting point for a formalisation is to consider these two models in isolation, with the aim of improving the rigour with which the syntax of UML model elements are associated with (or mapped to) their denotations.

4. FILLING THE SEMANTIC GAP

In this section, we illustrate how the outlined formalisation approach is applied to a small part of the core model. The modelling concept that will be investigated is generalisation/specialisation.

4.1 DESCRIPTION

In UML, a generalisation is defined as “a taxonomic relationship between a more general element and a more specific element”, where “the more specific element is fully consistent with the more general element” [OMG99], page 2-34 (it has all of its properties, members, and relationships) and may contain additional information.

Closely related to the UML meaning of generalisation is the notion of direct and indirect instances: This is alluded to in the meta-model as the requirement that “an instance is an indirect instance of ... any of its ancestors” [OMG99], page 2-56.

UML also places standard constraints on subclasses. The default constraint is that a set of generalisations are disjoint, i.e. “(an) instance may have no more than one of the given children as a type of the instance” [OMG99], page 2-35. Abstract classes enforce a further constraint, which implies that no instance can be a direct instance of an abstract class.

We now examine whether these properties are adequately specified in the UML semantics document.

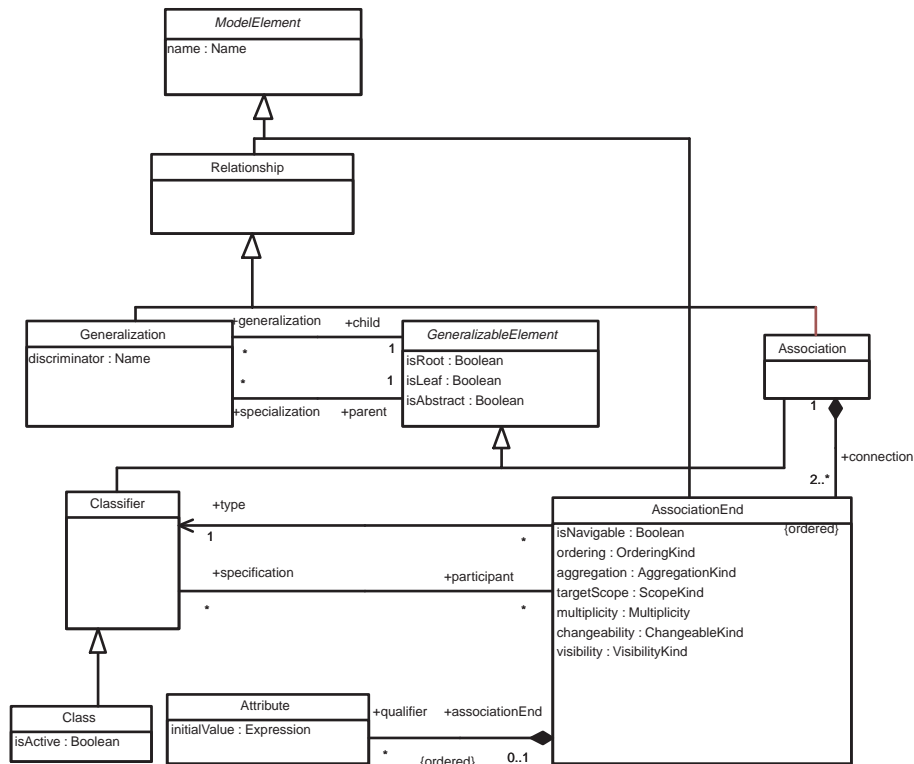


Figure 2.1 Fragment of the core relationships package

4.2 EXISTING FORMAL DEFINITIONS

France et al. [BR98] have defined a formal model of generalisation that fits very well with that adopted in UML. Classes are denoted by a set of object references, where each reference maps to a set of attribute values and operations. Generalisation implies inheritance of attributes and operations from parent classes (as expected). In addition, class denotations are used to formalise the meaning of direct and indirect instances, disjoint and abstract classes. This is achieved by constraining the sets of objects assigned to classes in different ways depending on the roles the classes play in a particular generalisation hierarchy. For example, assume that a_i is the set of object references belonging to the class a , and b and c are subclasses of a . Because instances of b and c are also indirect instances of a , it is required that $b_i \subseteq a_i$ and $c_i \subseteq a_i$, where b_i and c_i are the set of object references of b and c . Thus, a direct instance of b or c must also be an *indirect* instance of a . A direct instance is also distinguishable

