An Automatic Transformation of Event B Models Into UML Using an Interactive Inference Engine THINKER

Leila Jemni Ben Ayed, Mohamed Nidhal Jelassi

Research Unit of Technologies of Information and Communication
ESSTT, 5, Avenue Taha Hussein, P.B. : 56, Bab Menara, 1008 Tunis, Tunisia
Faculty of Science of Tunis

Abstract. In this paper, we describe Thinker, which can be considered as an interactive based-rules inference engine that supports inference rule where for some selected concepts we can have different results. This is the case of interactive transformation approach generating simultaneously different concepts from initial ones. For example, the transformation of formal notations from semi-formal ones, especially the application of rule-based approach translating B abstract machines into UML class diagrams. In addition, Thinker allows us to select one solution from a set of proposed solutions and to modify previous selections if there an ambiguous choice. Our inference engine is also generic and can be used in more than one domain. By the translation from B to UML class diagram, we illustrate our tool.

Keywords. Event B, Inference engine, Interactivity, Modeling, UML, Translation

1 Introduction

The complementary characteristic of formal and semi-formal methods has incited several research teams to propose approaches to integrate them. The integration of UML and the B methods has been in particular an interesting topic of several works. These two languages are complementary: UML [8] is a semi-formal method that is more accessible to the untrained user and provides a rich array of concepts that clarify the structure of a system. The B method, on the other hand, is a formal method based on relatively few and precisely defined mathematical concepts, and is very adequate for formal verification. B has been employed in different Security applications, recently Invariant Checking based on event B of Secrecy in Group Key Protocols has been proposed [4]. This distinction is also reflected in existing tool support: tools for UML are centered around graphical editors, they often allow for simulation and code generation, but otherwise offer rather limited (mostly syntactic) analysis techniques. The B method [2] is supported by tools such as Atelier B and BToolkit that prominently include a theorem prover to ensure the correctness of a development. It is therefore desirable to be able to propose a development approach which uses these two

Jemni Ben Ayed L. and Nidhal Jelassi M. (2009).
An Automatic Transformation of Event B Models into UML using an Interactive Inference Engine Thinker.
In *Proceedings of the 7th International Workshop on Modelling, Simulation, Verification and Validation of Enterprise Information Systems*, pages 119-124
DOI: 10.5220/0001998801190124

notations allowing user to go back and forth between B and UML. Previous works have mostly focused on formalizing UML models in formal methods such as B, aiming at the verification of UML designs and, consequently, at the elimination of inconsistencies and ambiguities. One problem with this approach is that such translations, aiming to be as comprehensive as possible, tend to result in unnatural and cluttered B specifications that are hard to understand and reason about. Another problem is the traceability of errors detected at the B level back to the UML model. For this reason, its interesting to propose a process that aims to construct an UML view of B models. The additional structure offered by UML could help to clarify the B specification better than the set theoretic language on which B is based.

Idani and Ledru [5] propose two steps to translate B specification into class diagrams: the first step gives a systematic transformation of a B specification into a class diagram by applying some rules. This diagram contains a lot of classes since they are derived from all the sets presented in the B machine. This work does not allow the generation of classes from different B concepts because it limits itself to abstract sets. The second step joins these classes to the B operations to keep the classes that seem to be the most significant. In the same way, Tatibouet and Hammad [10] propose an automatic transformation process (B2UML) of B machines into UML class and statechart diagrams. However it is not clear when to choose between the transformation into a class diagram or a statechart one since a machine could be derived in two types of diagrams. Because we really think that the parallel development of a class diagram and statechart diagrams is more appropriate, and that because the same concept in B models can have different corresponding results in UML class diagrams, it is not reasonable to hope for a fully automatic translation. We adopt a different solution for the derivation of UML models from B specifications.

