Declarative Metaprogramming to Support Reuse in Vertical Markets

Ellen Van Paesschen *
Ellen.Van.Paesschen@vub.ac.be
Programming Technology Lab
Vrije Universiteit Brussel, Pleinlaan 2, 1050 Brussel, Belgium

23rd May 2002

Abstract

To eliminate the difficulties caused by the implicit nature of the natural relationship between domain models (and the corresponding delta-analyses) and framework instances, we aim to provide a mechanism for the construction of a bidirectional, concrete, active link between domain models and framework code by constructing a new instance of declarative metaprogramming. This instance supports a symbiosis between a prototype-based, framebased knowledge representation language and an object-oriented programming language. This implies that framework code in the object-oriented language co-evolves with the corresponding knowledge representation. Representing domain-dependent concepts in the same knowledge representation allows us to transform delta-analyses into framework reuse plans, and to translate changes in the framework code into domain knowledge adaptations, at a prototype-based and frame-based level.

1 Introduction

Due to the increased popularity of the object-oriented paradigm, reuse has become a technological reality. This resulted in the rise of *vertical market* companies which focus on one problem domain by reusing a specific framework [11]. For each customer, the same framework is adapted, until it meets the customer's requirements, and instantiated. In this way, companies in a vertical market solve the duality of *tailor-made* and *off-the-shelf* software [1].

The most ideal way to instantiate a framework is to reuse the common framework core - or generalities - and to fill in the variabilities depending on the customer's requirements [11].

*Author financed with a doctoral grant from the Institute for Science and Technology (IWT), Flanders

In practice, it seems often necessary to intensively change or extend the framework core, before the variabilities can be filled in and the framework can be instantiated. Therefore, the development and reuse of frameworks is a complex, iterative process.

In this context, the classical analysis phase is replaced with a *domain analysis* [8], [10], [25]. The purpose of a domain analysis is to construct a *domain model*. To define which functionalities a new customer demands from his future system, a *delta analysis* is performed on this domain model. The results of a delta analysis - the deltas - describe the differences in functionality between an instance of the current framework and the system that is desired by the customer.

Although there exists a "natural" relationship between the deltas of the delta analysis and the future framework instance, this relationship is merely implicitly present in the minds of analysts and framework engineers. The result of this implicity is that frameworks are developed and reused in a very "handcrafted" way, based on delta analyses [1]. The absence of a concrete technical link between deltas and framework implies a range of problems that are experienced during framework development and reuse.

Our position is that to make the current framework development and reuse methodologies more structured, the currently implicit relationship between deltas and framework code should be explicitated into a technological, bidirectional, concrete, active link. Our approach will apply existing techniques from the domain of Artificial Intelligence [20] and a new instance of *declarative metaprogramming* [3], [26], [6], [9], [15] to realize a coupling between the (domain) analysis level and the framework implementation level.

2 Domain knowledge

Knowledge in vertical domains In vertical domains one can differentiate five important kinds of knowledge:

- **general domain knowledge** [21], [2] is the all-containing knowledge that describes the entire domain at various levels.
- **operational knowledge** [7] is a subset of the domain knowledge that specifies the concepts and aspects (subconcepts) (contained in a system) for the domain - at the implementation level, this kind of knowledge is typically modelled by object-oriented frameworks - either explicitely in the form of classes (concepts) or implicitely by methods (aspects).
- **configuration knowledge** [7] interacts with domain knowledge by describing how the elements of the operational knowledge can be combined, and specifies the consequences of specific combinations - also at the implementation level typically modelled by object-oriented frameworks.
- **inference knowledge** [21] interacts with the domain knowledge by describing the *inference steps* using the domain knowledge, to realize a reasoning process. A *generate-*

inference that specifies the generation of a time-dependent planning is a typical example of this kind of knowledge.

- **task knowledge** [21] represents the final goal of a system inside the domain and controls the system while interacting with the inference knowledge. A *planning task* for example is a *synthesis* task that aims at generating a time-dependent action plan making use of the generate-inferences described in the previous item.

