Translating Fusion/UML to Object-Z

Margot Bittner, Florian Kammüller
Technische Universität Berlin
Institut für Softwaretechnik und Theoretische Informatik
{margot,flokam}@cs.tu-berlin.de

Abstract

We present an extension of the development method Fusion/UML that translates the results of analysis and design into the formal specification language Object-Z. The extended process establishes a consistency relationship between analysis and design. Furthermore, a formal specification for the implementation is produced.

1. Introduction

Fusion [C+94] is a method for the analysis and the design of object-oriented software systems. It has contributed to the development of the UML. However, as we find, the current Unified Process, that is, the recommended method for the engineering of systems in the language UML, has not preserved much of the initial appeal of Fusion. Hence, we use a combination of the UML language with the Fusion process in teaching and projects. This method is now extended by an additional translation step into the object-oriented specification language Object-Z [S00] establishing a connection of engineering methods and formal specification which in turn renders possible formal refinement proofs and mechanical verification, e.g. [SKS02].

In co-design it seems crucial to establish a formal specification of the design of an embedded system. Although the Fusion/UML method is primarily aimed at the development of software designs, it establishes a clear picture of the system borderline and the interfaces to other systems. Therefore, we believe that the current extension by a translation step into a formal specification language contributes to the issues of co-design.

2. Fusion/UML

The method Fusion/UML is separated in two phases: analysis and design. Each phase guides the construction of models expressed as UML diagrams. There are six models for the analysis: (system) class model, use case model, time-line model, life cycle model, and operation model. The design models are four: object interaction model, reference model, class interface model, and inheritance model. The Fusion method gives a firm guideline in which order to develop the models and what information to carry on from step to step supporting the check of consistencies between the various views. The main models are briefly introduced in the following by means of an example of a very simple saving account.

For the *analysis*, the information gathered from the use cases, time-lines, and life-cycle models eventually enables to develop the initial class model further into the *system class model* determining not only the static structure but also the borderline between the system and the actors. As an example, consider the diagram depicted in Figure 1. The

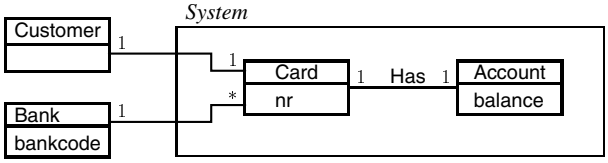

Figure 1. system class model for account

final result for the dynamic part of the analysis is the *operation model*. For example, the operation model for the bank system contains a system operation that enables a customer to pay into his account.

Operation	=	deposit
Description	=	A customer pays an amount into his account.
Input	=	$amnt : \mathbb{N}, acc : \mathbb{N}$
Reads	=	$\underline{c : Card}$ **with** $c.nr = acc, Has$
Changes	=	$\underline{a : Account}$ **with** $(c, a) \in Has$
Sends	=	$\underline{: Customer} : \{trans\_ok\}$
Pre	=	**true**
Post	=	$a.balance' = a.balance + amnt \wedge$ **is_sent** $\{trans\_ok\}$

The *design* has to be consistent with the analysis, in particular, for all system operations defined during the analysis, the *object interaction model* has to define *object interaction graphs* describing the execution of the system operations on the objects in the system. The object interaction graph of the operation deposit is shown in Figure 2.

Proceedings of the First ACM and IEEE International Conference on Formal Methods and Models for Co-Design (MEMOCODE'03) ISBN 0-7695-1923-7/03 $17.00 © 2003 IEEE

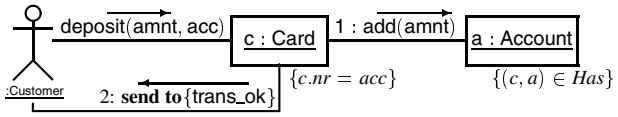

Figure 2. object interaction for deposit

3. Translation into Object-Z

The schematic process of translation into Object-Z is determined by a set of general translation rules. Starting from the system class model, these rules mainly use the operation model and the object interaction model as input to produce sets of classes in Object-Z. The schematic translation is here merely illustrated by showing its result when applied to the running example. For a full account of the set of rules and the general translation scheme see [BKa03]. Firstly, we show the translated system operation deposit being part of the one class $System_{OP}$ representing the system as it is viewed after the analysis phase. $System_{OP}$ comprises sets of system classes, their associations and operations. Actor specifications are separate classes but adhere to the interfaces defined by the operations.

$System_{OP}$

$$cards : \mathbb{P}\, Card$$
$$accounts : \mathbb{P}\, Account$$
$$Has : Card \leftrightarrow Account$$

deposit

$$\Delta(accounts)$$
$$amnt?, acc? : \mathbb{N}$$
$$m! : Report$$

$$\exists\, c : cards, a : accounts \bullet$$
$$c.nr = acc? \land$$
$$(c, a) \in Has \land$$
$$a.balance' = a.balance + amnt? \land$$
$$m! = trans\_ok$$

For the design, the object interactions are translated into the controller *Card* and the collaborator *Account*.

Card

$$a : Account$$
$$nr : \mathbb{N}$$

$$deposit \mathrel{\widehat=} [\Delta(a)\ amnt?, acc? : \mathbb{N} \mid self.nr = acc?] \bullet a.add$$

Account

$$balance : \mathbb{N}$$

add

$$\Delta(balance)$$
$$amnt? : \mathbb{N}$$

$$balance' = balance + amnt?$$

3.1. Refinement

The class $System_{OP}$ of the analysis is refined by several controller classes of the design – in our example *Card* and *Account*. To summarize those in one structure, the representation of the system in Object-Z contains a class $System_{OI}$ that entails sets of references to the controller classes and the associations. The refinement condition

$$System_{OP} \sqsubseteq System_{OI}$$

enables to verify that the axiomatic descriptions in the operation model conform to their representations in the object interaction model. Hence, Object-Z refinement ensures consistency between analysis and design in addition to the classical verification of system implementations.

4. Conclusions

Fusion/UML is a mature method to guide the process of designing systems. It supports the checking of consistencies inside and between the involved models. The presented translation into Object-Z, however, enables the schematic derivation of a formal specification. The formal specification reflects the results of the design. It enables the consistency check between analysis and design.

References

[BKa03] M. Bittner and F. Kammüller. *Controlling Consistency in UML with Fusion and Object-Z.* Technical Report 2003-6, TU-Berlin, 2003.

[C+94] D. Coleman *et al. Object-Oriented Development – the Fusion Method.* Prentice-Hall, 1994.

[S00] Graeme Smith. *The Object-Z Specification Language.* Kluwer Academic Publishers, 2000.

[SKS02] G. Smith, F. Kammüller, T. Santen. *Encoding Object-Z in Isabelle/HOL.* The Z and B User's Conference, Volume 2272 of LNCS, 2002.