RAMSES: a Reflective Middleware for Software Evolution

Walter Cazzola[1], Ahmed Ghoneim[2], and Gunter Saake[2]

[1] Department of Informatics and Communication,
Università degli Studi di Milano, Italy
cazzola@dico.unimi.it
[2] Institute für Technische und Betriebliche Informationssysteme,
Otto-von-Guericke-Universität Magdeburg, Germany
{ghoneim|saake}@iti.cs.uni-magdeburg.de

Abstract. Software systems today need to dynamically self-adapt against dynamic requirement changes. In this paper we describe RAMSES a reflective middleware whose aim consists of consistently evolving software systems against runtime changes. This middleware provides the ability to change both structure and behavior for the base-level system at run-time by using its design information. The meta-level is composed of cooperating objects, and has been specified by using a design pattern language. The base objects are controlled by meta-objects that drive their evolution. The essence of RAMSES is the ability of extracting the design data from the base application, and of constraining the dynamic evolution to stable and consistent systems.

Keywords: Software Evolution, XMI, UML, Reflection, Meta-Objects.

1 Introduction

Many object-oriented information systems today need to dynamically adapt themselves against runtime changes. Some of the changes such as modify its structure and behavior may cause the base-systems to behave in an unexpected way. Therefore, software systems need to be capable of dynamically adapting their structure and behavior at run-time and of checking their consistency to face sudden changes. Software development asked for the way to modify the base objects at runtime without going to rebuild the application again. It requires a new approach, which adapts the base application as well as on advances in software technology. This new perspective reifies the design data of the base application and by modifying such reification it adapts the base application against runtime changes.

A topical issue in the software engineering research area consists of producing software systems able to adapt themselves to environment changes by adding new and/or modifying existing functionalities. There are a number of mechanisms for obtaining adaptability. One of these mechanisms is *reflection* [2, 6]. A non-stoppable software systems provide an excellent way to dynamically adapt itself against runtime changes at its environment. A non-stoppable systems are characterized by long life cycle. Usually, these systems are deployed to be continuously online for several years. During this lifetime we want to be able to dynamically maintain and adapt these systems against runtime events without bring the whole system to a halt.

We propose an infrastructure to dynamic adapt software systems. In our approach,we adopt a reflective system [4, 5], that allows to render self-adaptable at runtime the base-level system. The meta-level systems is composed of an interpreter engine for managing the evolution and validating consistency processes for runtime changes.

The meta-level behavior is described by a family of patterns [3], the meta-level manages both the evolution and consistency of the base-level system. The cooperative meta-objects at the meta-level consult the engines (see figure 1), and adapting the reified objects for dynamic behavior. Changes to the reified system can be made at runtime and are immediately reflected to its base-components. The evolution and

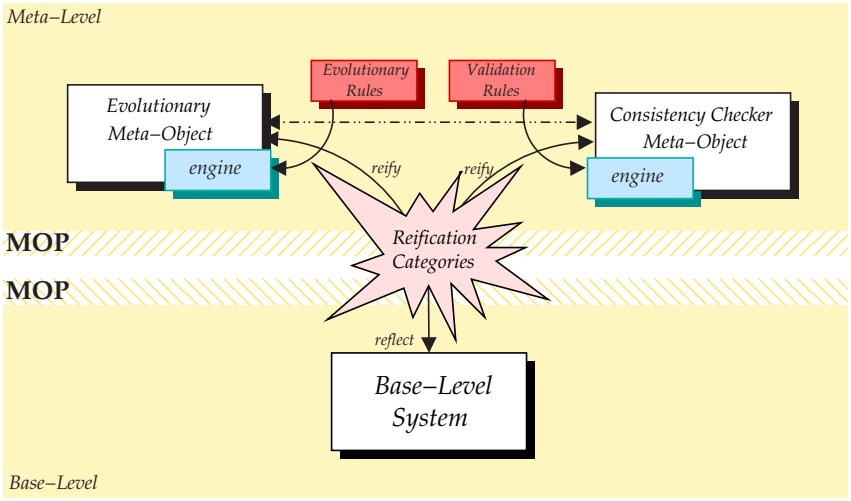

Fig. 1. RAMSES architecture.

consistency are not hard-coded, neither are they generated. Instead, we build a reflective framework of the base-systems that can be automatically self-adapted for any changes to be active long-life span. Our reflective architecture define two cooperative meta-objects (evolutionary and consistency) both of them refer to the engine to evolve the system and validate the consistency of its semantics at runtime.