Our contribution consists of an interactive transformation approach for generating simultaneously different concepts of class and statechart-UML diagrams from the features of an abstract B specification. We do not constrain the modeler to a specific style of translation. The proposed approach is guided by heuristics which give to the user different transformation possibilities and let him to choose the one that seems to be adequate but which can be overridden by the modeler, subject to integrity constraints that ensure the coherence of the translation. In our previous work [3], we have presented a first step in this direction by giving a rule-based approach for transforming B abstract machines into UML diagrams. We evolve this work by giving in this paper an interactive inference engine THINKER supporting the derivation of UML class diagrams from event B models.

2 Translation Approach

Our approach is composed of three steps. The first one involves the generation of classes from some features of B specification by applying some rules from which it generates attributes, associations, states and other relations (heritance, composition, dependence). The second step enriches this diagram by adding new classes and generating statechart diagrams for certain classes. The third step transforms operations. It depends on the model produced by the preceding steps to generate methods for classes

and transitions of object states. This transformation depends precisely on transformations done on variables used in these operations. For example, if the variables used are transformed in states then the corresponding operation will be transformed into a transition between states obtained from the pre-condition and states obtained from the post-condition. On the other hand, if these variables are transformed into attributes, then B operations will be transformed into operations in the corresponding class in the class diagram. Corresponding translation rules are given in [3].

3 THINKER: An Interactive Inference Engine

THINKER is an interactive inference engine supporting the derivation of UML class diagrams from event B models, which at each step allows us to select one solution from set of proposed solutions if there is an ambiguous choice and to modify previous decisions. Different inference engines have been developed for the construction of rules or object based expert systems such as CLIPS (Johnson, 1994), PROLOG [9] or JESS [7]. Despite the different characteristics of these inference engines, there is a common failure point: the lack of interactivity between the user and the engine in the case of ambiguity between several rules. We need at each derivation step to choose one rule if there are different rules with the same premise and also to backtrack to change previously selected rules. THINKER is an interactive inference engine operating by forward chaining. It employs a top-down method which takes facts as they become available and attempts to draw conclusions (from satisfied conditions in rules) which lead to actions being executed. The knowledge base of an inference engine is composed of a set of production rules *SR* and a set of facts *SF*. The set *SF* is generated from a B model using a developed parser and *SR* is the set of formalized translation rules. After loading *SF* and *SR*, THINKER starts its execution by extracting the set of rules that can be applied: *RuleApp*.

The conflict resolution strategy of THINKER is to fire the rule which the system designer defined first. Forward chaining continues until it encounters an ambiguity, i.e, in *RuleApp* we find several rules with the same set of premises. In this case, the interactivity between THINKER and the user is required. The user decides which rule to apply. In addition, the user is shown the result of each candidate rule before choosing one of them. Note that after each application of a rule, *SF* is incremented with potential new results. The set of facts is presented to the user as a result of his choice. After the first ambiguity, a boolean variable *Reb* which is initialized to 0 changes of value to 1. This is used to know when we must propose to the user the possibility of backtracking. So, when *Reb = 1*, after every application of a rule, Thinker asks the user if he wants to backtrack. If the answer is affirmative, our engine proposes to define the *Level* of backtracking. *SR* can be decomposed on SR_1, … SR_n, that is, we define *n* levels of inference. The user defines *level = i* if the concerned ambiguity is in the rules SR_i; after the user selects a Level, our inference engine restores *SF* by removing all facts added after the Level point. We mean by Level point, the ambiguity that Thinker will represent to the user for reviewing his choice.

If the required level is different from the actual one, Thinker will reload all the rules of the *SR* corresponding to level. Moreover, in the process of backtracking, Thinker

analyzes the rules which were carried out and which have a link with the conclusions of the last rule executed before the request of backtracking. Thus, the rule re-presented by Thinker to the user, depends directly or indirectly of the last facts added. Thinker is a program which terminates since the number of SR is bounded and the set of rules in each *SR* is also bounded.

