MoVES Newsletter

Work Package 2 - Modelling Languages & Restructuring
University in Focus: University of Antwerp

Moves Newsletter, No. 1, July 2010

Editorial

Dear Reader,

You are holding the first edition of the MoVES newsletter in your hand. MoVES is an acronym for "Modelling, Verification and Evolution of Software" and is a research project addressing fundamental issues in Software Engineering. The project is sponsored by the Belgian government (belspo, with its IAP/PAI programme) and consequently the project consortium brings together all software engineering research groups in Belgium.

The project itself is running towards its end. In the last year and a half we intend to release a newsletter every other month presenting the major results of the project. In that respect, this month's issue focuses on the work of work package 2 (Modelling Languages & Restructuring), which are explicit results of the project. However, we use the opportunity to also present related research that is conducted by one of the participating institutes, in this issue the University of Antwerp.

This newsletter contains contributions by Hubaux et al. on Textual Variability Languages, Degrandsart et al. on Model-driven Solutions for Dynamic Context Adaptation, Vanhooff et al. on Loosely Coupled Transformation Chains, and Wagelaar on Managing Platform Variability through Platform Models, as well as, contributions introducing the AnSyMo and ADReM research groups at the University of Antwerp, Schippers et al. on delMDSOC, Vangheluwe, et al. on AToM^3, and Demeyer et al. on Mining for Test Co-Evolution and Bug Tracking Databases.

But as a starter, we list all forthcoming events that we are aware of; many of them satellite events of the Automated Software Engineering – ASE 2010 conference that will be in Antwerp coming September.

Enjoy reading!

Serge Demeyer and Anne Keller

Upcoming Events & Recent Joint Publications

- *Joint Publication to Appear*: **Data-intensive System Evolution**, Anthony Cleve (INRIA), Tom Mens (UMons), Jean-Luc Hainaut (FUNDP Namur), IEEE Computer, August 2010.
- *Upcoming Conferences*: 25th IEEE/ACM International Conference on Automated Software Engineering, **ASE 2010**, 20-24 September 2010, Antwerp Belgium
  - **IWPSE-Evol 2010**: 4th International Joint ERCIM/IWPSE Symposium on Software Evolution, 20 September
  - **Moves-Verif**, 21 September
  - **WASDeTT-3**: 3rd International Workshop on Academic Software Development Tools, 20 September
- ACM/IEEE 13th International Conference on Model Driven Engineering Languages and Systems, **MODELS 2010**, 3-8 October 2010, Oslo Norway
  - **ME 2010**: International Workshop on Models and Evolution, 3 October
  - **ACES-MB 2010**: 3rd International Workshop on Model Based Architecting and Construction of Embedded Systems, 4 October
- *Submission Deadlines*: October 15th 2010, **CSMR 2011** - 15th European Conference on Software Maintenance and Reengineering
- October 31st 2010, Special Issue on **"Automated Software Evolution"** of the Elsevier Journal on Systems and Software

Arnaud Hubaux, Andreas Classen, Quentin Boucher, Patrick Heymans

Textual Variability Languages

Feature models (FMs) were introduced as part of the FODA (Feature Oriented Domain Analysis) method 20 years ago. They are a graphical notation whose purpose is to document variability, most commonly in the context of software product lines. Since their introduction, FMs have been extended and formalised in various ways. The majority of these extensions are variants of FODA's original tree-based graphical notation. But over time, textual dialects have also been proposed, arguing that it is often difficult to navigate, search and interpret large graphical FMs. The need for more expressiveness was a further motivation for textual FMs since adding constructs to a graphical language quickly starts harming its readability. Although advanced techniques have been suggested to improve the visualisation of graphical FMs, these techniques remain tightly bound to particular modelling tools and are hard to integrate in heterogeneous tool chains. Finally, our experience shows that editing functionalities offered by such tools are actually pretty limited and unhandy with large models.

Based on these observations, we proposed TVL, a textual FM dialect geared towards software architects and engineers. Its main advantages are that (1) it does not require a dedicated editor –any text editor can fit–, (2) its C-like syntax makes it both intuitive and familiar, and (3) it offers first-class support for modularity. However, TVL is meant to complement rather than replace graphical notations. It was conceived to help designers during variability modelling and does not compete, for instance, with graphical representations used during product configuration –which can actually be derived from it.

