An Overview of RoZ :
A Tool for Integrating UML and Z Specifications

Sophie Dupuy, Yves Ledru, and Monique Chabre-Peccoud

BP72 –38402 Saint-Martin d'Hères Cedex – France
Sophie.Dupuy@imag.fr

Abstract. This paper presents an approach and a tool to increase specification quality by using a combination of UML and formal languages. Our approach is based on the expression of the UML class diagram and its annotations into a Z formal specification. Our tool called RoZ supports this approach by making the transition between the UML world and the Z world : from an annotated class diagram, it automatically generates a complete Z specification, the specifications of some elementary operations and some proof obligations to validate the model constraints.

1 Introduction

Analysis and design methods for information systems propose graphical representations in order to express the structural, dynamic and functional aspects of a system. These aspects are described using models, each model representing a view of the studied domain. Graphical models facilitate communication with users by presenting a synthetic view of the system. But they are imprecise and incomplete. First their semantics can be ambiguous. This is for instance the case for composition in object-oriented modelling [1]. Moreover all the constraints compelling a system cannot be described graphically. So a model is generally complemented by constraints expressed in natural language. This prevents from consistency checking between the model and its constraints.

Recently, the Unified Modelling Language, UML [2], has been standardised by the OMG (Object Management Group). Like previous object-oriented notations, UML does not enable to express all integrity constraints. Thus some complementary proposals have been done to better integrate constraint expressions in UML. The first one [3] proposes to extend the meta-model adding to each class a part where constraints are written. The second one, the Object Constraint Language (OCL [4]) adds constraints expressed in first order logic to the diagrams. This language makes it possible to express constraints precisely and without ambiguity. Unfortunately, current OCL tools do not exploit this preciseness. Therefore we propose to take advantage of formal method tools to help verify the constraints of such annotated diagrams. Formal languages have precise notations based on mathematical concepts which enable to do proofs on specifications. Their use can help to increase information system quality by providing more precise and consistent models. But as writing formal specifications seems often difficult and tedious, we propose an approach and a tool generating automatically from an annotated UML diagram formal specifications in Z [5] and proof obligations.

Our tool, called RoZ, is integrated in the Rational Rose environment. It allows the integration of data specifications expressed in UML with formal annotations in the Z language. Starting from an annotated specification, the tool automatically produces a formal specification by translating the UML constructs and merging them with the annotations expressing predicates not included in the graphical design.

RoZ can also complete the UML class diagram by automatically generating in the UML environment Z specifications of elementary operations on the classes (attributes modifications, adding and deleting objects of the class). The tool also allows to record a guard for each operation and can generate corresponding proof obligations to show that guards are consistent with the actual pre-conditions of these operations. The Z-EVES prover [6] (from ORA) can be used to discharge these proof obligations.

The remainder of the paper is organised as follows. In section 2, an overview of our approach and the principles of our tool RoZ is presented. Section 3 proposes a guided tour of our tool. Section 4 introduces related work. Finally, section 5 draws the conclusions and perspectives of this work.

2 RoZ

2.1 Overview of the Approach

The proposed approach (Fig. 1) uses the complementarity of UML diagrams and the constraints annotating these diagrams to produce a formal specification :

1. From the initial problem, a UML specification is developed. The UML class diagram is edited as usual in the Rational Rose tool ([7, 8]) and complemented with annotations that state several constraints on the data and operation specifications. The annotations must be written in a formalism using first order logic and set theory in order to be expressible in Z. Actually they could be written in OCL.
2. This diagram is the starting point of a translation process which produces a formal specification skeleton. This translation gives a Z semantics for the UML diagram. Many translations from object-oriented notations into formal notations ([9, 10, 11, 12]) have already been proposed. But using a tool guarantees the strict correspondence between the UML model and the Z skeleton. In this paper, we will present the RoZ tool which automatically produces Z specification skeletons based on the translation rules for the main concepts (class, operation, association, inheritance, aggregation and composition) of class diagrams presented in [13].
3. The formal skeleton is completed with the annotations. This can also be done automatically by RoZ thanks to a classification of annotations presented in [14].
4. The formal specification is used to realise consistency checks with a Z prover, Z-EVES. First, the mutual consistency of the constraints can be checked. Secondly, operation guards can be designed to verify that operations do not violate constraints ([15]). The theorems to prove the validity of operations guards can be automatically generated by RoZ. Thanks to the translation process, this reasoning done on the formal specification can be reported to the UML diagram and to its annotations.