2 RAMSES Overview

RAMSES (Reflective and Adaptive Middleware for Software Evolution of Systems) performs two phases to carry our self-adaption. In the first phase, the RAMSES's meta-level extracts the design information as XMI schemas from the base application and it reifies them in the meta-level to constitute the meta-data. Whereas, in the second phase, RAMSES's meta-level plans the dynamic adaptation of the base-level system, gets the runtime events, evolves the meta-data against the detected event, checks the consistency, and finally reflects the modified data to the base-level. This infrastructure is considered to be dynamically adaptive because changes in the execution environment cause objects, attributes and collaborations to be created and modified at runtime to achieve new behaviors not previously foreseen by the original application. This goal is achieved by:

- adopting a reflective architecture which reifies system design information and reflects back the changes on the system design;
- manipulating the design information and checking the system consistency against evolution in the meta-level;
- using configurable rules to govern the system evolution through its design information.

Adaptation and validation are respectively driven by a set of rules which define how to adapt the system according to the detected event and the meaning of system consistency.

2.1 The Reflective Architecture

To render a system self-adapting[3], we encapsulate it in a two-layers reflective architecture as shown in Fig. 1. The base-level is the system that we want to render self-adapting whereas the meta-level is a second software system which reifies the base-level design information and plans its evolution when particular events occur. By using a reflective architecture, thanks to the transparency and separation of concerns properties of reflection, we can render self-adapting every software system without changing its code.

At the moment, this approach allows two kinds of dynamic evolution: *structural* and *behavioral* evolution. This limitation is due to the fact that we just consider the following design information related to the base-level system:

- *object model*, which describes objects and their relationships; this model represents the structural part of the system;
- *sequence diagrams*, which trace system operations between objects (inter-object connection); and
- *statecharts*, which represent the evolution of the state of each object (intra-object connection) in the system.

The meta-level is responsible of dynamically adapting the base-level and it is composed of some special meta-objects, called *evolutionary meta-objects*. There are two types of evolutionary meta-objects: the *evolutionary* and the *consistency checker* meta-objects (see Fig. 1). Their goal consists of consistently evolving the base-level system. The former is directly responsible for planning the evolution of the base-level through adding, changing or removing objects, methods, and relations. The latter is directly responsible for checking the consistency of the planned evolution and of really carrying out the evolution through the causal connection typical of each reflective system.

2.2 Design Information as Meta-Data

Through the causal connection, the base-level system and its design information are reified into *reification categories* in the meta-level. Classic reflection takes care of reifying the state and every other dynamic aspect of the base-level system, whereas the design information provides a reification of each design aspect of the base-level system such as the collaborations among its components. The reification categories content is the main difference of RAMSES with respect to standard reflective architectures. Usually, reifications represent the base-level system behavior and structure not its design information. Reification categories are meta-data that represent the base-level system design information in the meta-level. Both evolutionary and consistency checker meta-objects directly work on such representatives and not on the real system, this allows a safe approach to evolution postponing every change after validation checks. As described in [3] when an external events occur as a reaction, the evolutionary meta-object proposes an evolution to the consistency checker meta-object which validates the proposal and schedules the adaptation of the base-level system if the proposal is accepted.

2.3 Evolution Planning and Validation

Adaptation and validation are respectively driven by a set of rules which define how to adapt the system in accordance with the detected event and the meaning of system consistency.

[3] By the sentence *to render a system self-adapting* we mean that such a system is able to change its behavior and structure in according with external events by itself.

To give more flexibility to the approach, these rules are not hardwired in the corresponding meta-object rather they are passed to a sub-component of the meta-objects themselves, respectively called *evolutionary* and *validation engines*, which interpret them. Therefore, each meta-object has two main components: (i) the core which interacts with the rest of the system (e.g., detecting external events/adaptation proposals, or manipulating the reification categories/applying the adaptation on the base-level system) and implementing the meta-object's basic behavior, and (ii) the engine which interprets the rules driving the meta-object's decisions.

