Available online at www.sciencedirect.com
SCIENCE DIRECT®
Electronic Notes in Theoretical Computer Science
ELSEVIER Electronic Notes in Theoretical Computer Science 127 (2005) 55–58
www.elsevier.com/locate/entcs

Reflective Designs — An Overview

Robert Hirschfeld[1]

DoCoMo Communications Laboratories Europe
Munich, Germany

Ralf Lämmel[2]

Vrije Universiteit & Centrum voor Wiskunde en Informatica
Amsterdam, The Netherlands

Abstract

We render runtime system adaptations by design-level concepts such that running systems can be adapted and examined at a higher level of abstraction. The overall idea is to express design decisions as applications of design operators to be carried out at runtime. Design operators can implement design patterns for use at runtime. Applications of design operators are made explicit as design elements in the running system such that they can be traced, reconfigured, and made undone.
Our approach enables Reflective Designs: on one side, design operators employ reflection to perform runtime adaptations; on the other side, design elements provide an additional reflection protocol to examine and configure performed adaptations. Our approach helps understanding the development and the maintenance of the class of software systems that cannot tolerate downtime or frequent shutdown-revise-startup cycles.
We have accumulated a class library for programming with Reflective Designs in Squeak/Smalltalk. This library employs reflection and dynamic aspect-oriented programming. We have also implemented tool support for navigating in a system that is adapted continuously at runtime.

Note: This extended abstract summarises our full paper [7].

Keywords: Reflective Designs, Runtime Adaptation, Design Elements, Design Operators, Design Patterns, Reflection, Method-Call Interception, Meta-Programming, Aspect-Oriented Programming, Dynamic Weaving, Dynamic Composition, AspectS, Squeak, Smalltalk, Metaobject Protocol

[1] Email: hirschfeld@docomolab-euro.com
[2] Email: ralf.laemmel@cwi.nl

doi:10.1016/j.entcs.2004.08.033

Runtime system adaptation

Our work on Reflective Designs is concerned with adaptation of software systems at runtime, as needed for dynamic component coordination [11], runtime system configuration [1], dynamic service adaptation [5,6], and rapid prototyping without shutdown-revise-startup cycles [12]. Runtime adaptability is crucial for systems with strong availability demands, such as in telecommunications. Downtime of such systems can barely be tolerated. Software maintenance and evolution has to be carried out in the running system.

Reflective Designs enhance object-oriented design and programming by techniques for runtime system adaptation. There are two key notions: *design elements* and *design operators*, which we will explain in turn.

Design elements

We contend that a program is structured according to design decisions. We require that design decisions are represented explicitly in the program. Thereby, software design will be traceable in the program. We even require that design decisions are to be represented explicitly in the running system. We use the term design element to denote representations of design decisions in programs. In fact, we require that design elements are amenable to reflection such that design decisions can be observed and modified at runtime. With that, the notion of runtime system adaptations boils down to explicit construction, modification, and retirement of design elements.

Design elements can be examined and (re-) configured. Here, examination and (re-) configuration are used in the sense of introspection and intercession. Normal object-oriented introspection and intercession concerns the fields and methods of objects. Design-level introspection and intercession concerns design-level concepts such as the *participants* for a given design element. The examination of participants exemplifies design-level introspection. The configuration of participants and their roles exemplifies design-level intercession. Furthermore, for each object in the running system, we can introspect effective adaptations, i.e., the list of design elements that affect the object at hand.

Design operators

When compared to basic techniques such as the use of a metaobject protocol [8], the use of design elements makes runtime system adaptations more disciplined and more manageable. To this end, we provide abstractions that capture common design elements in a reusable manner. Applications of such

abstractions perform system adaptations at a design level; hence, we call them design operators. Our work, so far, has concentrated on operators that model the realisation of common design patterns. The view 'design patterns as operators' also occurs in previous work [15,2,9,10,13,14]. The novelty of our work is that our operators serve for *runtime* system adaptation, and *runtime* reflection on designs.

We can distinguish at least three kinds of operators. Additive operators superimpose additional structure or behaviour onto the running software system. Subtractive operators define and remove slices of behaviour or structure in the running software system. Refactoring operators revise the running system in a semantics-preserving manner.

It is clear that design operators can only be provided in the context of a sufficiently reflective programming system. Actual applications of design operators result in two effects. Firstly, the corresponding design elements are constructed. Secondly, the system's actual structure and behaviour is adapted as intended by the underlying design decision. Applications of design operators can be made undone by deactivating the corresponding design element. In case an inactive element is never ever needed again, we can let the element retire.

