SelfSync: A Dynamic Round-Trip Engineering Environment

Ellen Van Paesschen[1]
Advisor: Maja D'Hondt[2]

[1] Programming Technology Laboratory
[2] Software and System Engineering Laboratory
Vrije Universiteit Brussel
Pleinlaan 2, 1050 Brussel, Belgium
{evpaessc, mjdhondt}@vub.ac.be

Abstract. Model-Driven Engineering (MDE) advocates the generation of software applications from models, which are views on certain aspects of the software. In order to minimize the delta between these views we propose a highly dynamic Round-Trip Engineering (RTE) technique where the elements of the various views are one and the same. We combine Extended Entity-Relationship diagrams with an object-oriented prototype-based language in a two-phased technique that allows for the rapid prototyping of the modeled applications. Run-time objects are included in the RTE process resulting in advanced constraint enforcement. Moreover support is provided for straightforward role modeling.

1 Introduction

In *Model-Driven Engineering* (MDE) software applications are generated from models, which are views on certain aspects of the software. The goal of this research is to minimize the "distance" (the delta) between different views. We consider the following three views:

- A domain analysis view represented by a data modeling diagram
- Implementation objects, related to object-oriented programs at *code-time*
- Population objects, derived from implementation objects, containing actual data for running the application

During *Round-Trip Engineering* (RTE) these views need to be synchronized continuously [1],[5],[10]. We want to provide a highly dynamic approach to RTE, where the elements of the data modeling view and the corresponding implementation objects are one and the same. This contrasts with other approaches, which usually employ a synchronization strategy based on transformation [10],[24] (see Section 4). Moreover we want to include population objects in the RTE process.

An interesting academic case study in this context is *role modeling* [19]. In this case the distance between the data modeling view and a corresponding implementation is significant: from a modeling perspective roles are subtypes of the persons performing them but in the code a person object performing a role is more specialized than the role object itself [15], [9].

2 Our Approach

We propose a two-phased approach that continuously synchronizes between a data modeling view and a view on an object-oriented implementation [14], [15], [16]. For the data modeling view we selected the *Extended Entity-Relationship* diagramming technique [3] while the object-oriented implementation is developed in the prototype-based language Self [20].

EER Modeling EER diagrams consist of the typical data modeling elements, similar to *Class Diagrams* in the *Unified Modeling Language* (UML): entities (classes in the UML), attributes and operations[1] in entities, and association and inheritance relations between entities. The associations can be 1-to-1, 1-to-many, and many-to-many. The EER notation we use combines existing approaches [3], [6] but is merely a consequence of our choice of development platform.

There is an almost religious discussion between the (E)ER and the UML communities as to which approach is better. Typical claims are that (E)ER modeling is more formally funded but that the UML is more open [17]. In our work, however, the use of EER does not exclude the transfer of our conceptual results to an UML-based context. We describe Round-Trip Engineering on the data modeling level in terms of entities, attributes and operations, and association and inheritance relations. These EER modeling elements have equivalent modeling elements in Class Diagrams of the UML.

Self The object-oriented implementation language we employ is the prototype-based language Self. Prototype-based languages can be considered object-oriented languages without classes. As such, a *prototype* is used for sharing state between objects and new objects can be created by cloning a prototype. Self introduces another concept, *traits*, which share behavior among objects and let objects inherit from them, similar to classes. For more details on the language we refer to [23].

Part of the motivation to select Self is based on its reflective and dynamic character that is crucial in Round-Trip Engineering. Class-based languages such as Java and C++ that apply static typing and have few or no reflection facilities do not allow for synchronization at the level of the run-time population objects. Moreover they do not support role modeling in a straightforward manner [9]. The fact that in Self parents are shared and can be modified dynamically caused us to prefer Self to a dynamically-typed class-based language such as Smalltalk. Role modeling can be simulated in Smalltalk but not as natural as in Self [15].

2.1 A Two-Phased Approach

Our approach constitutes two typically but not necessarily subsequently executed phases, which we present in detail in [14]. In the first *active modeling*

---

[1] We extended the standard EER diagram with operation slots in addition to attribute slots.

phase a user draws an EER diagram while corresponding Self implementation objects — prototypes and traits — are automatically created. In reality, the Self objects *are* the modeled entities: drawing a new EER entity automatically results in a graphical EER *entity view* being created on a new Self object. Hence, we support incremental and continuous synchronization *per entity* and *per object*: changes to an EER entity are in fact changes to a view on one object and thus automatically propagated to the object via Self's reflection mechanism. Similarly, changes to an object are automatically propagated to the corresponding EER entity.

The second phase of our approach is an *interactive prototyping* process[2]. This phase allows a user to interactively create and initialize ready-to-use population objects from each implementation object created in the previous phase.