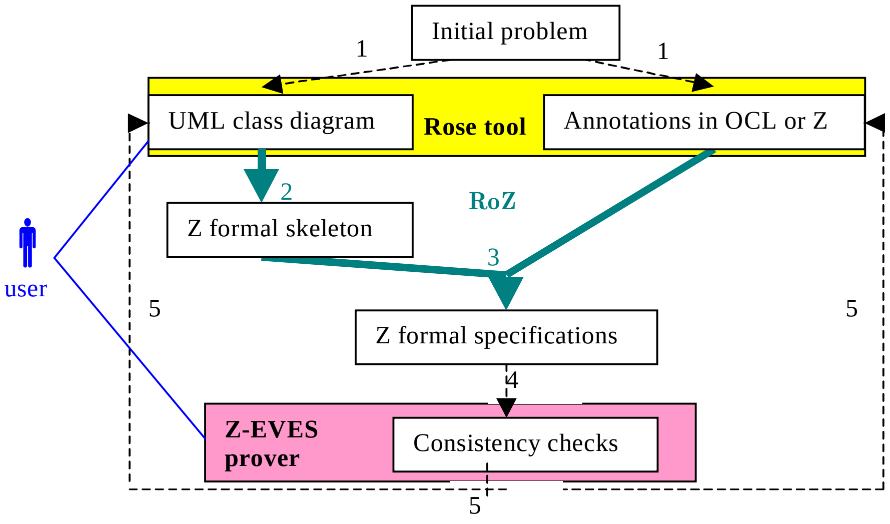


Fig. 1. the proposed approach

2.2 RoZ Principle

RoZ is an extension of the Rational Rose tool to integrate UML and Z notations. The Rose environment is used to build the UML models which are completed by formal annotations in Z. The annotations are expressed into forms. A form should contain all the information mandatory to complete Z skeletons. So each form corresponds to a kind of annotation. It is attached to the class diagram element that it complements. In the Rose environment, a pre-defined form is associated to each element of a diagram. We would like to modify these forms in order to add the fields needed to complete formal specifications. But this being impossible in Rose 4.0, we use the standard Rose forms.

From a class diagram and the information contained in its forms, RoZ uses Rose scripts to :

- *generate Z specifications*

From a UML class diagram and its annotations, a complete Z specification can be generated in a file. We choose to express Z specifications in the Latex style in order to be able to use Z tools such as Z-EVES.

- *generate elementary operations*

The main idea is that for some concepts, there are operations which always exist. For instance, a class has always operations to modify its attributes. Moreover if the class is concrete, there are also operations to add or remove an instance to or from the class instances. So for concrete classes, the specification of these operations can be generated automatically.

- *generate theorems to validate operation guards*

From a UML class diagram, elementary operations and a Z formal specification can be generated automatically. Data must often respect constraints which are not taken into account by the code generated automatically and only implicitly taken into account in the specification. So it is interesting to ``design" guards which will be evaluated before the operation execution and which can avoid to execute an operation violating the constraints. From the Z specification produced from the UML diagram, it is possible to prove that the proposed guards guarantee data integrity. So for each identified guard, a theorem is generated ([15]). The theorems demonstration uses a semi-automatic proof tool Z-EVES.

3 Guided Tour of RoZ

In this section, the functionalities of RoZ are illustrated on a simplified version of an access control system. We show how to describe a class diagram and its annotations in RoZ. Then we use RoZ to generate some elementary operations, to generate a complete Z specification from the diagram and its annotations and to generate theorems to validate operation guards.

3.1 Construction of a Class Diagram Example in RoZ

To illustrate the use of RoZ, we describe a simplified version of the access control system presented in [16]. This model (Fig. 2) features two classes : “PERSON” and “GROUP”. Each person has four attributes: his last and first names, the set of his telephone numbers and the number of his magnetic card. Each group is characterised by a name (e.g. “Accounting Department”) and a code (e.g. “AccD”). One can note that the types of the attributes are expressed in the Z style. For instance, the “tel” attribute is multi-valuated i.e. it corresponds to a set of telephone numbers (\finset tel).

The PersonGroupRel association links each person to one group.