4 Case Study

We present in Figure 1 the B specification of a system to control the access of persons to buildings.

```
MACHINE Batiment
SETS
    BAT; PERS
CONSTANTS
    aut, com
VARIABLES
    sit
INVARIANT
    aut ∈ PERS ↔ BAT ∧ com ∈ BAT ↔ BAT ∧
    sit ∈ PERS → BAT ∧ sit ⊆ aut ∧ com ∩ id(BAT) = {}
OPERATIONS
    pass ≙ ANY    p, b
           WHERE (p, b) ∈ aut ∧ (sit(p), b) ∈ com
           THEN   sit(p) := b
           END
END
```

Fig. 1. The running example

```
Machine(BUILDING)
Abstract_set(PERS;a;b)
Abstract_set(BAT;a;b)
Relation(aut;PERS;BAT)
Relation(com;BAT;BAT)
Function(sit ;PERS ;BAT)
Inclusion(sit ;aut)
Constant(aut ;BUILDING)
Con-
stant(com ;BUILDING)
Variable(sit ;BUILDING)
Type(aut;belong)
Type(com;belong)
Type(sit;belong)
Type(sit;included)
```

Fig. 2. Initial SF of the running example

```
R1   abstract_set(!A;!n;!m) > set(!A)
R2   relation(!x;!A;!B), set(!A) > class(!A)
R3   function(!x;!A;!B), set(!A) > class(!A)
R4   relation(!x;!A;!B), class(!A), class(!B) > association(!x;!A;!B)
R5   function(!x;!A;!B), class(!A), class(!B) > association(!x;!A;!B)
R6   constant(!x), association(!x;!A;!B) > is_frozen(!x)
R7   machine(!m), variable(!x;!m), type(!x; belong) > class(!m),
     attribute_1_value(!x;!m)
R8   machine(!m), variable(!x;!m), type(!x; included) > class(!m),
     attribute_*_value(!x;!m)
R9   machine(!m), constant(!x;!m), type(!x; belong) > class(!m),
     attribute_1_value(!x;!m)
R10  variable(!A;!m), inclusion(!A;!B), association(!B;!C;!D) >
     association(!A;!C;!D)
R11  variable(!A;!m), inclusion(!A;!B), association(!B;!C;!D) >
     attribute(!A;!C), type_attribut(!A;!D)
R12  variable(!A;!m), inclusion(!A;!B), association(!B;!C;!D) >
     class_association(!B;!C;!D), attribute(!A;!B), type_attribut(!A;
     booleen)
R13  variable(!A;!m), inclusion(!A;!B), relation(!B;!C;!D)
     >class_asociation(!C_!D;!C;!D), attribut(!A;!C_!D)
R14  machine(!m), class(!m), set(!A), class(!A) > composi-
     tion_relation(!A;!m)
```

Fig. 3. The initial set of rules

Persons and buildings are represented respectively by abstract sets PERS and BAT, the authorization of persons to access to buildings and the communication between buildings are represented respectively by two constant relations *aut* and *com*. For safety, the situation of a person can be only in an authorized building; also a person can pass from a building to another only if these two buildings communicate and if the second building is authorized for this person. The situation of a person is represented by a total function *sit*.

The first step of the transformation of the B specification consists of using the THINKER parser. This parser will provide all the initial facts of the SF illustrated by Figure 2. Then, begin the inference process. Our inference engine analyzes the *SR* rules (Figure 3), one by one, according rules order in *SR*.

The first facts added to *SF* are *set(PERS)* and *set(BAT)* as shown in Figure 4. These are obtained by rule R1. Afterwards, THINKER will apply the rules R2 and R3 which transform *PERS* and *BAT* into classes. R4, R5 and R6 transform *aut* into a frozen association between *PERS* and *BAT*. After the application of the rule R9, THINKER will meet the first ambiguity, between R10, R11 and R12.

There are three possible representations for the variable *sit*.

All these possible representations offered by THINKER are illustrated by Figure 4. We suppose that the user choose the rule R12, i.e., the variable *sit* is transformed into a Boolean variable in the class-association *aut*.

After choosing the rule which will be applied, our inference engine offers, before analyzing next rules, the opportunity to the user to review his choice. If the user does not want to backtrack, THINKER continues its exploration. The next rules applied are R13 and R14 that add the facts: *class-association(PERS_BAT;PERS;BAT)* and *attribut(sit;PERS_BAT)* to the *SF*.