3 Tool Support and Validation

SelfSync is a tool that implements the two-phased approach described in Section 2.1. First, we extended Self with a drawing editor for EER diagrams. Next, we added a new EER "view" on Self objects with the help of the Morphic framework and realized a bidirectional active link between EER views and implementation objects. As explained in Section 3.1, SelfSync supports (1) enforcing constraints on population objects steered from the data model, (2) advanced method synchronization between data model and implementation, (3) changing populations of objects steered from the data model and (4) a natural synchronization between modeling and implementing during role modeling.

3.1 Validation

Constraint Enforcement After the interactive prototyping phase we ensure that the multiplicity constraints imposed by a one-to-one or one-to-many relationship between two entities are satisfied at all times. When two entities are in a relationship in which the first one has a single reference (one or zero) to the second one, the uniqueness of this reference is enforced in the population objects in two ways. We first ensure that all population objects that have been derived from the first entity refer to at most one population object that has been derived from the second entity. Secondly, we also ensure that only one population object derived from the second entity refers to population objects derived from the first entity. If two entities are in a one-to-one relationship, this is enforced in the two directions.

Dependencies between entities and weak entities in an EER diagram result in another kind of enforcement of population objects derived from these entities. In this case we ensure that when a population object is deleted, all population objects that refer to it and have been derived from a weak entity that depends

---

[2] Note that a *prototype* is a special object in prototype-based languages for supporting data sharing of several objects whereas *prototyping* is the activity of instantiating and initializing a program into a ready-to-use, running system.

on the entity from which the deleted population object is derived, are deleted also. Note that for the enforcement to be actually performed, the population objects that are candidates for deletion are not allowed to be referenced by any other population object.

Method Synchronization The EER diagram used in SelfSync is extended with operations. These operations are linked to the method bodies of the corresponding methods in the implementation objects. First, the method bodies can be edited in the EER diagram, which is automatically synchronized with the actual method bodies in the implementation objects, and vice versa. Second, we also support the possibility to "inject" behavior before or after one or more selected operations in the EER diagram. This is a simple *visual* version of Aspect-Oriented Programming [13], where the join points are selected in a diagram instead of described by a pointcut. This new piece of code is again automatically added at the beginning or end of the method bodies of all selected operations in the EER diagram. These *code injections* maintain their identity: at any point in time the layers of different code injections of an operation can be consulted. Each of these injections can be removed locally or in all operations where this specific injection was added.

Object Generations Changing a method in an implementation object, more specifically in the traits has repercussions on all population objects that have been derived from it. Since we allow changing method bodies by manipulating the corresponding operations in the EER diagram, SelfSync supports behavioral evolution of entire existing generations of population objects, steered from the EER diagram.

The Role Modeling Concept SelfSync was successfully extended with a role modeling concept to represent the fact that one entity can dynamically adapt the behavior of another entity. To tackle the paradox of roles being both sub- and supertypes [19] we introduced the concept of *warped hierarchies* [15]. Modeling roles by means of our extension to the EER diagram results in corresponding implementation objects being automatically created with the structure of warped hierarchies. In the corresponding population objects, an arbitrary number of roles can be added or removed dynamically thanks to multiple inheritance and dynamic parent modification. The technique is based on meta-programming and Self's state inheritance mechanism called *copy-down* [23].

4   Existing Approaches

Round-Trip Engineering The state-of-the-art in RTE includes application such as Rational XDE [22], Borland Together [24], and FUJABA [21]. One of the leaders in this domain is Borland's Together. In this commercial tool set the synchronization mechanism between UML Class Diagrams and implementation is realized by the *LiveSource* technology. More specifically, the implementation

model (i.e. the source code) is parsed and rendered as two views: a UML Class Diagram and in a formatted textual form. LiveSource is in fact a code parsing engine. The user can manipulate either view and even the implementation model. However, all user actions are translated directly to the implementation model and then translated back to both views. Population objects are not included in the RTE process. There is no real support for constraint enforcement or for manipulating operations in the Class Diagram. Role modeling as described above is rather hard in a Class Diagram [9], [19]. Other related work in RTE, is mostly concerned with characterizing RTE rather than providing concrete tool support [1].

Role Modeling We summarize the four approaches that are most relevant to our approach described in [15]. For more approaches we refer to [19]. The category concept [7] is defined as the subset of the union of a number of roles (types). As in our approach the Entity-Relationship diagram was extended: relationships are not defined on entity types, but on categories. In [2] roles are considered temporal specializations: statically, a manager is a specialization of a person. However, when a particular person object becomes a manager, its type is changed from person to the subtype employee thereby inheriting all aspects of its new role. In this way reversed specializations, similar to warped hierarchies, are realized temporarily. [18] also separate between static and dynamic type hierarchies: state sharing, behaviour sharing, as in Self, and subset hierarchies are combined into a new specialization modeling concept. In [12] delegation is used to implement dynamic roles that "import" state and behavior from their parent objects.

The role modeling concepts in the approaches described above provide suitable alternatives but were – to the best of our knowledge – never integrated in an object-oriented RTE modeling environment that supports automatic synchronization between the modeled roles and a corresponding implementation.