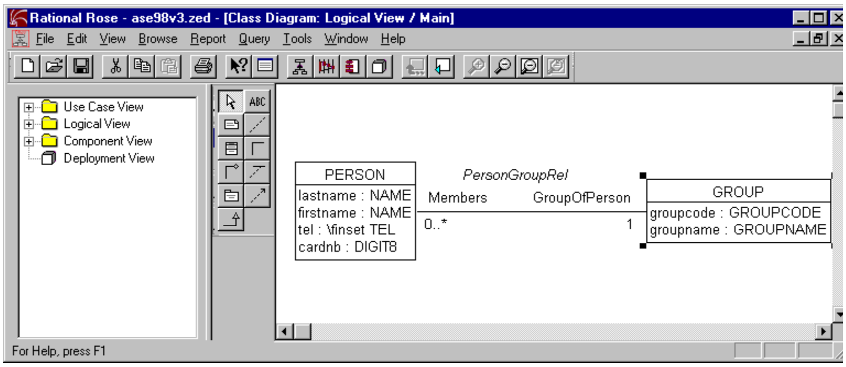


Fig. 2. Rose interface - the access control class diagram

This class diagram does not express the full specification of the application data. Three additional constraints complement this diagram :

1. Every person has at least one telephone number.

2. The card number is a key of persons.
3. The telephone numbers of the members of a given group have the same prefix.

The UML class diagram of the access control system is built in a class diagram window of the Rose environment. This diagram can be complemented by the expression of the constraints. The constraints are expressed in the Z Latex syntax in the “Documentation” field of the forms.

The first constraint (every person has at least one telephone number) holds on the attribute “tel”. So it is written in the field “Documentation” of the attribute “tel” (Fig. 3). It is expressed in Z that the set of telephone numbers cannot be empty :

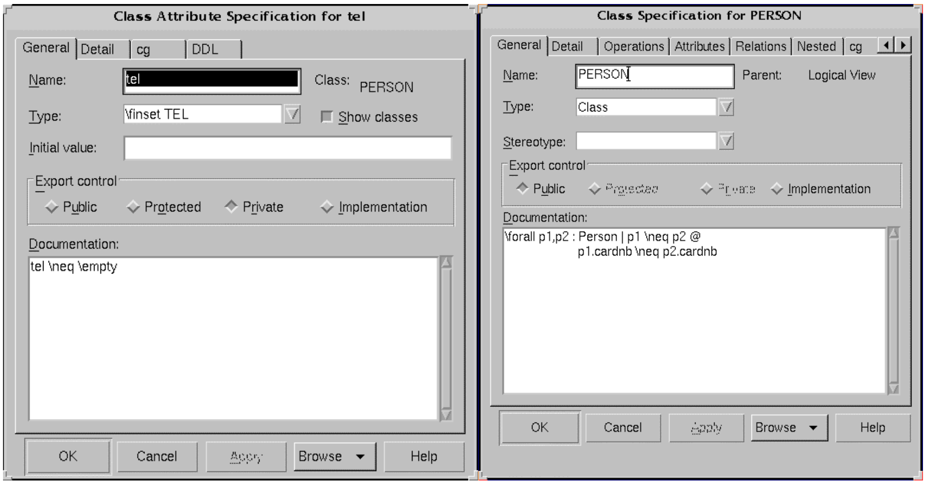
```
tel ≠∅
```

This is expressed in the Z Latex style:

```
tel \neq \empty
```

The second constraint (the card number is a key of persons) means that two different persons have distinct card numbers. It is a comparison between existing objects of the class, so it is in the field documentation of the class “PERSON” (Fig. 3) :

```
∀p1,p2 : Person | p1≠p2 • p1.cardnb ≠ p2.cardnb
```


Class Attribute Specification for tel

Name: tel
Class: PERSON
Type: \finset TEL
Show classes
Initial value:
Export control: Public / Protected / Private / Implementation
Documentation:
tel \neq \empty

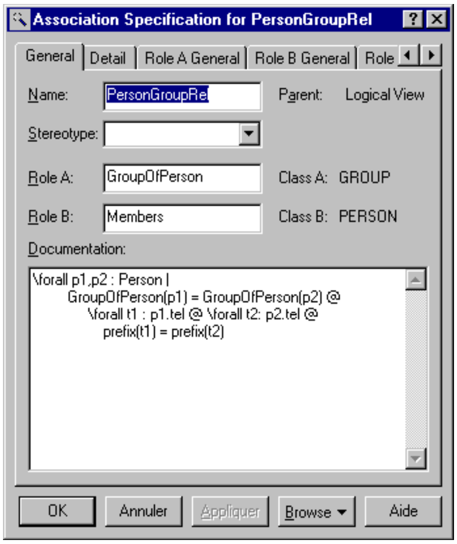
Class Specification for PERSON

Name: PERSON
Parent: Logical View
Type: Class
Stereotype:
Export control: Public / Protected / Private / Implementation
Documentation:
\forall p1,p2 : Person | p1 \neq p2 @
p1.cardnb \neq p2.cardnb

Fig. 3. the forms for “tel” and “Person”

