THE ANNALS OF "DUNĂREA DE JOS" UNIVERSITY OF GALAȚI
FASCICLE I - 2005, Economics and Applied Informatics, Year XI, ISSN 1584-0409

PATTERN MATCHING IN MODELS

Cristian GEORGESCU

"Dunărea de Jos" University of Galati
Department of Accounting and Economic Informatics
cristian.georgescu@ugal.ro

The goal of this paper is to investigate how such a pattern matching could be performed on models, including the definition of the input language as well as the elaboration of efficient matching algorithms. Design patterns can be considered reusable micro-architectures that contribute to an overall system architecture. Frameworks are also closely related to design patterns. Components offer the possibility to radically change the behaviors and services offered by an application by substitution or addition of new components, even a long time after deployment. Software testing is another aspect of reliable development. Testing activities mainly consist in ensuring that a system implementation conforms to its specifications.

Keywords: *metamodelling, pattern matching, model transformation.*

1. Introduction

As models and model transformations are relevated to first-class artifacts within the software development process, there is an increasing need to introduce more automatic ways of searching models.

One of the basic needs we have is the ability to perform efficient pattern matching at the model level. This is indeed usefull for instance for performing refactorings on models.

It is also needed when we want to do aspect weaving at model level, the pattern being in this case the aspect pointcut, and the places where it matches, the joinpoints. Patterns can be expressed as model snippets with wildcards, and really are predicates over models.

The main objective is to develop model-based methods and tools to help the software designer to obtain a certain degree of confidence in the reliability of component in particular, investigating modeling languages allowing specification of both functional and non-functional aspects and which are to be deployed on distributed systems.

The key problems are components modeling and the development of formal manipulation tools to refine the design, code generation and test activities. Components help reducing costs by allowing reuse of application frameworks and components instead of redeveloping applications. Components offer the possibility to radically change the behaviors and services offered by an application by substitution or addition of new components, even a long time after deployment. The validation techniques used are based on complex simulations of models building on the standards in the considered domain.

2. Design patterns

The research domain is the reliable and efficient design of software product lines by assembling software components described with the UML. A software system can be described like an graph with states, events and transitions. The main advantage of event structures is to represent at the same time concurrency and alternative in a partial order model and this is the reason for notations such as UML.

The object-oriented approach is now widespread for the analysis, the design, and the implementation of software systems. Rooted in the idea of modeling, object-oriented analysis, design and implementation takes into account the incremental, iterative and evolutive nature of software development.

In the object-oriented standard approach, objects are instances of classes. A class encapsulates a single abstraction in a modular way. Large software

system are seldom developed from scratch, and maintenance activities represent a large share of the overall development effort. Complexity in an object-oriented system is well known to be in the collaboration between objects, and large systems cannot be understood at the level of classes and objects. Still these complex collaborations are made of recurring patterns, called design patterns.

Design patterns play many roles in the development process. Design patterns can be considered reusable micro-architectures that contribute to an overall system architecture. Frameworks are also closely related to design patterns.

An object-oriented software framework is made up of a set of related classes which can be specialized or instantiated to implement an application. It is a reusable software architecture that provides the generic structure and behavior for a family of software applications, along with a context which specifies their collaboration and use within a given domain. The framework is in charge of managing the bulk of the application, and the application programmer just provides various bits and pieces.

3. Aspect oriented software development

The aspect oriented modeling (AOM) approach provides mechanisms for separating crosscutting functionality from core functionality in design models. Crosscutting functionality is described by aspect models and the core application functionality is described by a primary model. The integrated system view is obtained by composing the primary and aspect models.

The technique is capable of detecting some conflicts that can arise during composition. Languages for aspect-oriented programming (AOP), such as AspectJ, are now popular, and the concepts used by the AOP community such as join points, pointcuts and advice are well-known.

At the same time, in recent years, the aspect oriented software development (AOSD) approach has been developing itself beyond the programming activity. More particularly, the Early Aspects Initiative advocates the management of crosscutting properties, i.e. aspects, at the early development stages of requirements engineering and architecture design to identify the impact of aspects as soon as possible.

Some composition operators of aspects exist for these development stages, but they do not closely match standard AOP concepts. We propose an automatic way for weaving behavioural aspects given as scenarios. With these kinds of behavioural modelling languages, aspect weaving cannot always be performed at the abstract syntax level.

4. The metamodel semantics

The increasing software complexity and the reliability and reusability requirements fully justify this aproach. The main themes studied are reliable software components composition, UML-based developments validation, and test generation from UML models.

UML semantic variation points provide intentional degrees of freedom for the interpretation of the metamodel semantics. The interest of semantic variation points is that UML now becomes a family of languages sharing lot of commonalities and some variabilities that one can customize for a given application domain. This works propose to reify the various semantic variation points of UML 2.0 statecharts into models of their own to avoid hardcoding the semantic choices in the tools.

Where object-oriented languages deal with objects as described by classes, model-driven development uses models, as graphs of interconnected objects, described by metamodels. A number of new modeling languages have been and continue to be developed for this model-based paradigm, both for model transformation and for general programming using models.

