Validation of UML Conceptual Schemas with Operations*

Anna Queralt and Ernest Teniente

Universitat Politècnica de Catalunya {aqueralt, teniente}@lsi.upc.edu

Abstract. The purpose of validating a conceptual schema is to check whether it specifies what the designer intended. Our approach to validation consists in translating the schema into logic in such a way that any reasoning method can be used to perform the validation tests defined by the designer. An important contribution of this work is that it takes into account the operations defined in the schema.

1. Introduction

In software quality assurance, the purpose of the validation process is to answer to the question *Am I building the right system*?. In the context of conceptual modeling, validation can be used to assure the quality of a conceptual schema instead of a piece of code. To this end, it is desirable to provide the designer with some assistance, so that he can check whether the conceptual schema properly specifies what he intended.

A *conceptual schema* consists of a *structural schema*, which defines the relevant static aspects of the domain, and a *behavioral schema*, which defines the only changes that can be performed on the information. It includes a set of *system operations*, which view the system as a black box and are not assigned to classes [4].

Fig. 1 shows the structural schema of an on-line auction site that we will use as an example. The system stores information about users, and each user is the owner of a set of products. Users bid for products by specifying the amount they offer. Additionally, this structural schema includes some textual integrity constraints that must be satisfied.

A test that the designer can perform to validate the schema is to check whether it accepts at least one instance satisfying all the constraints. For example, the following instantiation of the schema: "Mick is a user who owns a book, and bids 200\$ for a bicycle, owned by Angie, who had set a starting price of 180\$" satisfies all the graphical and textual constraints. However, the fact that the structural part of a schema is satisfiable does not necessarily imply that the whole conceptual schema also is. That is, when we take into account that the only changes admitted are those specified by the operations, it may happen that the properties fulfilled by the structural schema alone are no longer satisfied. For instance, if the schema does not contain any operation that successfully populates the class User, it will not be possible to populate any other class (instances of product will neither exist, since each Product needs an owner and, in turn, bids need products and users).

^{*} This work has been partly supported by the Ministerio de Ciencia y Tecnología under projects TIN2005-06053 and TIN2005-05406

102 Proceedings of CAiSE'08 Forum





This means that our conceptual schema must include, for example, an operation *registerUser* that a designer could define by means of the following operation contract:

Op:	registerUser(id:String, e-mail:String)
Pre:	
Post:	<pre>User.allInstances()->exists(u u.oclIsNew() and u.id=id and u.e-mail=e-mail)</pre>

We assume a strict interpretation of the contracts [7], which prevents the application of an operation if a constraint is violated by the state satisfying the postcondition.

In this work we propose an approach to validate a UML conceptual schema, with its constraints and operations specified in OCL¹. To do this, we translate the schema into a set of logic formulas. The result of this translation ensures that the only changes allowed are those specified in the behavioral schema, and can be validated using any reasoning method or tool that is capable of dealing with negation of derived predicates.

2. Translation of a Conceptual Schema into Logic

When considering the behavioral schema in the validation, it must be taken into account that the population of classes and associations at a certain time t is just the result of all the operations that have been executed before t. For instance, *Angie* may only be an instance of *User* at a time t if the operation *registerUser* has created it at some time before t and no other operation has removed it between its creation and t.

For this reason, it must be guaranteed that the population of classes and associations at a certain time depends on the operations executed up to that moment. To do this, we propose that operations are the basic predicates of our logic formalization. Classes and associations will be represented by means of derived predicates, and their derivation rules will ensure that their instances are precisely given by the operations executed.

Then, an instance of a predicate p representing a class or association exists at time t if it has been added by an operation at some time t2 before t, and has not been deleted by any operation between t2 and t. Formally, the general derivation rule is:

 $p([P,],P_1,...,P_n,T) \leftarrow addP([P,]P_1,...,P_n,T2) \land \neg deletedP(P_i,...,P_j,T2,T) \land T2 \leq T$ $deletedP(P_i,...,P_i,T1,T2) \leftarrow delP(P_i,...,P_i,T) \land T>T1 \land T\leq T2$

where *P* is the OID (object identifier), which is included if *p* is a class. $P_{i},...,P_{j}$ are the terms of *p* that suffice to identify an instance of *p*. In particular, if *p* is a class or association class, $P=P_i=P_j$. Predicates *addP* and *delP* are also derived predicates that hold if some operation has created or deleted an instance of *p* at time *T*, respectively.

¹ The subset of OCL considered consists of all the OCL operations that result in a boolean value, including *select* and *size*, which can also be handled by our method.

Let op-add P_i be an operation, with parameters $Par_1,...,Par_n$ and precondition pre_i such that its postcondition specifies the creation of an instance of a derived predicate p. For each such operation we define the following rule:

 $addP([P,]Par_i,...,Par_kT) \leftarrow op-addP_i([P,]Par_1,...,Par_m,T) \land pre_i(T_{pre}) \land T_{pre}=T-1$ where $Par_i,...,Par_k$ are those parameters of the operation that indicate the information required by the predicate p, and T is the time in which the operation occurs. The literal $pre_i(T_{pre})$ is the translation of the precondition of the operation [6].