Finally we have to express the third constraint (the telephone numbers of the members of a given group have the same prefix.). This constraint is related to the relationship between “PERSON” and “GROUP”, so it is expressed in the “Documentation” field of “PersonGroupRel” (Fig. 4). Let us consider that we have defined a “prefix” function which gives the prefix of a telephone number. The constraint states that for two persons which belong to the same group, each telephone number of their sets of telephone numbers have the same prefix :

```
∀p1,p2 : Person |
          GroupOfPerson(p1)=GroupOfPerson(p2) •
           ∀t1 : p1.tel • ∀t2 : p2.tel •
            prefix(t1) = prefix(t2)
```



Name:	PersonGroupRel	Parent:	Logical View
Stereotype:			
Role A:	GroupOfPerson	Class A:	GROUP
Role B:	Members	Class B:	PERSON

Documentation:

```
\forall p1,p2 : Person |
     GroupOfPerson(p1) = GroupOfPerson(p2) @
       \forall t1 : p1.tel @ \forall t2: p2.tel @
          prefix(t1) = prefix(t2)
```

Fig. 4. the form «PersonGroupRel»

As we illustrate it for constraints on classes and on associations, each form contain a kind of constraints, defining so a constraint classification. This classification covers the main concepts of a UML class diagram. In particular, we propose different kinds of constraints for inheritance (constraints on the super-class, on the subclass or on inheritance relationship) which can be explicitly expressed in some RoZ forms. Then our translation into Z guarantees that constraints on a super-classes are inherited by its subclasses.

Using the same principle, forms can also be exploited to write operation specifications. This kind of form use is illustrated with the specification of the "ChangeLastname" operation in the next subsection.

This annotated UML diagram will be the basis for the generation of a specification (sect. 3.3).

3.2 Generation of Elementary Operations

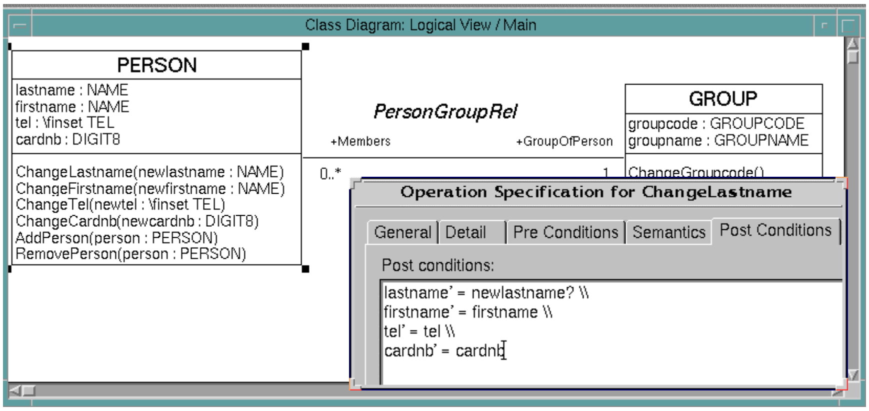
In information systems, some operations associated to classes or associations are very often used. They enable to create or delete an object or a link, or to modify an attribute value. Their systematic generation has been studied in [17, 18].

RoZ uses the same principle to generate the specification of all the elementary operations of the diagram classes : for each attribute of a given class, a modification operation is generated. Moreover if the class is concrete, the specification of operations adding or deleting an object to/from the class objects are added.

The generation of elementary operations does not take into account the eventual constraints on a diagram. In order to check that an operation is consistent with the constraints, we must validate its guard by proving a theorem generated by RoZ (see 3.4)

For example, for the class “PERSON”, the operations “ChangeLastname”, “ChangeFirstname”, “ChangeTel”, “ChangeCardnb”, “AddPerson” and “RemovePerson” are generated automatically by RoZ (Fig. 5). The specification of these operations is contained in their form.

For instance, let us consider “ChangeLastname” which modifies the last name of a person. In the operation part of “PERSON”, one can see the operation signature : “ChangeLastname” has an argument “newlastname” which is the input parameter of the operation. In the “PostConditions” (Fig. 5) tab of the “ChangeLastname” operation, the post condition of the operation is written as a Z Latex style : it means that the new value of the attribute “lastname” is the argument of the operation “newlastname” and that the others attributes of “PERSON” are unchanged by the operation. RoZ also fills the “Semantics” tab with the key-word *intension operation* which signifies that “ChangeLastname” is an operation on the attributes of the class.



```
PERSON
lastname : NAME
firstname : NAME
tel : \finset TEL
cardnb : DIGIT8

ChangeLastname(newlastname : NAME)
ChangeFirstname(newfirstname : NAME)
ChangeTel(newtel : \finset TEL)
ChangeCardnb(newcardnb : DIGIT8)
AddPerson(person : PERSON)
RemovePerson(person : PERSON)

GROUP
groupcode : GROUPCODE
groupname : GROUPNAME

ChangeGroupcode()
```

```
lastname' = newlastname? \\
firstname' = firstname \\
tel' = tel \\
cardnb' = cardnb
```