The evolutionary meta-object plans the evolution of the base-level system when an event that requires its adaptation occurs. The evolutionary meta-object passes to its engine all the data about the occurred event and the entities that could be involved by the evolution. On this basis, the engine chooses and applies a group of evolutionary rules that serve to build the plan for evolving the base-level exploiting the reified meta-data. The evolutionary meta-object proposes the planned evolution to the consistency checker meta-object which validates its soundness. Similarly to the planning phase, the consistency checker meta-object demands the validation to its engine that exploits the validation rules and the base-level's meta-data. The plan for the evolution is concretized on the base-level if and only if the consistency checker considers its application sound otherwise a new plan has to be designed.

3 Benefits and Drawbacks

Our approach to software evolution has the following benefits:

- evolution is not tailored on a specific software system but depends on its design information;
- evolution is managed as a nonfunctional features, therefore, can be added to every kind of software system without modifying it;
- evolution strategy is not hardcoded in the system but it can dynamically change by substituting the evolutionary and validation rules; and
- RAMSES decreases the complexity of evolution and validation by defining only one meta-level. That represent the meta-processes of maintaining and evolving the meta-data.

Unfortunately there are also some drawbacks: i) we need a mechanism for converting UML diagrams in the corresponding XMI schemas (problem partially overcome by using Poseidon for UML [1]); ii) decomposing the evolution process in evolution and consistency validation could be inadequate for evolving systems with tight time constraints.

4 Conclusion

We have presented the RAMSES (Reflective and Adaptive Middleware for Software Evolution of Systems) middleware whose aim consists of self-adapting object-oriented systems against environmental changes. In this paper we have given an overview of the whole reflective architecture for dynamically evolving and validating consistency of a software system. The main features of our infrastructure can be highlighted as follows: 1) it allows to extract the system design information as XMI schemas from base objects; 2) by using MOP capability the XMI schemas will be reified to constitute the meta-data used in the meta-level; 3) both evolution and consistency are managed by the collaborations between meta-objects. Finally, 4) by using reflection we reflect the modified design information to the base-level. We are currently working on implementing a prototype of RAMSES.

References

1. Marko Boger, Thorsten Sturm, and Erich Schildhauer. *Poseidon for UML Users Guide*. Gentleware AG, Vogt-Kölln-Str. 30, D-22527 Hamburg, Germany, 2000. http://www.gentleware.com/products/documentation/PoseidonUsersGuide.pdf.
2. Walter Cazzola. Evaluation of Object-Oriented Reflective Models. In *Proceedings of ECOOP Workshop on Reflective Object-Oriented Programming and Systems (EWROOPS'98)*, in 12th European Conference on Object-Oriented Programming (ECOOP'98), Brussels, Belgium, on 20th-24th July 1998. Extended Abstract also published on ECOOP'98 Workshop Readers, S. Demeyer and J. Bosch editors, LNCS 1543, ISBN 3-540-65460-7 pages 386-387.
3. Walter Cazzola, James O. Coplien, Ahmed Ghoneim, and Gunter Saake. Framework Patterns for the Evolution of Non-stoppable Software Systems. In Pavel Hruby and Kristian Elof Søresen, editors, *Proceedings of the 1st Nordic Conference on Pattern Languages of Programs (VikingPLoP'02)*, pages 35–54, Højstrupgård, Helsingør, Denmark, on 20th-22nd of September 2002. Microsoft Business Solutions.
4. Walter Cazzola, Ahmed Ghoneim, and Gunter Saake. Reflective Analysis and Design for Adapting Object Run-time Behavior. In Zohra Bellahsène, Dilip Patel, and Colette Rolland, editors, *Proceedings of the 8th International Conference on Object-Oriented Information Systems (OOIS'02)*, Lecture Notes in Computer Science 2425, pages 242–254, Montpellier, France, on 2nd-5th of September 2002. Springer-Verlag. ISBN: 3-540-44087-9.
5. Walter Cazzola, Ahmed Ghoneim, and Gunter Saake. Software Evolution through Dynamic Adaptation of Its OO Design. In Hans-Dieter Ehrich, John-Jules Meyer, and Mark D. Ryan, editors, *Objects, Agents and Features: Structuring Mechanisms for Contemporary Software*, Lecture Notes in Computer Science, Heidelberg, Germany, February 2004. Springer-Verlag.
6. Pattie Maes. Concepts and Experiments in Computational Reflection. In Norman K. Meyrowitz, editor, *Proceedings of the 2nd Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA'87)*, volume 22 of *Sigplan Notices*, pages 147–156, Orlando, Florida, USA, October 1987. ACM.