TVL comes with a set of constructs that subsumes all known textual alternatives and a complete formal semantics. The language was recently evaluated and improved in co-operation with four organisations of different sizes (from one to 28 000 employees) and domains (hardware, product software and FLOSS). Currently, tool support mainly consists of a library that checks TVL specs for well-formedness and type correctness. There are also plug-ins for text editors that provide syntax highlighting and basic text collapse/expand. The library is being extended to include translators that make TVL models amenable to advanced reasoning by third party tools such as SAT and CSP solvers.

Further Information: *http://www.info.fundp.ac.be/~acs/tvl/*
Contact: *ahu@info.fundp.ac.be, acs@info.fundp.ac.be, qbo@info.fundp.ac.be, phe@info.fundp.ac.be*

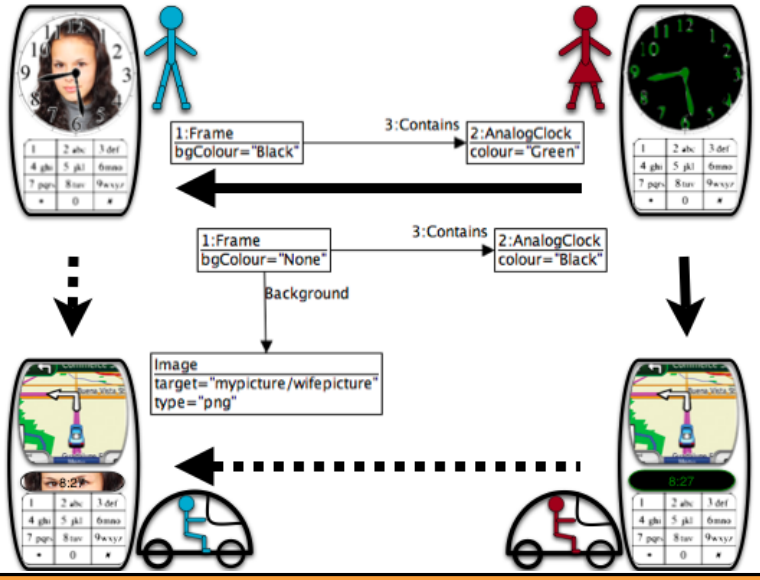
Sylvain Degrandsart, Tom Mens, Michael Hoste

Model-Driven Solutions for Dynamic Context Adaptation

Ubiquitous or *pervasive* computing, and *ambient intelligence*, form an emerging computing paradigm that is becoming more and more popular and relevant. One of its essential characteristics is the notion of *context awareness*, referring to the fact that computers (hardware) and computer programs (software) are aware of the environment (i.e., the context) in which they operate. What's more, they can dynamically react and respond to context changes in this environment, which is commonly referred to as *context adaptation*.

The ability to dynamically adapt to the context of use is particularly relevant in the field of *mobile computing*. Today's mobile devices (PDAs, mobile phones, GPS, digital cameras and the like) feature a wide spectrum of hardware infrastructure allowing them to directly interact with the environment in which they operate. As such, writing software for such devices becomes quite a challenge, in order to cope with frequent changes in their context of use. The traditional software development approach does not suffice to address this challenge. What is needed are new context-oriented programming languages and modelling languages.

Existing model-driven solutions lack scalability because they tend to provide context-specific models for every possible context of use. Researchers from the AnSyMo lab of Universiteit Antwerpen and the Software Engineering Lab of University of Mons are therefore exploring an alternative based on model transformation. These *model transformations* specify how to adapt a context-specific model to a different context. Together, these model transformations form a specification of the context-sensitive application. Using scripting technology, scripts can be generated from these models, allowing a context-aware application to adapt itself dynamically to changes in its context of use, without any need for recompiling the entire application.