Many of these use single-object approaches to typing, derived from solutions found in object-oriented systems, while others use metamodels as model types, but without a clear notion of polymorphism. Both of these approaches lead to brittle and overly restrictive reuse characteristics.

 This paper was recommended for publication by Associate Prof. Vasile MAZILESCU Ph. D

Object-oriented meta-languages such as MOF (Meta-Object Facility) are increasingly used to specify domain-specific languages in the model-driven engineering community. These meta-languages focus on structural specifications and have no built-in support for specifications of operational semantics.

This contribution presents a simple extension to object-oriented typing to better cater for a model-oriented context, including a simple strategy for typing models as a collection of interconnected objects.

Reliability is an essential requirement in a context where a huge number of softwares (and sometimes several versions of the same program) may coexist in a big telecommunication network. On one hand, software should be able to evolve very fast, as new features or services are frequently added to existing ones, but on the other hand, the occurrence of a fault in a system can be very costly, and time consuming. A lot of attention should then be paid to interoperability, i.e. the ability for software to work properly with other.

We think that formal methods may help solving this kind of problems. Note that formal methods should be more and more integrated in an approach allowing system designer to build software globally, in order to take into account constraints and objectives coming from user requirements. These methodologies are in their early years.

A good example of such techniques is the object oriented approach, which is becoming more popular in the telecommunication world.

Software testing is another aspect of reliable development. Testing activities mainly consist in ensuring that a system implementation conforms to its specifications. Whatever the efforts spent for development, this phase is of real importance to ensure that a system behaves properly in a complex environment.

5. Conclusions

Today's mainstream approaches build on the concept of component based software. The assembly of these components makes it possible to build families of products made of many common parts, while remaining opened to new evolutions. As component based systems grow more complex and mission-critical, there is an increased need to be able to represent and reason on such assemblies of components.

This is usually done by building models representing various aspects of such a product line, such as for example the functional variations, the structural aspects (object paradigm), of the dynamic aspects (languages of scenarios), without neglecting of course non-functional aspects like quality of service (performance, reliability, etc.) described in the form of contracts, or the characteristics of deployment, which become even dominating in the field of reactive systems, which are often distributed and real-time.

The key problems about components modeling and the development of formal manipulation tools is to refine the design.
All studies about new techniques for the reliable construction of software product lines, especially for distributed and reactive software will be integrated in code generation and test activities in the software developer life cicle.

A goal of this aproach is to explicitly connect research results to industrial problems through technology transfer actions. This implies, in particular, taking into account the industrial standards of the field, namely UML, Corba Component Model (CCM), Com+/.Net and Enterprise JavaBeans. It will be at the frontier of two fields of software: the field of specification and formal proof of software, and that of design which, though informal, is organized around best practices (Design Patterns or the use of off-the-shelf components).
The research activity focuses at the same time on development efficiency and reliability. The main applications of this research is clearly related to the

telecommunication domain, and mainly concern reliable construction of communication software, and object oriented systems testing.

We believe that the use of this techniques will make it possible to improve the transition between these two worlds, and will contribute to the fluidity of the processes of design, implementation and testing of software.

References

1. *J. BAYER, S. GÉRARD, O. HAUGEN, J. MANSELL, B. MOLLER-PEDERSEN, J. OLDEVIK, P. TESSIER, J.-P. THIBAULT, T. WIDEN. Families Research Book, LNCS, chap. A Unified Conceptual Model for Product Family Variability Modelling, no to be published, Springer Verlag, 2005*

2. *BENOIT BAUDRY, FRANCK FLEUREY, ROBERT FRANCE, RAGHU REDDY. Exploring the Relationshipbetween Model Composition and Model Transformation, in "Aspect Oriented Modeling (AOM) Workshop.*

3. *FRANCK CHAUVEL, JEAN-MARC JÉZÉQUEL. Code Generation from UML Models with Semantic Variation Points, in "Proceedings of MODELS/UML'2005, Montego Bay, Jamaica", S. KENT L. BRIAND (editor). ,LNCS, October 2005.*

4. *JEAN-MARC JÉZÉQUEL. Model Driven Engineering for Distributed Real Time Embedded Systems, J.-P. BABAU S. GÉRARD (editor). , chap. Real Time Components and Contracts, Hermes Science Publishing Ltd, London, 2005.*

5. *TRUNG DINH-TRONG, SUDIPTO GHOSH, ROBERT FRANCE, BENOIT BAUDRY, FRANCK FLEUREY. A Taxonomy of Faults for UML Designs, in "Model Design and Validation (MoDeVa) Workshop, Montego Bay, Jamaica", October 2005.*

6. *SÉBASTIEN GÉRARD, JEAN-MARIE FAVRE, PIERRE-ALAIN MULLER, XAVIER BLANC (editors).IDM05, Actes des 1ères Journées sur l'Ingénierie Dirigée par les Modèles, no ISBN 2-7261-1284-,http://planetmde.org/idm05/actes.pdf, Paris, June 2005.]*

 This paper was recommended for publication by Associate Prof. Vasile MAZILESCU Ph. D