Implementation in Squeak/Smalltalk

We have developed the Reflective Designs framework as a class library for Squeak/Smalltalk. The implementation makes original use of infrastructure for reflection, method wrappers [3], and dynamic aspect-oriented programming with AspectS [4]. The Reflective Designs framework involves several layers of abstraction, while these layers are presented as APIs to the programmer. The idea is that layers at a higher level of abstraction perform less lower level reflection. Using the Reflective Designs framework, we have exercised some scenarios of runtime system adaptations.

We have also provided interactive tool support for reflective designs. Accordingly, we have extended some existing tools, such as the normal system browser, and we have provided new tools such as a dedicated 'reflective designs center'. The tool extensions are particularly interesting in so far that we have implemented them as self-applications of the Reflective Designs framework, e.g., the system browser is adapted by appropriate design elements.

Concluding remarks

We believe that Reflective Designs and our prototypical implementation of this approach provide useful input for further research on runtime system

adaptation. Major directions for future work are the following. Firstly, the fusion of Reflective Designs and refactoring transformations should be completed. We note that we have focused on additive and subtractive adaptations in our work so far. Secondly, the robustness of Reflective Designs should be improved by dedicated system analyses and rollback mechanisms. Thirdly, our practical approach to reflective designs needs to be complemented by formal support. The ultimate goal is an approach where runtime system adaptations are as powerful and robust as static meta-programs today.

References

[1] Akkawi, F., A. Bader and T. Elrad, *Dynamic Weaving for Building Reconfigurable Software Systems* (2001), in Online Proc. of OOPSLA 2001 Workshop on Advanced Separation of Concerns in Object-Oriented Systems.

[2] Aßmann, U., *AOP with design patterns as meta-programming operators*, Technical Report 28, Universität Karlsruhe (1997).

[3] Brant, J., B. Foote, R. Johnson and D. Roberts, *Wrappers to the Rescue*, in: E. Jul, editor, *ECOOP '98*, LNCS **1445** (1998), pp. 396–417.

[4] Hirschfeld, R., *AspectS – Aspect-Oriented Programming with Squeak*, in: M. Aksit, M. Mezini and R. Unland, editors, *Objects, Components, Architectures, Services, and Applications for a Networked World*, LNCS **2591** (2003), pp. 216–232.

[5] Hirschfeld, R. and K. Kawamur, *Dynamic Service Adaptation*, in: *Proc. DARES (ICDCSW'04)*, Hachioji, Tokyo, Japan, 2004, pp. 290–297.

[6] Hirschfeld, R., K. Kawamura and H. Berndt, *Dynamic Service Adaptation for Runtime System Extensions*, in: R. Battiti, M. Conti and R. Lo Cigno, editors, *Wireless On-Demand Network Systems*, LNCS **2928** (2004), pp. 227–240.

[7] Hirschfeld, R. and R. Lämmel, *Reflective Designs*, IEE Proceedings Software (2004), Special Issue on Reusable Software Libraries. To appear. Available at http://homepages.cwi.nl/~ralf/rd/.

[8] Kiczales, G., J. des Rivieres and D. Bobrow, "The Art of the Metaobject Protocol," MIT Press, Cambridge, MA, USA, 1991, viii + 335 pp.

[9] Krishnamurthi, S., Y.-D. Erlich and M. Felleisen, *Expressing Structural Properties as Language Constructs*, in: S. Swierstra, editor, *Proc. ESOP'99*, number 1576 in LNCS, 1999, pp. 258–272.

[10] Ludwig, A., "Automatische Transformation großer Softwaresysteme," Ph.D. thesis, Universität Karlsruhe (2002).

[11] Pinto, M., L. Fuentes, M. Fayad and J. Troya, *Separation of Coordination in a Dynamic Aspect Oriented Framework*, in: *Proc. AOSD'02* (2002), pp. 134–140.

[12] Popovici, A., T. Gross and G. Alonso, *Dynamic Weaving for Aspect Oriented Programming*, in: *Proc. AOSD'02* (2002), pp. 141–147.

[13] Sullivan, G., *Advanced Programming Language Features for Executable Design Patterns*, Technical Report AIM-2002-005, MIT AI Laboratory (2002).

[14] von Dincklage, D., *Making Patterns Explicit With Metaprogramming*, in: *Proc. GPCE'03*, LNCS (2003), pp. 287–306.

[15] Zimmer, W., "Frameworks und Entwurfsmuster," Ph.D. thesis, Universität Karlsruhe (1997).