Run-Time Round-Trip Engineering with Self

Ellen Van Paesschen[1] - Maja D'Hondt[2]

[1] Programming Technology Laboratory
Vrije Universiteit Brussel
Pleinlaan 2, 1050 Brussel, Belgium
evpaessc@vub.ac.be
[2] Laboratoire d'Informatique Fondamentale de Lille
Université des Sciences et Technologies de Lille
59655 Villeneuve d'Ascq Cédex, Lille, France
maja.d-hondt@lifl.fr

Summary

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]. 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 [9]. Moreover we want to include population objects in the RTE process.

An interesting academic case study in this context is *role modeling* [7]. 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 [6], [3].

We propose a two-phased approach that continuously synchronizes between a data modeling view and a view on an object-oriented implementation [4], [6], [5]. For the data modeling view we selected the *Extended Entity-Relationship* diagramming technique [2] while the object-oriented implementation is developed in the prototype-based language Self [8].

Our approach constitutes two typically but not necessarily subsequently executed phases, which we present in detail in [4]. In the first *active modeling* 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[1]. This phase allows a user to interactively create and initialize ready-to-use population objects from each implementation object created in the previous phase.

SelfSync is a tool that implements the two-phased approach described above. 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. The power of SelfSync lies in four characteristics: (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.

References

1. U. Assman. Automatic roundtrip engineering. *Electronic Notes in Theoretical Computer Science*, 82.
2. P. P. Chen. The entity-relationship model - toward a unified view of data. *ACM Trans. Database Syst.*, 1(1):9–36, 1976.
3. M. Fowler. Dealing with roles. Technical report, Department of Computer Science, Washington University, 1997.
4. E. V. Paesschen, M. D'Hondt, and W. D. Meuter. Rapid prototyping of eer models. In *ISIM 2005, Hradec Nad Moravici, Czech Republic, April 2005, Proceedings*, pages 194–209. MARQ, 2005.
5. 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.
6. 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).
7. 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.
8. 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.
9. Together: http://www.borland.com/together/.

[1] 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.