Enhancing the UML with shadows for Agile development

Marc Conrad, Marianne Huchard

Enhancing the UML with Shadows for Agile Development

Marc Conrad   Marianne Huchard

University of Bedfordshire   *Université Montpellier II*
United Kingdom   *France*
marc.conrad@beds.ac.uk   *huchard@lirmm.fr*
http://perisic.com/marc   *http://www.lirmm.fr/∼huchard*

Abstract: Agile methodologies can be enhanced by the use of shadows as this feature because of its inherent ability to dynamically change the behavior of classes and objects, provides mechanisms to ease common tasks such as prototyping, deprecating, dynamic classification and interclassing at run-time. We feel however that shadows should be considered a notion beyond any specific programming languages, so that they can as well be integrated in model-driven software engineering. Therefore we introduce Shadows-UML, an UML extension, that would help to push forward the convergence between model-driven and agile methodologies.

1 The Shadow Concept

Keeping in mind the current trends in software engineering there are indeed pragmatic reasons to expect the UML to play a significant role in the agile software community [TG02, Rum06]. We show how the concept of *shadows*, as introduced in LPC [P+98], is an excellent means to support Agile Software Development strategies and go on to present the integration of shadows into the UML.

The core concept behind the shadow functionality is to mask one or more methods in a target object (the "shadowed" object). Every invocation of a shadowed method is first received by the shadow. The shadow can then forward this call to the shadowed object or do something else.

Obviously such a concept needs clarification in a number of issues as for instance shadowing of attributes, or about which object is allowed to add shadows to other objects etc. On the programming language level these issues have been discussed in [CFH+06].

In Software Engineering context shadows are useful in the following situations:

Deprecation and Prototyping. Experience shows that software libraries are under constant evolution. Shadows are a means to add additional (temporary) functionalities to an existing library. This is helpful when deprecated methods are to be "phased out" of a software package or when methods are added during prototyping.

Realization of non-standard inheritance relationships. Features like reclassification [DDDG01] and interclassing [CL02] that aim on a more dynamic use and extension of inheritance relationships are well known in the research community but indeed are virtually

never used in "real world" software development projects. Shadows provide a unified way to make these features available. For details we refer the reader to [CFH$^+$06].

2 Shadows and the UML

The examples in the previous section show that Shadows are an excellent means to enhance the unanticipated evolution of written programs. In the context of model-driven development however they are as well valuable for solving design-to-code problems: Shadows help to translate almost unconstrained models that may include multiple inheritance, multi-instantiation or dynamic classification into main-stream programming languages that are extended with the single new feature of shadows (as shown in [CFH$^+$06] for Java).

However with the increasing tendency to develop software driven by modelling in agile contexts it is clear that shadows must be somehow integrated into the UML itself. An UML-based notation for shadows (that we call Shadows-UML) is useful both for unanticipated evolution and design-to-code translation. For the latter these Shadows-UML diagrams will specify the use of shadows in translating an unconstrained UML class diagram (independently of a specific programming language). Only after this, code will be written. In this context patterns of use of shadows can be defined for easing translation.

Following the UML 2.0 Superstructure Specification [Obj04] we extended the UML in the standard way by providing a profile, composed of stereotypes, tagged values and constraints that specify the new semantics. This is indeed necessary as for instance to clarify the behavior of objects that are equipped with multiple shadows and relevant priority rules.

References

[CFH$^+$06] Marc Conrad, Tim French, Marianne Huchard, Carsten Maple, and Sandra Pott. Enriching the Object-Oriented Paradigm via Shadows in the Context of Mathematics. *Journal of Object Technology*, 5(6):107–126, July-August 2006.

[CL02] P. Crescenzo and P. Lahire. Using both Specialisation and Generalisation in a Programming Language: Why an How, 2002.

[DDDG01] S. Drossopoulou, F. Damiani, M. Dezani-Ciancaglini, and P. Giannini. Fickle: Dynamic object re-classification. In *ECOOP'01*, LNCS 2072, pages 130–149. Springer, 2001.

[Obj04] Object Management Group. *UML 2.0 Superstructure Specification. ptc/04-10-02*, October 2004.

[P$^+$98] Lars Pensjö et al. LPC, 1998. http://www.lysator.liu.se/mud/lpc.html.

[Rum06] Bernhard Rumpe. Agile Test-based Modeling. In *Proceedings of the 2006 International Conference on Software Engineering Research & Practice. SERP'2006.s*, USA, 2006. CSREA Press.

[TG02] Ivan Tyrsted and Peter Gerken. IDEOGRAMIC, 2002. http://www.ideogramic.com.