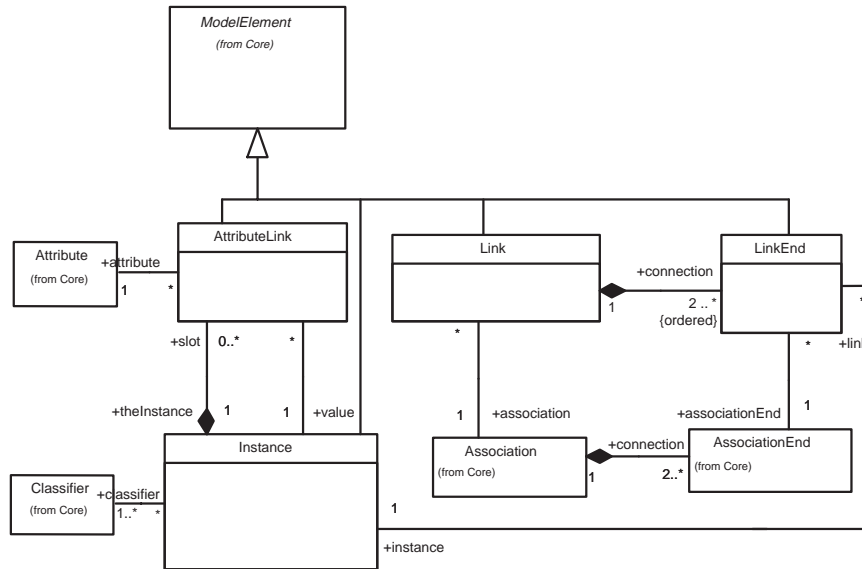


Figure 2.2 Fragment of the common behaviour package

from an indirect instance if there does not exist a specialised class of which it is also an instance.

This model also enables constraints on generalisations to be elegantly formalised in terms of simple constraints on sets of object references. In the case of the standard ‘disjoint’ constraint on subclasses, the following must hold: $b_i \cap c_i = \emptyset$, i.e. there can be no instances belonging to both subclasses. For an abstract class, this constraint is further strengthened by requiring that b_i and c_i partition a_i . In other words, there can be no instances of a , which are not instances of b or c . Formally, this is expressed by the following constraint: $b_i \cup c_i = a_i$.

We adopt this model in order to strengthen the existing meta-model definition of generalisation, and therefore result in a precise definition of the generalisation/specialisation modelling element.

4.3 SYNTAX AND WELL-FORMEDNESS

The abstract syntax of generalisation is described by the meta-model fragment in Figure 2.3 of the core relationships package:

The most important well-formedness rule which applies to this model element, and is not already ensured by the class diagram, is that circular inheritance is not allowed. Assuming `allParents` defines the transitive closure of the relationship induced by `self.generalization.parent`, which happens to be the set of all ancestors, then it must hold that:

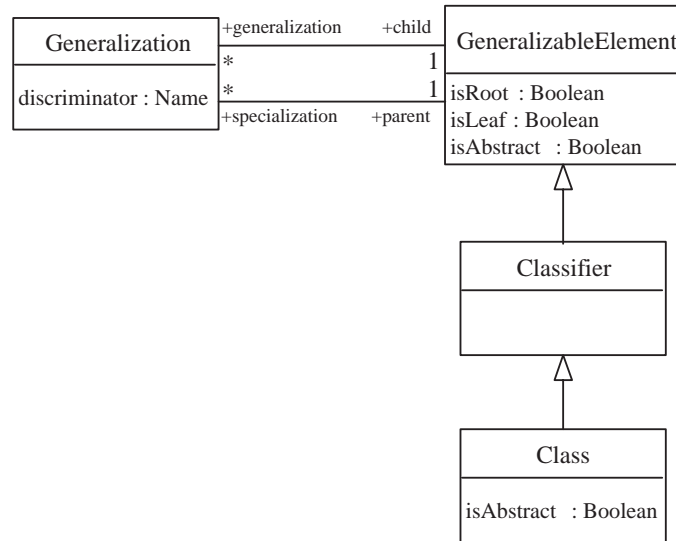


Figure 2.3 Fragment of the core relationships package

```

context GeneralizableElement
not self.allParents -> includes(self)

```

4.4 SEMANTICS

The completeness of the semantic formalisation vs. the desired properties of generalisation is now examined. We concentrate on determining whether the following properties of generalisation are captured in the meta-model:

- direct and indirect instantiation of generalised classes;
- disjoint constraints on sub-classes;
- abstract classes.

As noted in Section 3., the UML meta-model already describes a denotational relationship between *Class* and *Instance*. The meta-model fragment in Figure 2.4 describes this relationship.

However, unlike the formal model described above, the UML meta-model does not describe the constraints that generalisation implies on this relationship. For example, an *Instance* can be an instance of many classes, yet there are no constraints that the classes are related. Thus, the meta-model must be strengthened with additional constraints on the relationship between model elements and their denotations.

4.5 MODEL STRENGTHENING

The first constraint relates to the meaning of direct and indirect instances.

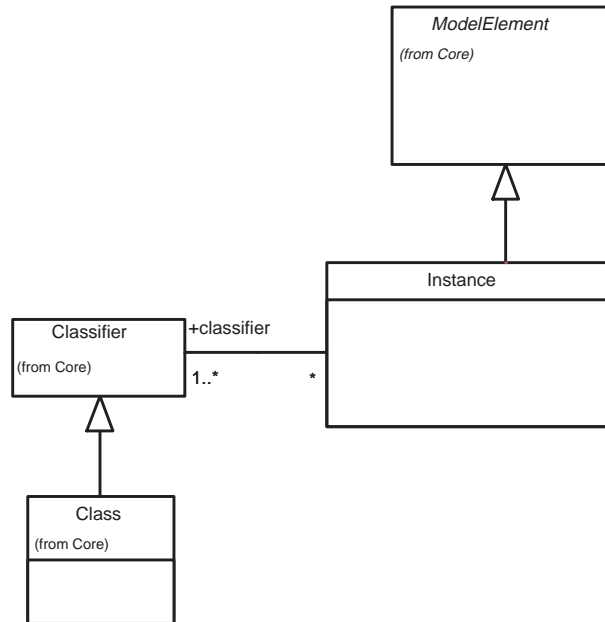


Figure 2.4 Meta-model fragment for Class and Instance relationship

The fact that a direct instance of a class is also an *indirect* instance of the superclass of the class is specified as follows:

```

context c : Class
invariant
  c.generalization.parent -> forall(s : Class |
    s.instance -> includesAll(c.instance))
  
```

This states that all the instances of a class, *c*, are included in the instances of *c*'s superclasses (if any exist).

Unfortunately, this constraint does not guarantee that every instance is a direct or indirect instance of a related class (it only states that generalisation implies the existence of indirect instances).

Thus, an additional constraint must be added in order to rule out the possibility of an instance being instantiated from two or more un-related classes:

```

context i : Instance
invariant
  i.classifier -> exists (direct : Class |
    direct.allSupertypes = i.classifier-Set{direct} )
  
```

The above constraint states that `direct` is a class such that all classes that contain `i` as an instance are superclasses of it.

Therefore the property states that the *only* classes that an object can be instantiated from are the ancestors of the classes that it is directly instantiated from.