Fig. 5. Post-condition for «ChangeLastname»

3.3 Generation of a Z Specification

RoZ automatically generates the Z schema skeletons corresponding to a UML class diagram. The Z skeletons are produced by translation of the concepts (class, association, inheritance…) of the diagram. In order to complete these specification skeletons, you must add information like the definition of the types of the attributes and the constraints on the class diagram. In sections 3.1 and 3.2, we have shown how constraints and operation specifications can be included in the RoZ environment.

On the contrary, we did not find a place global to the model to express type definitions in the Rose standard forms. So type are not defined in the RoZ environment, but in a file containing them. In the example of access control system, we have to define the types NAME, TEL, DIGIT8, GROUPCODE and GROUPNAME. Moreover the third constraint (the telephone numbers of the members of a given group have the same prefix.) requires the declaration of the prefix function.

So from the class diagram and the annotations, RoZ generates a complete Z specification. For example, from the class "PERSON" and its annotations, RoZ generates two Z schemas ("PERSON" and "PersonExt"). A Z schema is a data specification structure composed of two parts: the declarations which constitute the local variable lexicon and the predicates expressing constraints on these variables. The "PERSON" schema defines the attributes of the class and the "PersonExt" one describes the set of existing persons. These schemas include the constraints expressed in the "Documentation" fields of "PERSON" and "tel". The constraint on the telephone numbers (constraint 1) is expressed in the Z schema defining attributes, while the key constraint (constraint 2) is defined in the Z schema representing the existing objects of "PERSON".

$PERSON$

$lastname : NAME$
$firstname : NAME$
$tel : \mathbb{F}\, TEL$
$cardnb : DIGIT8$

$tel \neq \varnothing$

$PersonExt$

$Person : \mathbb{F}\, PERSON$

$\forall\, p1, p2 : Person \mid p1 \neq p2 \bullet p1.cardnb \neq p2.cardnb$

3.4 Theorems Generation

At this point, we have produced a Z specification and we can start to investigate the properties of our model. Let us consider the operation "ChangeTel" which changes the value of the "tel" attribute and corresponds to the following instruction :

```
tel := newtel?
```

The automatically generated specification for this operation is :

$$\begin{array}{|l}
\multicolumn{1}{l}{\textit{PERSONChangeTel}} \\
\Delta PERSON \\
newtel? : \mathbb{F}\ TEL \\
\hline
lastname' = lastname \\
firstname' = firstname \\
tel' = newtel? \\
cardnb' = cardnb \\
\hline
\end{array}$$

In this specification, newtel? is the input parameter. The last predicate constraints the new value of the “tel” attribute (denoted by tel’) to be equal to the input parameter. The remaining predicate expresses that the other attributes keep their initial value. The first line of the declaration part ΔPERSON expresses that the effects of this operation are limited to change an object of type “PERSON”.

Its actual pre-condition can be computed from the text of the specification and the constraints listed in the included schemas. When specifications get complex, it is good software engineering practice to state these pre-conditions more explicitly. For the “PERSONChangeTel” operation, we identify the pre-condition

```
newtel? ≠ ∅
```

This pre-condition depends only on the constraint concerning the “tel” attribute because the “PERSONChangeTel” operation only changes an object of type “PERSON” and is not promoted at the level of existing objects of “PERSON”. So we do not have to take into account constraints on the existing objects of “PERSON” or on the association with “GROUP”. These constraints will be considered for alternate versions of this operation which promote “PERSONChangeTel” at the level of the existing persons and for the operations on the “PersonGroupRel” association. For these alternate versions, the precondition become more complex in order to consider the key of " PERSON" and the telephone number prefix constraint.