Together with (the results of) domain- and delta-analysis processes, we refer to this set of knowledge as *domain-dependent concepts*.

Examples of domain (interacting) knowledge The kinds of knowledge described above have been identified in the domain of *working hours scheduling*. In this domain we have created a framework that plans a schedule of working hours for employees in an irregular shift system. Typical instances of such a framework are planners for police people, nurses, and taxidrivers. These systems are constrained by a sector-specific legislation on the number of worked nights, free weekends, holidays, etc. The fact that "a nurse cannot work more than two subsequent weeks in night shifts" and "a police man needs a minimum of 50 percent free complete weekends on a period of two months", are examples of *general domain knowledge*. Both nurses and police men are concepts of the *operational knowledge*, while the fact that "police men have to work "on the field" in teams of two" is an example of *configuration knowledge*. Also the working hours schedule for one month, that needs to be filled in by the system, is a typical example of a concept in the operational knowledge, which has the individual day schedules as its aspects. The final goal of this system is the creation of such a working hours schedule that respects the constraints described above. Therefore, the main part of the task knowledge in this domain contains *planning tasks* (e.g. for planning the working hours of one month) that are divided into *subtasks* that plan the schedule of one day. These tasks use *generate-inferences* that generate a schedule for a whole month or for just one day.

Representing Knowledge To represent domain and related knowledge in this context, we prefer *frame-based prototype-based* knowledge representations. Frame-based knowledge representations [12] have extended the flexible, non-formal original frames [17], [23] among others with semantics and various inheritance mechanisms. These representations have proven to be a suitable representation medium - as opposed to the rigid logic representations - when not all the knowledge can initially be identified, which is often the case when modelling systems in vertical domains. Compared to the class-based paradigm, the prototype-based way of thinking [18], [19] increases these advantages since, in this specific context of knowledge representation, it is often easier to think of a concrete example rather than to abstract some class. Moreover, the not so rare representation of a few exceptions on a base class often demands the creation of a totally new class that will be instantiated only once. Prototype-based knowledge representations solve this much more elegantly by *cloning* a prototype and adapting it afterwards. A last advantage of these representations

lays in the possibility to change the dynamic behaviour of an object. The behaviour is situated in the prototype which is cloned at runtime: state and behaviour can be changed afterwards.

KRS [24] is an interesting example of a frame-based prototype-based knowledge representation language. This language, implemented on top of LISP, was developed in the eighties at the Artificial Intelligence Laboratory of the Free University of Brussels. The representation primitive is a fully reflexive conceptual graph [22] which is defined and inspected by a concept language. The most important features of this concept language are lazy interpretation and a lexical scoping mechanism. KRS supports a prototype-based singular inheritance mechanism, a hierarchical controle mecahnism and a backward-chaining reasoning mechanism. Popular applications of KRS are data modelling, data-driven programming (supported by an innovative strategy to trigger daemons), representing traditional representation formalims, e.g. production systems, and implementing explicit data retrieval structures, e.g. classification trees.

We will inspire a new language (KRS') on KRS, to represent domain-dependent concepts and framework knowledge. In the related, future experiments we will focus on the following issues:

- Is the separation of the knowledge described above and its representation sufficient to describe the domain in question?
- Is it possible to describe delta's and can we extract a general proces to notate them?
- Is it necessary - at this level - to include knowledge such as "when a new element of operational knowledge occurs in more than two systems of the domain, add it as a default concept"?

3 Representing framework knowledge

Co-evolution and Declarative Meta-programming Co-evolution [3], [26], [6], [9], [15] consists of the realization of an active link between design and implementation in such a way that they can evolve together whenever changes occur. Currently, design is being considered as declarative knowledge about the implementation. This declarative knowledge is coded into logic rules, as in Prolog, and into knowledge bases. Well-known reasoning algorithms (backward and forward chainers) are being used to couple the knowledge (design) to the implementation in an active manner. Whenever changes occur in design or implementation, the necessary inference algorithms are activated. This technique is called *declarative meta-programming* [3], [26], [6], [9], [15] (DMP). Unfortunately, the current instances of DMP [26], [6], [9] are exclusively rule-based at the design level. This implies that there is no other medium than rules and facts to represent knowledge. Together with the frame-based prototype-based approach for representing domain-dependent concept, this observation leads to the development of a new instance of DMP.