Once `direct` and `indirect` instances are formalised, it is possible to give a precise description to the meaning of constraints on generalisations (for example the disjoint constraint described above).

The disjoint constraint can be formalised as follows:

```
context c : Class
invariant
  c.specialization.child -> forall(s1,s2 : Class |
    s1 <> s2 implies s1.instance ->
      intersection(s2.instance) -> isEmpty)
```

This states that for any pair of direct subclasses of a class, `s1` and `s2`, the set of instances of `s1` will be disjoint from the set of instances of `s2`.

Finally, the following OCL constraint formalises the required property of an abstract class that it can not be directly instantiated:

```
context c : Class
invariant
  c.isAbstract implies
    c.specialization.child.instance -> asSet = c.instance
```

Note, the result of the `specialization.child` path is a bag of instances belonging to each subtype of `c`. Applying the `asSet` operation results in a set of instances. Equating this to the instances of `c` implies that all the instances of `c` are covered by the instances of its subtypes. This, in conjunction with the disjoint property above, implies the required partition of instances.

4.6 MODEL EXTENSION

The above definition of the ‘disjoint’ constraint is adequate provided that it applies across all generalisations. However, UML also permits other types of pre-defined constraints, e.g. subclass overlapping, to be applied to generalisations. Unfortunately, the different effects that pre-defined constraints have on a model cannot be formalised. This is because the current definition of a constraint in the UML semantics does not encompass pre-defined constraints. Therefore, extension mechanisms must be introduced in the meta-model as described in Figure 2.5.

Here, pre-defined constraints are modelled as subclasses of *Constraint*. A better definition of the disjoint constraint can now be given:

```
context d : Disjoint
invariant
  d.constrained -> forall(s1,s2 : Class |
    s1 <> s2 implies s1.instances ->
      intersection(s2.instances) -> isEmpty)
```

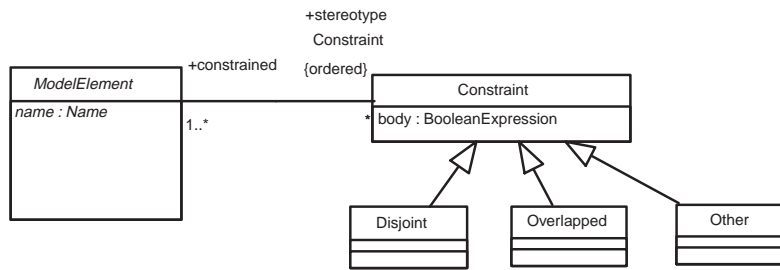


Figure 2.5 Fragment of the meta-model for extension mechanisms

This states that if two or more classes are both constrained to be disjoint, then their instances will be disjoint. Typically this constraint is only applied to direct subclasses of a common superclass. We may therefore define a well-formedness condition for applying disjoint constraints such as the following:

```

context d : Disjoint
invariant
  d.constrained -> forall(s1,s2 : Class |
    s1.generalization.parent = s2.generalization.parent)
  
```

Thus, a more precise description of what is meant by generalisation has emerged. This can be used to gain further insight into the properties of generalisation and be used to compare and contrast it with other models of generalisation. Finally, exploring the UML semantics in a precise way has helped identify areas where the existing meta-model requires extension and clarification.

5. MODEL TRANSFORMATION AS PROOF

The ability to prove properties about UML diagrams is an essential step towards being able to develop provably correct systems in UML. For example, given two models of a system, one that represents its specification, and another that represents its design, it is currently not possible to formally prove the correctness of the design with respect to the specification. This is because there is no notion of a deductive relationship between UML models in the semantics. Such a relationship should set down (precisely) the conditions under which certain properties can be derived from (or checked against) a particular UML model, or under which one UML model can be deemed to be a valid consequence of another UML model. Note that we are primarily concerned here with the meaning of proof as opposed to refinement. This is because refinement conditions can always be expressed in terms of a theorem to be proved.