Similarly, for each operation $op-delP_i(Par_1,...,Par_m,T)$ with precondition pre_i that deletes an instance of p we define the derivation rule:

 $delP(Par_i,...,Par_j,T) \leftarrow op-delP_i(Par_1,...,Par_m,T) \land pre_i(T_{pre}) \land T_{pre}=T-1$ where $Par_i,...,Par_i$ are those parameters that identify the instance to be deleted.

For instance, the class *User* of our example will be represented by:

 $user(U, Id, Email, T) \leftarrow addUser(U, Id, Email, T2)$

 $addUser(U,Id,Email,T) \leftarrow registerUser(U,Id,Email,T)$

where U corresponds to the unique OID. In turn, *addUser* is a derived predicate whose definition depends on the operations of the behavioral schema that create instances of *User*. In particular, it will hold if the operation *registerUser* has been executed.

Since our schema does not include any operation to remove users, the derived predicate *deletedUser* must not be defined in this case.

Additionally, a set of constraints must be added to the translation to ensure the correct occurrence of the operations. In particular, since two operations cannot occur at the same time, for each operation O with parameters $p_1,...,p_n$ we define the following constraint for each parameter $p_i: \leftarrow o(P_1 1,...,P_n 1,T) \land o(P_1 2,...,P_n 2,T) \land P_i 1 <> P_i 2.$

And for each pair *O*, *O*2 of operations: $\leftarrow o(P_1,...,P_n,T) \land o2(Q_1,...,Q_m,T)$.

Moreover, all constraints of the UML structural schema are also translated into formulas in denial form according to [6], but now they are defined in terms of derived predicates instead of basic ones.

3. Our Approach to Validation

Our approach to validation is aimed at providing the designer with the ability to define his own tests to see how the schema behaves in a particular situation, and then compare the results obtained with the ones expected according to the requirements. This will be done taking into account both the structural and the behavioral parts of the schema.

Our method consists in reducing the problem to checking the satisfiability of a derived predicate. In this way, a derived predicate that formalizes the desired test is defined. With this input, together with the translated schema itself, any satisfiability checking method that is able to deal with derived predicates can be used to validate the schema. For instance, an interesting question could be "Can all the classes of the schema be populated?". The following derived predicate formalizes this test:

 $populated \leftarrow user(U,Uid,Em,T) \land product(P,Pid,Price,Own,T) \land bid(B,Pr,Us,Amt,T)$

It can easily be seen that the schema of our example does not satisfy this property, since its behavioral schema does not allow creating an instance of *User* owning at least one *Product*, as required by the cardinality constraint in *Offered by*. This means that the conceptual schema is not correct and the designer must solve this situation either by

104 Proceedings of CAiSE'08 Forum

making the operation *registerUser* responsible of creating instances of the association *Offered by*, or by changing the cardinality constraint from 1..* to *.

By studying the results of the tests, and with his knowledge about the requirements, the designer will be able to decide if the schema is correct, and modify it if necessary.

4. Related Work

We briefly summarize the work related to the validation of conceptual schemas with a behavioral part. One of the first methods to do this belongs to the area of deductive databases [2], and proposes a framework to validate a schema using planning methods.

In the context of UML, there is an approach that combines two methods: UML-B [8] to translate a UML schema into B, and ProB [5], to validate it. However, UML-B only accepts a subset of the UML, and does not admit OCL. Moreover, ProB requires that the possible values of types are enumerated, which does not guarantee completeness.

The rest of existing UML/OCL approaches that somehow consider the behavioral part may report as valid a state satisfying all the constraints but that is impossible to construct using the operations defined in the schema [1, 3].

5. Conclusions

We have proposed a new approach to validate a complete UML conceptual schema, with its textual constraints and operations expressed in OCL. Our approach helps the designer to check that the schema defined correctly specifies the requirements.

This is achieved by translating the conceptual schema, including its behavioral part, into a logic representation such that any satisfiability checking method able to deal with derived predicates can be used to validate the schema.

References

- Brucker, A. D., Wolff, B.: The HOL-OCL Book. Swiss Federal Institute of Technology (ETH), 525 (2006)
- Costal, D., Teniente, E., Urpí, T., Farré, C.: Handling Conceptual Model Validation by Planning. CAiSE'96 LNCS 1080 (1996) 255-271
- 3. Gogolla, M., Bohling, J., Richters, M.: Validating UML and OCL Models in USE by Automatic Snapshot Generation. Software and System Modeling 4(4) (2005) 386-398
- 4. Larman, C.: Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development. 3rd edn. Prentice Hall PTR (2004)
- Leuschel, M., Butler, M.: ProB: An Automated Analysis Toolset for the B Method. Software Tools for Technology Transfer DOI: s10009-007-0063-9 (2008)
- Queralt, A., Teniente, E.: Reasoning on UML Class Diagrams with OCL Constraints. In: Conceptual Modeling - ER 2006. LNCS 4215 (2006) 497-512
- Queralt, A., Teniente, E.: Specifying the Semantics of Operation Contracts in Conceptual Modeling. Journal on Data Semantics JoDS VII (2006) 33-56
- Snook, C., Butler, M.: UML-B: Formal Modeling and Design Aided by UML ACM Trans. on Soft. Engineering and Methodology 15(1) (2006) 92-122