Further Information: *http://win.ua.ac.be/~sdgrand/*
Contact: *Sylvain.Degrandsart@ua.ac.be, Tom.Mens@umons.ac.be, Michael.Hoste@umons.ac.be*

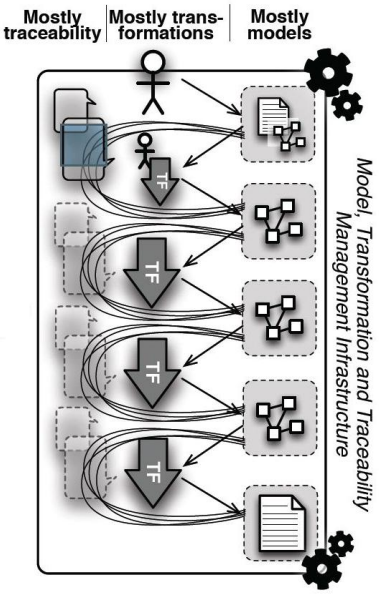
Bert Vanhooff, Stefan Van Baelen, Wouter Joosen, Yolande Berbers

Loosely Coupled Transformation Chains

Model Driven Engineering (MDE) emphasises the use of models in software development. Models allow developers to describe several views on a system each at a certain abstraction level, reducing the system complexity by abstracting away irrelevant technical implementation details. Another key MDE ingredient are model transformations. Transformations may add technical details automatically by translating high-level models into more concrete models (model-to-model) or source code (model-to-text). MDE has a significant impact on the development process and development environment (tools, guidance, etc.). We propose *Model Management* to automate the organisation of large amounts of MDE artefacts, such as models, metamodels, and transformations. The key idea is to represent all MDE artefacts and their relationships as a model itself.

Instead of a single monolithic transformation that translates high-level models into source code directly, we propose to build a *transformation chain* (a *composite transformation*) that gradually transforms models. This approach has two main advantages. First of all, each subtransformation is easier to implement and maintain since it focuses on a single concern. Secondly, opportunities for reuse of transformations grow considerably. We introduce *Transformation Management* as an extension of Model Management to support modelling and execution of transformations and transformation chains in a technology-neutral fashion.

Certain subtransformations require detailed knowledge on relationships between their input models in order to generate a meaningful target model. In order to get this information, implementation-level dependencies between subsequent transformations may arise, negatively affecting reuse potential of these transformations. We propose a technique that uses automatically generated traceability links to expose the necessary relationships and avoid dependencies. In co-operation with AtlanMod research team (INRIA & EMN, France), supporting infrastructure has been built on top of the Atlas MegaModel Management (AM3) framework.



Further Information: *http://distrinet.cs.kuleuven.be*
Contact: *Stefan.VanBaelen@cs.kuleuven.be*

Dennis Wagelaar

Managing Platform Variability through Platform Models

Software-intensive systems are seldom built from scratch nowadays. In most cases, such systems are based on existing *platforms*. A platform can be described as the combination of (visible) underlying hardware and software, on top of which the envisioned software-intensive system will be built. This definition fits within the vision of software families, or Software Product Lines (SPL), in which a product line of software-intensive systems is based on the same central platform. Pohl et al. describe a platform as follows:

'A platform is any base of technologies on which other technologies or processes are built.' [Pohl et al., 2005]

The moment a developer of software-intensive systems chooses for an underlying platform, a dependency on that platform starts growing. This dependency has a primarily technical foundation, because the developed software simply invokes the services of the platform. As the software evolves, more services of the platform are typically being used. This makes it increasingly difficult to switch to another underlying platform. This essentially technical problem is partly the cause of the economic problem of *vendor lock-in*, and varying the platform becomes very hard.

While some solutions transparently bridge platform differences (Java, Flash, ...), they may need to be augmented by other solutions (Design patterns, MDA, ...) that are not transparent to the developer, but allow for bridging platform differences for which no general solution exists. The existence of many different Java sub-platforms already indicates some areas where no general platform bridging solution exists. This is where we propose to use explicit Platform Ontologies, which help track the platform dependencies introduced by the chosen platform-specific mappings (platform-specific code/components/models, model transformations, ...), as well as specific target platforms on which the developed software must run, both based on a central platform ontology.

Our current focus within this research topic is in the first place on extending the existing platform ontologies to (1) support OWL 2 concrete data domains for reasoning on number values (e.g. amount of RAM, battery power, etc.), and (2) support enterprise platforms that are relevant to many software developers (e.g. JEE, EJB, Spring, etc.). We also plan to enhance current model transformation technology, such that the required development and maintenance effort for model transformations is reduced.