EER and Object-Orientation Since the late eighties, it has been encouraged to combine (E)ER models and object-orientation (OO) [4]. Various approaches and techniques exist for translating EER into object-orientation [8],[11]. Such mappings can be used in the domain of object-relational (O/R) mappers [25]. These tools generate an object implementation from a data model such as (E)ER, and possibly support synchronization of both models. Some of them generate code to enforce constraints on relationships and dependencies between implementation objects, based on the data model. However, these applications do not consider behavior (operations) at the level of the data model.

References

1. U. Assman. Automatic roundtrip engineering. *Electronic Notes in Theoretical Computer Science*, 82.
2. C. Bock and J. Odell. A more complete model of relations and their implementation: Roles. *JOOP*, 11(2):51–54, 1998.

3. P. P. Chen. The entity-relationship model - toward a unified view of data. *ACM Trans. Database Syst.*, 1(1):9–36, 1976.

4. P. P. Chen, B. Thalheim, and L. Y. Wong. Future directions of conceptual modeling. In *Conceptual Modeling*, pages 287–301, 1997.

5. S. Demeyer, S. Ducasse, and S. Tichelaar. Why unified is not universal? In *UML'99, Fort Collins, CO, USA, October 28-30. 1999, Proceedings*, volume 1723 of *LNCS*, pages 630–644. Springer, 1999.

6. R. Elmasri and S. B. Navathe. *Fundamentals of Database Systems*. Addison-Wesley World Student Series, 3 edition, 1994.

7. R. Elmasri, J. Weeldreyer, and A. Hevner. The category concept: an extension to the entity-relationship model. *Data Knowl. Eng.*, 1(1):75–116, 1985.

8. J. Fong. Mapping extended entity relationship model to object modeling technique. *SIGMOD Record*, 24(3):18–22, 1995.

9. M. Fowler. Dealing with roles. Technical report, Department of Computer Science, Washington University, 1997.

10. A. Henriksson and H. Larsson. A definition of round-trip engineering. Technical report, Linkopings Universitet, Sweden, 2003.

11. R. Herzig and M. Gogolla. Transforming conceptual data models into an object model. In *ER'92, Karlsruhe, Germany, October 1992, Proceedings*, volume 645 of *Lecture Notes in Computer Science*, pages 280–298. Springer, 1992.

12. A. Jodlowski, P. Habela, J. Plodzien, and K. Subieta. Dynamic object roles – adjusting the notion for flexible modeling. In *IDEAS*, pages 449–456, 2004.

13. G. Kiczales, J. Lamping, A. Menhdhekar, C. Maeda, C. Lopes, J.-M. Loingtier, and J. Irwin. Aspect-oriented programming. In *Proceedings of the 15th European Conference on Object-Oriented Programming (ECOOP'01)*, pages 220–242. Springer-Verlag, 1997.

14. E. V. Paesschen, M. D'Hondt, and W. D. Meuter. Rapid prototyping of extended entity relationship models. In *ISIM 2005, Hradec Nad Moravici, Czech Republic, April 2005, Proceedings*, pages 194–209. MARQ, 2005.

15. E. V. Paesschen, W. D. Meuter, and M. D'Hondt. Role modeling in selfsync with warped hierarchies. In *Proceedings of the AAAI Fall Symposium on Roles, November 3 - 6, Arlington, Virginia, USA*, 2005 (to appear).

16. E. V. Paesschen, W. D. Meuter, and M. D'Hondt. Selfsync: a dynamic round-trip engineering environment. In *Proceedings of the ACM/IEEE 8th International Conference on Model-Driven Engineering Languages and Systems (MoDELS'05), October 2-7, Montego Bay, Jamaica*, 2005 (to appear).

17. K.-D. Schewe. UML: A modern dinosaur? In *Proc. 10th European-Japanese Conference on Information Modelling and Knowledge Bases, Saariselkä (Finland), 2000*. IOS Press, Amsterdam, 2000.

18. M. Snoeck and G. Dedene. Generalization/specialization and role in object oriented conceptual modeling. *Data Knowl. Eng.*, 19(2):171–195, 1996.

19. F. Steimann. A radical revision of UML's role concept. In A. Evans, S. Kent, and B. Selic, editors, *UML 2000, York, UK, October 2000, Proceedings*, volume 1939 of *LNCS*, pages 194–209. Springer, 2000.

20. D. Ungar and R. B. Smith. Self: The power of simplicity. In *OOPSLA '87, Orlando, Florida, USA*, pages 227–242, New York, NY, USA, 1987. ACM Press.

21. Fujaba: http://wwwcs.uni-paderborn.de/cs/fujaba/.

22. Rational: http://www-306.ibm.com/software/awdtools/developer/rosexde/.

23. Self: http://research.sun.com/self/.

24. Together: http://www.borland.com/together/.

25. Toplink: http://www.oracle.com/technology/products/ias/toplink/index.html.