As an example, consider the following model:

The diagram on the left represents a simple model in which all instances of the abstract class A must be (by definition) either instances of B or C. This is the hypothesis of the proof. The model on the right represents some fact about the original model that

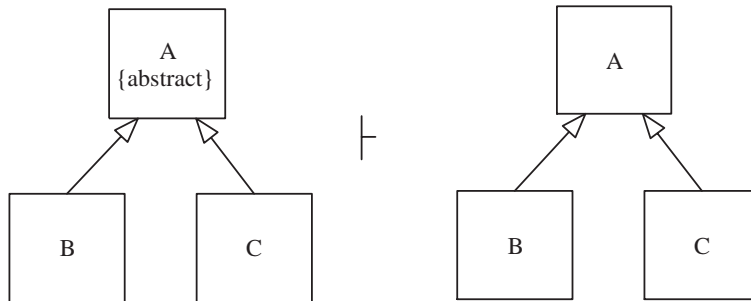


Figure 2.6 Model transformation as a proof of correct realisation

we wish to prove to be valid (the conclusion). In fact, it is the simple property that any set of instances that can be assigned to the abstract class and its children, can also be assigned to the non-abstract class and its children.

Intuitively, this property seems trivially true. If the following assignment of objects is allowed in the model representing the hypothesis: $A = \{a, b\}$, $B = \{a\}$, $C = \{b\}$, it is clear that these objects would satisfy the model representing the conclusion. However, what are the general conditions that make the second model a valid consequence of the first model? Furthermore, can these conditions be described in the core semantic model?

The meta-model extension in Figure 2.7 together with the following constraint illustrates a possible solution (as applied to the generalisation fragment given in Figure 2.4):

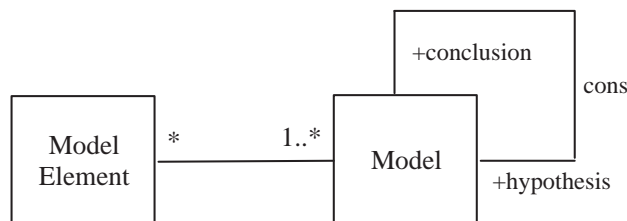


Figure 2.7 Meta-model extension for description of semantic information

```
context m : Model
invariant
  m.modelElement->includesAll(m.hypothesis.modelElement) and
  m.modelElement->select(isOclKindOf(Classifier)) =
    m.hypothesis.modelElement->select(isOclKindOf(Classifier))
```

The consequence association has the property that it separates the meaning of the model from the syntax of the model. It states that every valid set of meanings in a model

representing a hypothesis must be valid in the model representing the consequence. For example, if a particular configuration of objects can be assigned to the classes in the hypothesis, then clearly, a conclusion that also permits the same assignment of objects (or a more liberal one) will represent a valid deduction. Note that syntactical elements, such as classes, remain the same between models, as they do not represent a property of the diagram.

Consider for example that the models in Figure 2.6 were reversed, so that it was to be determined whether the properties of a non-abstract class implied those of an abstract class. This would not be a valid deduction, as the abstract class places an additional constraint on the possible set of objects assigned to itself and its subclasses.

This approach to formalising proof can provide a basis for understanding how transformations between UML models can be used as a basis for diagrammatical proof. For further details the reader is referred to [E98, EFLR98].

6. CONCLUSION

This paper has described ongoing work by members of the precise UML group, who are seeking to develop UML as a precise modelling language. By applying previous knowledge and experience in formalising OO concepts and semantic models, it has been shown how important aspects of the current UML semantics can be clarified and made more precise. A formalisation strategy was also described, with the aim that it will act as a template for exploring further features of UML and for developing new proof systems for the standard language.

Finally, we re-iterate the factors driving the work outlined in this paper. Given the UML's intended role as a modelling notation standard, it is imperative that it has a well-founded semantics. Only once such a semantics is provided can the UML be used as a rigorous modelling technique. Moreover, the formalisation of UML constructs can lead to a deeper understanding of OO concepts in general, which in turn can lead to the more mature use of OO technologies. Such insights can be gained by exploring consequences of particular interpretations, and by observing the effects of relaxing and/or tightening constraints on semantic models.