[Pohl et al., 2005] Software Product Line Engineering: Foundations, Principles and Techniques by: Klaus Pohl, Günter Böckle, Frank J. van der Linden

Further Information: *http://soft.vub.ac.be/soft/research/platformvariability/start*
Contact: *Dennis.Wagelaar@vub.ac.be*

Serge Demeyer, Dirk Janssens, Hans Vangheluwe

AnSyMo - Antwerp Systems and software Modelling

At the beginning of the MoVES project, Software engineering research in Antwerp was conducted in two distinct research groups co-operating on many fronts. Recently, a Hans Vangheluwe joined our ranks; hence we took the opportunity to join forces — l'Union fait la force! We want to use the opportunity of this newsletter to present to you a brand new research group named AnSyMo: Antwerp Systems and software Modelling.
AnSyMo is a research group investigating foundations, techniques, methods and tools for the design, analysis and maintenance of so-called software-intensive systems. AnSyMo targets four research themes as primary areas of interest. We envision them to be mutually reinforcing themes so that the whole becomes more than the sum of the parts.
[MODELS] One way to tackle the increased complexity of software intensive systems is to represent all knowledge about their structure and behaviour explicitly in the form of models. We design new techniques and build tools for comparing these models, checking their consistency, transforming them into one another, etc. The focus is on Multi-Paradigm (multi-formalism and multi-abstraction) modelling.
[LANGUAGES] An important class of models describes the behaviour of systems (or system components). This is the realm of programming languages semantics in the traditional sense, but also of formalisms such as Petri nets or Statecharts. We contribute to recent work concerning modularity concepts (such as aspects) and their implementation.
[EVOLUTION] Unlike traditional engineering products (e.g., cars and bridges), software systems should be seen as continuously evolving artefacts. We investigate how the availability of models (especially featuring new modularity concepts) may be used to improve the maintainability and evolvability of software systems.
[RESOURCE BOUNDED COMPUTATION] The availability of more but smaller computational devices and the interaction with (physically) external system components has led to the study of hybrid systems. These systems typically consist of distributed components with limited resources; hence computation should be scheduled carefully. We study how models (and simulations thereof) can be used for the efficient management of resources such as memory, time and energy.
Consequently, AnSyMo contributes to work packages 2 and 4. Some of our past research activities are also listed in this newsletter: in particular, the work on delMDSOC, the work on AToM^3, and the work on mining software repositories (joint work with the University of Delft).

Further Information: *http://ansymo.ua.ac.be/*
Contact: *Serge.Demeyer@ua.ac.be, Dirk.Janssens@ua.ac.be, Hans.Vangheluwe@ua.ac.be*

Boris Cule, Bart Goethals

ADReM – Advanced Database Research and Modelling

The ADReM research group is one of the leading groups in database query language research and data mining. ADReM is performing fundamental research on data and knowledge representation, data and knowledge management, and data mining. Our interest lies mostly in the structures, basic properties and the power of existing or new languages, algorithms and methods in the context of processing large quantities of information. This research includes topics such as
[DATA MODELS] The study and characterisation of data models for databases, such as the relational model, the semi-structured model, and RDF.
[DATA LANGUAGES] The study of data management languages, including very formal and abstract languages such as algebras and caculi as well as user languages such as SQL and XQuery.
[DATA MINING] The study of data mining techniques, in order to find patterns and properties in very large databases without asking for explicit information that is contained in the database. This includes the development of mining algorithms for new pattern classes, as well as the theoretical study of data-mining problems and methods in general.
[INDUCTIVE LANGUAGES] The study of inductive databases and query languages, towards the goal of integrating data mining in database systems.
The above research topics are relevant in a MoVES context as well. In particular, data mining algorithms are applicable for mining software repositories, i.e. the systematic perusal of version control systems, bug tracking databases and mailing lists for nuggets of information. An example is shown in the article on mining test code and bug tracking databases.

