enumerative model-checking of safety and liveness properties expressed in a Linear Temporal Logic (LTL). The model-checking problem being considered can be reformulated as a cycle detection problem in an oriented graph, and the basic principles behind our algorithms rely on efficient solutions to detecting cycles in a distributed environment. In particular, we employ specific structural properties of the underlying graphs (often computable in advance from the given system specification), use additional data structures to divide the problem into independent sub-problems, or translate the model-checking problem into a different problem that admits efficient parallel solution. Several of our algorithms are based on sequentially less efficient but parallelisable breadth-first exploration of the graph or on placing bounds that limit the size of the graph to be explored.

Development of a tool that supports the distributed verification of systems is one of our recent projects. The goal is to build an environment for easy implementation of parallel and distributed verification algorithms on clusters of workstations, followed by experimental evaluations and comparisons. The main characteristics are the support for the distributed generation of the state space, dynamic load balancing, distributed generation of counter-examples, fault-tolerance and re-partitioning. The distributed environment quite naturally allows for the integration and cooperation of methods and algorithms as well.

The cluster PC SGI 1200 is being used for experimental evaluation of the parallel algorithms.

Other research performed in the laboratory is focussed on the development of original methods and techniques for the automated verification of large-scale industrial critical systems, with an emphasis on the practical aspects of their application. Further, it looks at applying these and other known methods and techniques to real-life systems, optimising them to improve their efficiency, and providing software support to use them.

Link:
http://www.fi.muni.cz/paradise

Please contact:
Luboš Brim, Masaryk University Brno/
CRCIM, Czech Republic
Tel: +420 540 459 647
E-mail: brim@fi.muni.cz

Aspect-Oriented Software Evolution

by Tom Mens, Kim Mens and Tom Tourwé

All software systems are subject to evolution. This poses great challenges for software engineers. Aspect-oriented software development (AOSD), in addition to separating the different concerns during software development, can be seen as a way of overcoming many of the problems related to software evolution. At the same time, AOSD may benefit from tools and techniques that automate software evolution. ERCIM members are therefore exploring the cross-fertilisation between these two active research domains.

Real-world software needs to evolve continually in order to cope with ever-changing software requirements. Manny Lehman identified this characteristic in his so-called first law of software evolution, which addresses continuing change: a program that is used must be continually adapted else it becomes progressively less satisfactory.

This need for software to evolve continuously poses important challenges for software engineers. Advanced automated software engineering techniques and tools are needed to improve software evolution support. An ERCIM Working Group on Software Evolution is currently being formed to address this need. Two important techniques that will be investigated are software restructuring and aspect-oriented software development.

Software Restructuring

Software restructuring should be an essential activity in software engineering, according to Lehman's second law of software evolution, which addresses increasing complexity: as a program evolves its complexity increases unless work is done to maintain or reduce it.

In program transformation research, two different restructuring approaches can be distinguished. The term rephrasing is used to refer to techniques that improve the structure of the software without changing the implementation language. A typical example is software refactoring, which tries to improve the internal structure of a program without changing its external behaviour.

The term 'translation' refers to techniques that restructure the software across programming languages. A typical example is migration of legacy

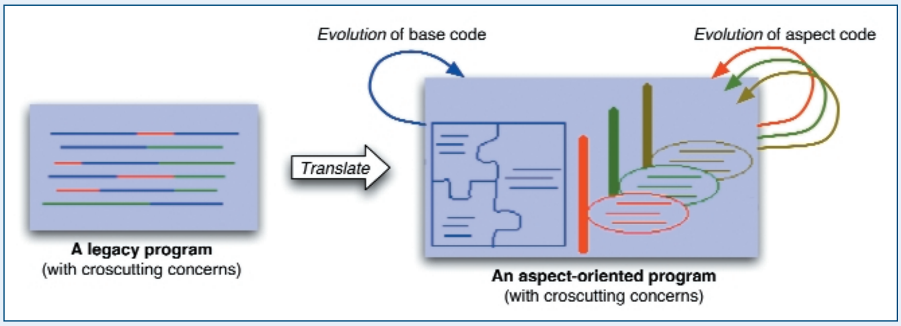

Figure 1: Cross-fertilisation of software evolution and AOSD.

code to an object-oriented equivalent (eg COBOL to Java).