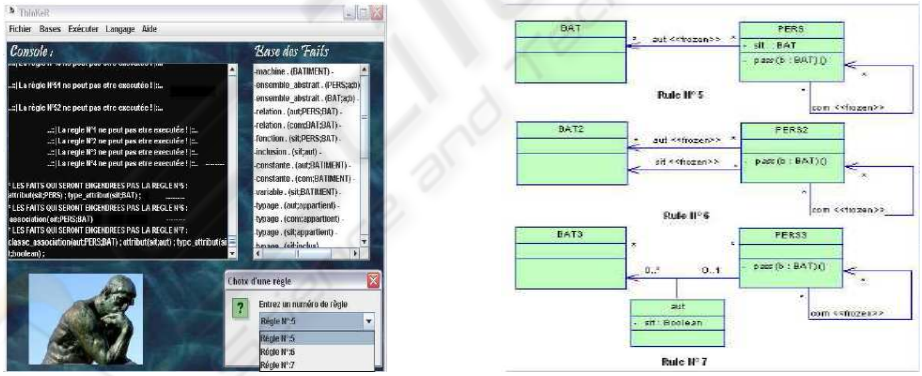

Fig. 4. Different representations of the variable *Sit*

When the user loads a new *SR*, the level is incremented. The ambiguity is recorded in a sort of history allowing the user to backtrack at each moment of the execution. Every ambiguity found by the program is set up like a break point. Moreover, THINKER records this ambiguity according the last fact found before the backtracking is requested and restore the state of the *SF* just before ambiguity (L).

The different representations of the variable sit are presented again to the user. We suppose that the user will choose the rule R11 that transforms sit into an attribute of the class PERS. The inference process continues as far as there is a rule that can be applied.

5 Conclusions

In this paper we have presented an interactive inference engine THINKER. It is a generic tool which can be used to support inference requiring user intervention to select one rule from several rules having the same premises. THINKER uses back-tracking to allow the modification of previous selected rules if there an ambiguous choice. We have illustrated this tool over a translation process of B models into UML diagrams. This case requires interactivity because we do not constrain the developer to a specific style of translation. The translation is guided by heuristics which require user validation.

References

1. Moore, R., Lopes, J., 1999. Paper templates. In *TEMPLATE'06, 1st International Conference on Template Production*. INSTICC Press.
2. Abrial, J.-R., 1996. Extending B without changing it (for developing distributed ystems). *In 1st Conference on the B method, Putting into Practice Methods and Tools for Information System Design*. pp. 169-190, Nantes.
3. Fekih, H., Jemni Ben Ayed, L., Merz, S., 2006. Transformation of B Specifications into UML Class Diagrams and State Machines. *In: 21st Annual ACM Symposium on Applied Computing*, pp. 1840-1844, Dijon.
4. Gawanmeh, A., Tahar, S., Jemni Ben Ayed, L., 2008. Event-B based Invariant Checking of Secrecy in Group Key Protocols, *The 33rd **IEEE** Conference on Local Computer Networks (LCN), Workshop on Network Security (WNS) (LCN 2008)*, Montreal
5. Idani, A., Ledru, Y., 2006. Dynamic graphical UML views from formal B specifications. *In Information and Software Technology*, Vol 48, n°3. pp. 154-169.
6. Johnson, L.B., 1994 Third conference on Clips proceedings.
7. Jovanovic, J., Gasevic, D., Devedzic, V., 2004. *A GUI for Jess. Expert Systems with Applications*. Elsevier. pp. 625-637.
8. Object Management Group. *Unified Modeling Language Specification*, Version 2.0. Specification, OMG (2003). http://www.uml.org/.
9. Schnupp, P.H., 1989. Building Expert Systems in Prolog. Munich.
10. Tatibouet, B., Hammad, H, Voisinet, H.C., 2002. From an abstract B specification to UML class diagrams. *In 2nd IEEE Intl. Symp. Signal Processing and Information Technology (ISSPIT'2002), Marrakech, Morocco.*