A symbiosis of a knowledge representation and an OO language We propose to develop a new instance of DMP that supports a symbiosis between a frame-based prototype-based representation language KRS' (a dialect of KRS, see section 2) and the object-oriented programming language Smalltalk [13]. In this way, a framework that is implemented in Smalltalk can be described in KRS' while changes on one level will be translated and adapted into the other. From this context some interesting research topics arise:

- The symbiosis between frame-based and object-oriented
- The symbiosis between prototypes and classes
- The identification of the reusable subset of KRS (to implement KRS') and SOUL [26] (to implement the symbiosis with Smalltalk)
- A possible analogy with other languages like SOUL [26] (a symbiosis of Prolog and Smalltalk using a rule-based backward-chainer), Agora98 [4] (a prototype-based object-oriented language), and SqueakKan [9], [5] (a symbiosis of KAN and Smalltalk using a rule-based forward-chainer)

4 Adding an intelligent component to support reuse and framework changes

When the DMP instance described above is available, the domain-dependent concepts will be added at the KRS' level. This implies that these represented concepts can be linked with a framework, that is described by the new DMP instance, at the frame-based prototype-based KRS' level (cfr. figure 1). This link is realized by an intelligent component, consisting among others of a large set of meta-rules, that will be added to the new DMP instance. It is responsible for:

- The adaptation of the domain-dependent concepts at the KRS' level, based on framework changes at the implementation level
- The planning of reusing the framework at the implementation level, based on the delta's at the KRS' level

By performing a number of delta-analyses en reuse plans between the domain-dependent concepts (KRS') and the framework (new DMP instance) by hand, we will extract the requirements of the intelligent component, which will be added to the new DMP instance afterwards.

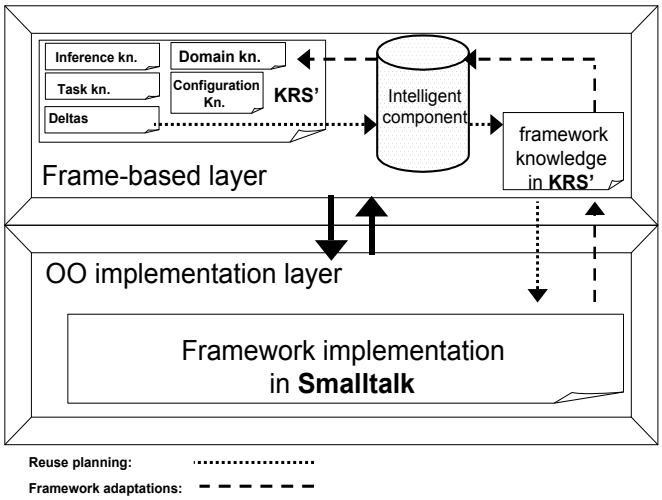

Figure 1: An intelligente DMP instance for reuse planning and framework adaptations

5 Conclusion

This paper describes the ongoing research to make the natural relation between domain-dependent concepts and framework instances explicit. At the level of the domain-dependent concepts the separation of five kinds of knowledge will be represented by KRS', a dialect of the frame-based prototype-based knowledge representation language KRS. To represent frameworks a new instance of DMP, that provides a symbiosis between KRS' and the object-oriented programming language Smalltalk, is proposed. By adding an intelligent component to this instance at the level of KRS', domain-dependent concepts and framework implementations can be coupled, making it possible to automatically translate framework adaptations to the domain-dependent level, and to automatically plan reuse of the framework based explicitely on the delta's at the KRS' level.

References

[1] Wim Codenie, Koen De Hondt, Patrick Steyaert, Arlette Vercammen, *From custom applications to domain-specific frameworks*, Communications of the ACM, 1997.