Aspect-Oriented Software Development

An essential problem with software development is the tyranny of the dominant decomposition. No matter how carefully a software system is decomposed into modular units, there will always be concerns (typically non-functional ones) that cut across the chosen decomposition. The code of these crosscutting concerns will necessarily be spread over different modules, which has a negative impact on the software quality in terms of comprehensibility, adaptability and evolvability.

Aspect-oriented software development (AOSD) has been proposed as a solution to this problem. In order to capture crosscutting concerns in a localised way, a new abstraction mechanism (called an aspect) is added to existing programming languages (eg AspectJ for Java). As a result, cross-cutting concerns are no longer distributed over different modules. This means the software is easier to maintain, evolve and understand.

Cross-fertilisation

In order for AOSD to become truly successful, existing software systems need to be translated into their aspect-oriented equivalents and rephrased continuously (see Figure 1). Given the size and complexity of industrial software systems, this must be achieved with as much automated support as possible. More specifically, automated support is needed for three essential activities:

- aspect mining - techniques should be used to identify the relevant concerns in the source code
- aspect introduction - techniques are needed to define the appropriate aspects for any of the identified concerns, in order to translate the software into an aspect-oriented equivalent
- aspect evolution - techniques are required in order to evolve aspect-oriented software.

Our research investigates how formal techniques, successful in supporting traditional software evolution, can support these three new activities. For example, in a recent experiment, we used formal concept analysis to mine for cross-cutting concerns in object-oriented source code. With this approach we detected interesting features corresponding to cross-cutting functional or non-functional concerns, as well as occurrences of design patterns like the Visitor design pattern (see Figure 2).

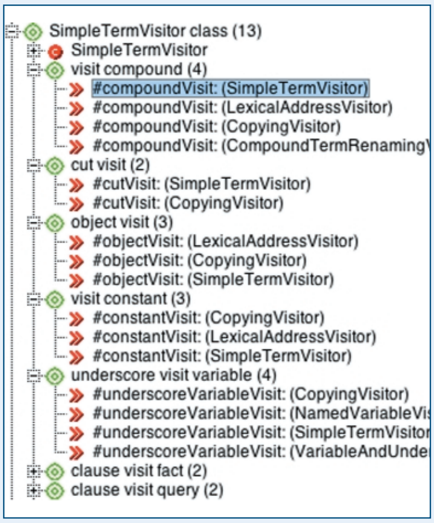

Figure 2: An occurrence of the Visitor design pattern detected by formal concept analysis. The results are classified hierarchically into groups of related software entities (in this case, classes and methods).

Typically, the implementation of such a pattern spans multiple classes and methods, and grouping these in a single hierarchical view allows the developer to understand and manipulate the pattern more easily.

In another experiment we dealt with aspect introduction and aspect evolution. More specifically, we investigated how evolution of the base code affects the definition of aspects that work on it. In order to address this issue, we proposed a more sophisticated aspect-oriented programming language, accompanied by an advanced development environment. The environment helps a developer to define aspects as logic rules, and is able to assess the impact of evolution on these aspects automatically. To this extent, the environment relies on a machine learning algorithm called inductive logic programming.

European Collaboration

The authors are involved in many successful international research activities that have been initiated in the domains discussed above:

- a Belgian FWO-funded scientific research network on 'Foundations of Software Evolution' (http://prog.vub.ac.be/FFSE/network.html)
- a Belgian IWT-funded inter-university research project on 'Architectural Resources for the Restructuring and Integration of Business Applications' (http://arriba.vub.ac.be)
- a Belgian FNRS-funded 'Research Center on Structural Software Improvement'
- a European ESF-funded scientific network on 'Research Links to Explore and Advance Software Evolution' (http://labmol.di.fc.ul.pt/projects/release/)
- a European EU-funded network of excellence on 'Aspect-Oriented Software Development'
- an ERCIM working group on 'Software Evolution'.

Please contact:
Tom Mens
Université de Mons-Hainaut, Belgium
Tel: +32 65 37 3453
E-mail: tom.mens@umh.ac.be