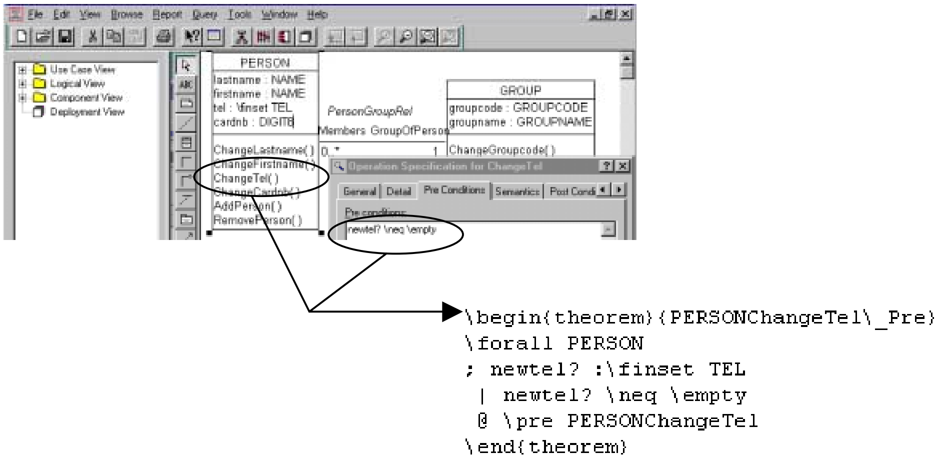
We guess that the pre-condition (`newtel? ≠ ∅`*)* is strong enough to imply the actual pre-condition of the operation. This predicate states that the new set of telephone numbers must not be empty. We add it to the field “Pre Conditions” of the operation “ChangeTel”:

```
newtel? \neq \empty
```

Then RoZ is used to generate a theorem to validate proposed pre-conditions for each operation. It uses the information given in the operation form to generate the following proof obligation for “ChangeTel” (Fig. 6):

```
\begin{theorem}{ PERSONChangeTel \_Pre}
          \forall PERSON ;newtel? : \finset TEL |
             newtel? \neq \empty @
               \pre PERSONChangeTel
\end{theorem}
```

It means that the proposed pre-condition (`newtel? □ □`) is stronger than the computed pre-condition (`pre PERSONChangeTel`).



```
\begin{theorem}{PERSONChangeTel\_Pre}
\forall PERSON
; newtel? :\finset TEL
 | newtel? \neq \empty
 @ \pre PERSONChangeTel
\end{theorem}
```

Fig. 6. Generation of theorems

If you want to prove the theorems generated, you can use any Z standard prover. We use the Z-EVES theorem prover because it offers a high level of proof automation. For instance, the pre-condition theorem for “ChangeTel” is proved automatically using the following commands :

```
try lemma PERSONChangeTel\_Pre;

prove by reduce;
```

which demonstrates that the proposed pre-condition is correct.

4 Related Work

Two kinds of environments have been proposed to couple different kinds of notations. In the first one [19], the various specifications are developed independently and some consistency rules are defined to check the consistency between them. These tools have the advantage of being formalism-independent. But the consistency rules do not guarantee that the specifications are semantically equivalent. So if you use such tools to couple UML and formal specifications, you cannot be sure that the formal specifications are equivalent to the UML ones. The reasoning done on the formal specifications cannot be reported to the UML ones. The other kind of tools are based on some translation from a formalism to another one so as to have equivalent versions of the specifications. As this is the approach we have chosen, the following paragraphs develop the characteristics of such tools and a comparison with ours.

Some tools are based on meta-CASE such as GraphTalk ([20, 21, 18]). A meta-CASE is a tool intended to build other tools. Others, like ours, propose to extend existing environments. In particular, the Rational Rose tool is used to make the link between UML and VDM++ ([22]) and UML and Z ([23]). Using a meta-CASE enables to build exactly the desired environment. But its development is much longer than the extension of an existing tool and rarely reaches the standard of dedicated environments. Moreover, using a “standard” environment enables to keep close to the practice of conceptual modellers.

Another criterion of comparison is the way of coupling the different kinds of specifications. [21] and [23] propose multi-modelling environments in which the different kinds of specifications are available in the same modelling tool. Graphical and formal specifications are thus developed in parallel. In this approach, the different formalisms used must be mastered to be able to add constraints to formal specifications or to specify operations.

Another approach consists in automatically generating the formal specifications into a file. The main advantage is to have a file that can be used by the tools for formal notations. That is what is proposed in [24]. But [24] does not specify how the formal skeleton generated in the file can be complemented and how the model is updated taking into account the code added to the skeleton.

Our proposal goes on a little further by proposing to perform all the modelling work (diagrams and annotations) in the UML environment. Only the strictly necessary formal specifications are seen to permit their exploitation. This one depends on the level of mastery of the formal language. It is not necessary to know the details of the formal language to have a documentation with formal specification. But the verification of the formal specifications by proving operation pre-conditions for instance, requires a higher degree of expertise.