Acknowledgments

This material is partially based upon work supported by: the National Science Foundation under Grant No. CCR-9803491; the Bayerische Forschungsstiftung under the FORSOFT research consortium and the DFG under the Leibnizpreis program, and the Laboratório de Methodos Formais of the Departamento de Informatica of Pontificia Universidade Catolica do Rio de Janeiro.

References

- [BFG⁺93] M. Broy, C. Facchi, R. Grosu, R. Hettler, H. Hußmann, D. Nazareth, F. Regensburger, O. Slotosch, and K. Stølen. The Requirement and Design Specification Language SPECTRUM, An Informal Introduction, Version 1.0, Part 1. Technical Report TUM-I9312, Technische Universität München, 1993.

- [BHH⁺97] Ruth Breu, Ursula Hinkel, Christoph Hofmann, Cornel Klein, Barbara Paech, Bernhard Rumpe, and Veronika Thurner. Towards a formalization of the unified modeling language. In Satoshi Matsuoka Mehmet Aksit, editor, *ECOOP'97 Proceedings*. Springer Verlag, LNCS 1241, 1997.
- [BR98] J-M. Bruel and R.B.France. Transforming UML models to formal specifications. In *UML'98 - Beyond the notation*, LNCS 1618. Springer-Verlag, 1998.
- [BRJ98] G. Booch, J. Rumbaugh, and I. Jacobson. *The Unified Modeling Language User Guide*. Addison-Wesley, 1998.
- [EFLR98] Andy Evans, Robert France, Kevin Lano, and Bernhard Rumpe. Developing the UML as a formal modelling notation. In Jean Bezivin and Pierre-Allain Muller, editors, *UML'98 Proceedings*. Springer-Verlag, LNCS 1618, 1998.
- [E98] A. S. Evans. Reasoning with UML class diagrams. In *WIFT'98*. IEEE Press, 1998.
- [FELR98] R. France, A. Evans, K. Lano, and B. Rumpe. The UML as a formal modeling notation. *Computer Standards & Interfaces*, 19, 1998.
- [OMG99] Object Management Group. OMG Unified Modeling Language Specification, version 1.3r2. found at: <http://www.rational.org/uml>. 1999.
- [PUML99] The pUML Group. The precise UML web site: <http://www.cs.york.ac.uk/puml>. 1999.
- [RJB99] J. Rumbaugh, I. Jacobson, and G. Booch. *The Unified Modeling Language Reference Manual*. Addison-Wesley, 1999.
- [S86] D. A. Schmidt. *Denotational Semantics: A Methodology for Language Development*. Allyn and Bacon, 1986.
- [S92] J.M. Spivey. *The Z Reference Manual, 2nd Edition*. Prentice Hall, 1992.

About the Authors

Andy Evans has taught and researched in the area of formal methods and their application to object-oriented and real-time systems. He is co-founder of the precise UML group and forthcoming co-chair of UML'2000. He has authored and co-authored papers on formalising UML and object-oriented standards.

Robert France is currently an Associate Professor in the Computer Science Department at Colorado State University. Currently, his primary research activities revolve around the formalization of object-oriented modeling concepts and the development of rigorous software development techniques.

Kevin Lano has carried out research and development using formal methods both in industry and academia. He is the author of "Formal Object-oriented Development" (Springer, 1995) and "The B Language and Method" (Springer, 1996). His current research is on the integration of formal methods and safety analysis techniques, and on the formalisation of UML.

Bernhard Rumpe has taught and supervised research in the area of object-oriented modelling and programming, formal methods and embedded systems. His work includes refinement and composition techniques for structural as well as behavioral notations, methodical guidelines, and the development of formalisation approaches for UML. He co-authored and co-edited three books. He is program chair of UML'99.