[2] Krzysztof Czarnecki, Ulrich W. Eisenecker, *Generative programming*, Addison-Wesley, 2000.

[3] Theo D'Hondt, Kris De Volder, Kim Mens, Roel Wuyts, *Co-evolution of object-oriented software design and implementation*, TACT Symposium Proceedings, Kluwer Academic Publishers, 2000.

[4] Wolfgang De Meuter, *AGORA: the Story of the Simplest MOP in the World - Or the Scheme of Object-Orientation*, Free University Brussels, Programming Technology Laboratory; 1997.

[5] Wolfgang De Meuter, Maja D'Hondt, Sofie Goderis and Theo D'Hondt. *Reasoning with Design Knowledge for Interactively Supporting Framework Reuse*. Proceedings of the Second International Workshop on Soft Computing Applied to Software Engineering (SCASE '01), 2001, p. 31-36.

[6] Kris De Volder, *Type oriented logic meta programming*, PhD thesis, Laboratorium voor programmeerkunde, Vrije Universiteit Brussel, 1998.

[7] U. Eisenecker, C. Czarnecki, P. Steyaert, *Beyond objects: generative programming*; 1997.

[8] X. Ferré, S. Vegas, *A classification of domain analysis methods*; 1997.

[9] Sofie Goderis, *A reflective forward-chained inference engine to reason about object-oriented systems*, Bachelors thesis, Laboratorium voor programmeerkunde, Vrije Universiteit Brussel, 2000.

[10] D. Holibaugh, *Joint integrated avionics working group (JIAWG) object-oriented domain analysis method (JODA)*; 1993.

[11] Ralph E. Johnson, *Components, frameworks, patterns.*

[12] Peter D. Karp, *The design space of frame knowledge representation systems*, SRI AI center, Technical report; 1993.

[13] Wilf R. Lalonde, John R. Pugh, *Inside Smalltalk, Volume II*, Prentice-Hall, 1991.

[14] H. Lieberman, *Using Prototypical Ojects to Implement Shared Behavior in Object Oriented Systems*, in Oopsla '86; 1986.

[15] Kim Mens, *Automatische architecturale conformiteitscontrole dmv logisch meta-programmerem*, PhD thesis, Laboratorium voor programmeerkunde, Vrije Universiteit Brussel, 2000.

[16] Tom Mens, *A formal foundation for object-oriented software evolution*, PhD thesis, Laboratorium voor programmeerkunde, Vrije Universiteit Brussel, 2000.

[17] Marvin Minsky, *A framework for representing knowledge*; 1975.

[18] R. Nado, R. Fikes, *Saying more with frames: slots as classes*, Computers and mathematics with applications; 1992.

[19] J. Noble, A. Taivalsaari, I. Moore, *Prototype-based programming: concepts, languages, and applications*, Springer-Verlag; 1999.

[20] Stuart Russel, Peter Norvig, *Artificial intelligence, a modern approach*, Prentice-Hall, 1995.

[21] Schreiber, Akkermans, Anjewierden, de Hoog, Shadbolt, Van de Velde, Wielinga, *Knowledge engineering and management: the commonKADS methodology*, MIT Press; 1999.

[22] John F. Sowa, *Conceptual Structures - Information Processing in Mind and Machine*, The Systems Programming Series, Addison-Wesley; 1984.

[23] John F. Sowa, *On logical, philosophical, and computational foundations of knowledge representation*, PhD thesis, STARlaboratorium, Vrije Universiteit Brussel1999.

[24] Kris Van Marcke, *The use and implementation of KRS*, PhD thesis, Artificiële intelligentie laboratorium, Vrije Universiteit Brussel, 1988.

[25] D. Weiss, C. Lai *Software Product-Line Engineering: A Family-Based Software Development Process*, Addisson-Wesley, Reading, MA; 1999.

[26] Roel Wuyts, *Synchronizing implementation and design using logic metaprogramming*, PhD thesis, Laboratorium voor programmeerkunde, Vrije Universiteit Brussel, 2000.