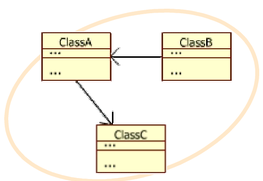
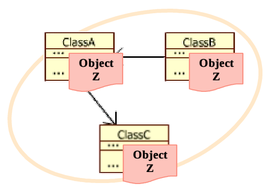
Experimenting Formal Methods through UML



Model the UML class diagram that represents your system static structure.

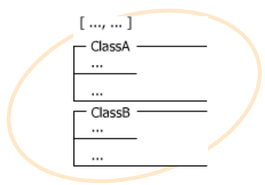
For this purpose, you need to get the Rational Rose CASE tool and our *RoZe Add-In*, which can be found at ***www.cin.ufpe.br/~lmf/roze***.



Add the Object-Z constraints using the $LaTeX$ Object-Z macros.

This includes:

- the types of attributes and of operation parameters;
- class invariants, pre- and post-conditions of operations.



Generate the specification according to the desired analysis (*Z* or *Object-Z*).

Go to the *Tools ® RoZe* menu in the Rose tool and make a choice:

- *Generate Object-Z;*
- *Z Property Checker;*
- *Z Refinement*.

Run *Z/EVES* or *Wizard* using the start button in the dialog according to the previous step.

Only for *Z* choice is necessary that you have the *Z/EVES* installed and configured in the *Tools ® RoZe ® Configuration*.

Check the results.

If there is a problem with the specification, go back to the second step and review your model following the tool response.

UML Overview

The Unified Modelling Language (UML) is the standard graphical notation provided by the Object Management Group (OMG) and currently the most popular notation to model system requirements.

UML has various diagrams, each one with its own specific purpose. Here, we use UML class diagrams, that are provided for analysis and design of the static aspects of systems. In such diagrams, the designer is concerned with classes, interfaces, collaborations, and relationships.

Figure 1 shows a class diagram that models a banking system. This diagram uses almost all elements of UML class diagrams: classes, associations, and inheritance.

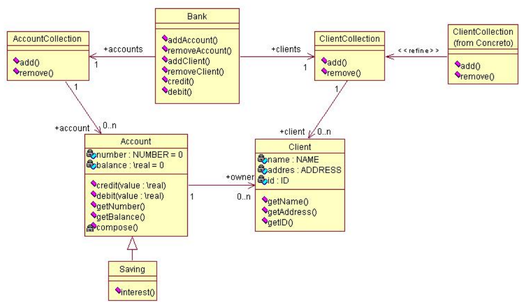


Figure 1: A UML class diagram for a banking system

Annotating UML Diagrams

It is well known that, in a formal language like Object-Z, we can specify pre- and post-conditions of operations as well as class invariants. Although UML cannot specify these conditions, class diagrams are equipped with places to accommodate them and others general constraints as notes attached to any UML graphical element.

Since the present work is oriented towards industrial applications, our approach uses the Rational Rose CASE tool, which allows the designer to insert additional constraints in the diagrams, as defined in UML.

For UML classes we typically annotate its invariant and expected properties. And for every method, we can annotate its pre- and post-conditions, specify whether it changes(Δ) the class state space as well as if such a method is defined in terms of others by means o the Z schema calculus. In associations, we specify its invariant and type; in particular, we annotate the retrieve field when such an association is indeed a refinement relation (see Figure 2).

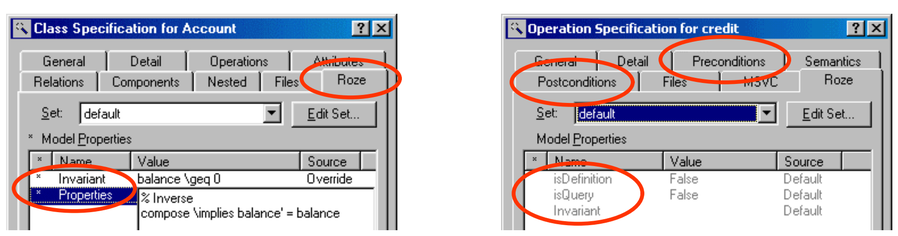


Figure 2: Screen shot of Account class specification.

Mapping to Pure Object-Z

Transforming an annotated UML class diagram into an Object-Z specification is almost direct. A UML class corresponds to an Object-Z class in such a way that the name of the UML class becomes the name of the Object-Z class. UML attributes and invariants are mapped to Object-Z state variables and invariants, and UML methods are captured by Object-Z schemas where the separated pre- and post-conditions in a UML method are put together as the Object-Z schema predicate.

We also need to consider the UML association and generalization relationships. UML associations are captured in Object-Z as class attributes, and the superclass in a UML generalization is mapped as the base class in Object-Z .

With these considerations, we can, for example, obtain an Object-Z class in Figure 3, which represents the Account class in Figure 1.

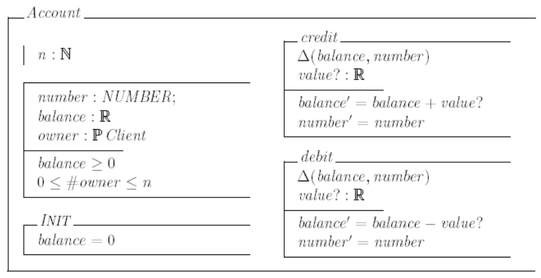


Figure 3: Object-Z specification for Account class

Formal Development

The use of formal methods aims at guaranteeing the overall quality of critical and complex software products. In this respect, the most common tasks to provide means of quality assurance are the proof of desired properties and refinements.

Although, Object-Z tool support is limited to the type-checker Wizard, a model checking strategy, and a theorem prover via an encoding in the HOL system. A single Object-Z class can be seen as a Z specification and thus all tools available for Z can be used for Object-Z. Furthermore, with such an extension, we can apply a refinement calculus and derive code.

Conclusion

In this work, we experience annotating UML class diagrams with fragments of the Object-Z specification language, providing means of type-checking, proof of properties and data refinements. This is an initial effort in the direction of the project ***ForMULa,*** which is supported by the ***Laboratory of Formal Methods (LMF)***.

Future Work. As future research, we intend to extend our graphical notation to **UML-RT**, a UML extension able to model real-time systems, as well as our formal language to ***OhCircus,*** in order to capture static and dynamic aspects of systems simultaneously, and providing a uniform way of deriving program code from specifications.

Acknowledgements. We would like to thank Augusto Sampaio for his comments on earlier versions of this work and Paulo Moura regarding participation on earlier versions of the tool.

Joabe Júnior Rafael Borges Rafael Duarte Alexandre Mota
Centre of Informatics — Federal University of Pernambuco
P.O.Box 7851 — Cidade Universitária
50740-540 — Recife - PE - Brazil