Moreover, tools like [22, 23] allow modifications of the model from both the UML and the formal method environments. We have chosen to use Rose as the only interface to the model which means that any update of the Z specifications should be done by modifying the corresponding model or annotations.

5 Conclusion and Perspectives

In this article, we present RoZ, an automate tool for specification and illustrate its use on a simple example. The RoZ tool aims at increasing system quality by adding precision to the system specification. It exploits the standard Rose environment to make UML notations and formal annotations live together: the class diagram provides the structure of Z formal skeleton while details are expressed in forms attached to the diagram. RoZ offers the following possibilities:

- automatic generation of Z formal specifications from an annotated UML class diagram,
- generation in Rose of the specifications of elementary operations on classes,
- generation of proof obligations to validate operation guards.

5.1 Advantages

An important characteristics of RoZ is that all the modelling work (diagrams and annotations) is realised in Rose to be as close as possible to the usual practitioner working environment. So the environment is employed as usual and some information can complement the diagrams when necessary. Our ambition is that it does not require to have a deep understanding of Z to be able to have complete Z specification. In particular, we expect that many constraints can be expressed be adapting similar ones from other RoZ diagrams (see 3.2).

The main advantage of RoZ is that it automatically realises some tedious work. First, it generates formal specifications from annotated UML diagrams. Secondly, it creates elementary operations on classes. Finally, it generates proof obligations. This automation limits human efforts of designers to write precise specifications and to check them.

Moreover RoZ offers guidance to know where to express each kind of annotation. In particular, the definition of the tool gives rise to a classification of integrity constraints on class diagrams. So each kind of constraints is expressed in a specific form. This helps the user to know where to write constraints and how to complement a class diagram.

Finally, it is important to have synchronised evolutions of the UML and Z specifications. Our tool allows to always have equivalent versions of the UML and the Z specifications. As all the modelling work is made in the UML environment, a specification change is made in this environment, and the corresponding Z specification is generated again accordingly. The consistency of the constraints may be checked against this new version and corrected.

The classification of annotations and the synchronised versions of UML and Z specifications facilitate specification evolutions by providing a clear framework about where changes must be made.

5.2 Future Work

Currently, RoZ only considers static constraints. It does not enable to express dynamic constraints. This is due to the fact that Z is intended to express invariant constraints. In order to take into account other constraints, RoZ must be extended to translate dynamic aspects of UML diagrams into other formal languages such as Lustre [25], more appropriate for dynamic aspects.

Moreover annotations written in the prototype are currently in the Z Latex style which is not intuitive. But all annotations expressed in a formalism close to first order logic could be easily integrated into Z specifications. Particularly using OCL can be a way to get closer to practitioner uses. As OCL and Z are close formalisms, annotations could be written in OCL and automatically translated into Z before their integration in the Z skeletons. So RoZ can be an easily usable tool for people knowing UML and its constraints language, OCL.

We can also imagine to simplify the writing of constraints by developing an interface which would propose a list of “standard” constraints. For instance, one could choose in the list, the key constraint. Then one would just have to give the name of the key attribute(s) and the corresponding constraint would be automatically written in Z. This knowledge would enable to produce more relevant elementary operations, by avoiding for instance to generate operations modifying a key attribute. This could also be avoided by extending Rose to express more characteristics about attributes such as the modification permission defined in [26].

Another way of improving the generation of elementary operation could be to consider elementary operations of other concepts. As we do it for classes, we can generate the elementary operations of associations. This would enable to consider dependencies between classes related to association cardinalities. For example, if the cardinality of the association between the A and B classes is 1, this means that if you create (res. delete) an object, a, of A, you must create (res. delete) an object of B linked to a.

Finally, proof is a very important task, even if it is a difficult one, since it can reveal errors or unsuspected problems. Currently RoZ generates proof obligations, but it does not provide any help in order to prove them. This help can be to increase proof automation by using domain knowledge. We are currently working on the definition of proof patterns for elementary operations in order to help the designer to validate the system.

Although RoZ is only at its initial development step, we hope that it can bring a useful help in the improvement of specifications quality.

Acknowledgments

We are grateful to the SIGMA and the PFL research teams at Grenoble, in particular to Olivier Maury for his comments on the RoZ use. Thanks are also due to José-Celso Freire from UNESP (São Paulo, Brasil) for his useful suggestions on the documentation and the distribution of RoZ. Finally we would like to thank the anonymous reviewers for their positive and detailed comments and feedback.

References

1. J. Brunet. An enhanced definition of composition and its use for abstraction. The 5th international conference on Object Oriented Information Systems, Paris, September 1998.
2. Rational Software Corporation. UNIFIED MODELING LANGUAGE – notation guide version 1.1. September1997.
3. Y. Ou. On Using UML Class Diagram for Object-Oriented Database Design – Specification of Integrity Constraints. Proc „ UML „'98 Beyong the Notation, Mulhouse, France, June 1998.
4. J. Warmer and A. Kleppe. The Object Constraints Language. Addison-Wesley, 1998.
5. J.M. Spivey. The Z notation – a reference manual (second edition). Prentice-Hall International, 1992.
6. M. Saaltink. The Z/EVES system. In j. Bowen, M. Hinchey and D. Till editors, Proc. 10th Int. Conf. On the Z formal method (ZUM), volume 1212 of Lecture Notes in Computer Science, pages 72-88, Reading, UK, April 1997.
7. Rational Software Corporation. Rational Rose – Using Rational Rose 4.0. 1996.
8. Rational Software Corporation. Rational Rose – Using Rational Rose 98. 1998.
9. R. France and JM. Bruel and M. Larrondo-Petrie and M. Shroff. Exploring the Semantics of UML type structures with Z. Proc. 2nd IFIP Workshop on Formal Methods for Open Object-Based Distributed Systems (FMOODS), 1997.
10. M. Shroff and R.B. France. Towards a Formalization of UML class structures in Z. COMPSAC'97, 1997.
11. K. Lano, H. Houghton, P. Wheeler. Integrating Formal and Structured Methods in Object-Oriented System Development. In Formal Methods and Object technology, Chapter 7, Springer, 1996.

12. P. Facon, R. Laleau Laleau and H.P. Nguyen. Mapping Object Diagrams into B Specifications. Proc of the Method Integration Workshop, Springer-Verlag, Leeds, March 1996.
13. S. Dupuy, Y.Ledru and M. Chabre-Peccoud. Vers une intégration utile de notations semi-formelles et formelles : un expérience en UML et Z. revue l'Objet, numéro spécial sur les méthodes formelles pour les systèmes à objets, 2000, to appear.
14. S. Dupuy. RoZ version 0.3 an environment for the integration of UML and Z. technical report Laboratoire LSR-IMAG, 1999. http://www-lsr.imag.fr/Les.Groupes/PFL/RoZ/index.html
15. Y.Ledru. Identifying pre-conditions with the Z/EVES theorem prover. Proc Int. IEEE Conference on Automated Software Engineering'98, IEEE Computer Society Press, October 1998.
16. Y. Ledru. Complementing semi-formal specifications with Z. KBSE'96, IEEE Computer Society Press, 1996.
17. N. Hadj-Rabia and H. Habrias. Formal specification from NIAM model : a Bottom-Up Approach. ISCIS XI (11th int. Symposium on Computer and Information Sciences), Antalya, Turkey, November 1996.
18. H.P. Nguyen. Dérivation de Spécifications Formelles B à Partir de Spécifications Semi-Formelles. PhD thesis, CNAM, December 1998.
19. B. Nuseibeh and A. Finkelstein. ViewPoints : a vehicle for method and tool integration. IEEE Proc. Of the 5th Int. Workshop on CASE (CASE'92), p50-60, Montreal, Canada, July 1992.
20. R. Laleau, N. Nadj-Rabia. Génération automatique de spécifications VDM à partir d'un schéma conceptuel de données. Dans Actes du 13ème congrès INFORSID, Grenoble, France, June 1995.
21. J.C. Freire Junior, M. Chabre-Peccoud, A. Front and J.-P. Giraudin. A CASE Tool for modeling of methods and information systems. OOIS'98, Int. Conference on Object-Oriented Information Systems, Springer, Paris, France, September 1998.
22. IFAD. the Rose-VDM++ Link. http://www.ifad.dk/Products/rose-vdmpp.htm
23. Headway Software. RoZeLink 1.0. http://www.calgary.shaw.wave.ca./headway/index.htm
24. R.B. France, J.-M. Bruel, M.M. Larrondo-Petrie. An Integrated Object-Oriented and Formal Modeling Environment. Journal of Object Oriented Programming, November/December 1997.
25. P. Caspi, N. Halbwachs, D. Pilaud and J. Plaice. LUSTRE, a declarative language for programming synchronous systems. In 14th Symposium on Principles of Programming Languages (POPL'87), ACM, p 178-188, Munich, 1987.
26. G. Booch, I. Jacobson and J. Rumbaugh. The Unified Modeling Language- User Guide. Addison-Wesley, 1998.