Further Information: *http://www.adrem.ua.ac.be*
Contact: *Boris.Cule@ua.ac.be, Bart.Goethals@ua.ac.be, Jan.Paredaens@ua.ac.be*

Hans Schippers, Tim Molderez

delMDSOC – A Dedicated Machine Model for MDSOC

"Separation of concerns" is an established principle in software engineering. Ideally, a software system should be modularised in such a way that each module captures a difficult design decision, a so-called concern. This is not a trivial task, as some concerns are typically crosscutting in a certain modularisation, meaning that they appear scattered among multiple modules. Several techniques have been developed which strive to modularise crosscutting concerns, including aspect-oriented and context-oriented programming. We use the term "multi-dimensional separation of concerns" (MDSOC) in order to refer to all of these techniques in general.

Implementations of MDSOC techniques typically operate at the language level, meaning they essentially transform an MDSOC application into an object-oriented one prior to execution. MDSOC mechanisms are expressed in terms of object-oriented mechanisms, resulting in a rather verbose representation. The fundamental problem here is that the machine model which is targeted by MDSOC compilers has no inherent support for MDSOC language mechanisms.

We designed delMDSOC, a dedicated machine model for MDSOC based on the notion of virtual join points, i.e., loci of late binding of messages to functionality. In order to realise this model, we merely depend on the established concepts of objects, message sending and delegation. For validation purposes, we developed informal mappings towards our model, involving four high-level programming languages exhibiting different mechanisms for the modularisation of crosscutting concerns, ranging from classes over aspects to dynamic layers.

The formal semantics of delMDSOC has been expressed as a set of graph rewrite rules, which allows for simulation of machine-level program execution in the AGG tool. Recently, we have extended delMDSOC with actor-based concurrency, allowing for the expression of modularisation mechanisms which deal with parallelism.

Next steps include formal descriptions of refactorings in MDSOC languages, comparison of different MDSOC mechanisms from a semantics perspective, and the development of an actual virtual machine prototype.

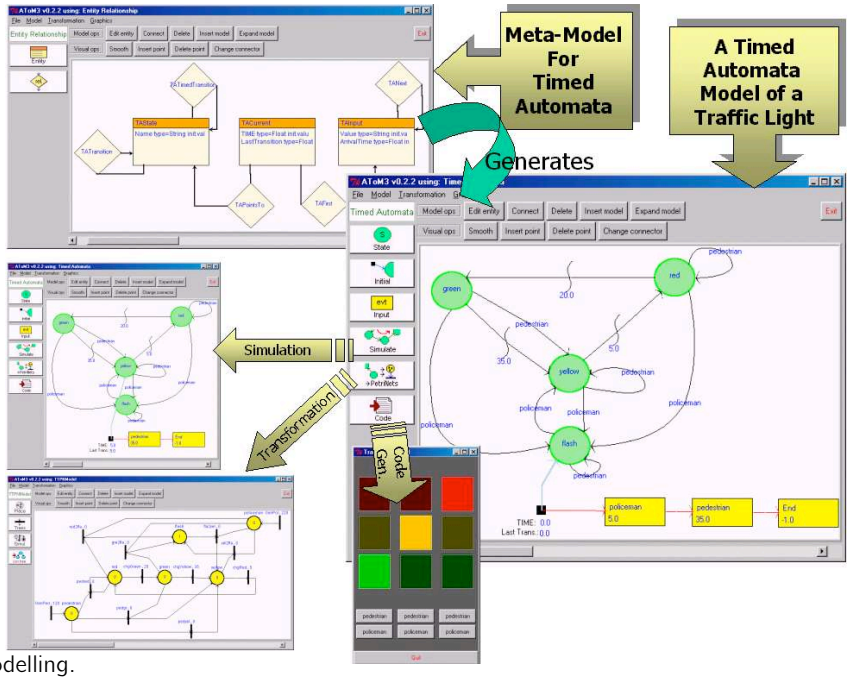
Further Information: *http://fots.ua.ac.be/delmdsoc/*
Contact: *Hans.Schippers@ua.ac.be, Tim.Molderez@ua.ac.be*

Hans Vangheluwe, Bart Meyers

AToM^3

AToM^3, "A Tool for Multi-formalism and Meta-Modelling" was developed by Juan de Lara (now at the Universidad Autónoma de Madrid, Spain) and Hans Vangheluwe (University of Antwerp, Belgium) and their students, while both were at McGill University in Montreal, Canada.

AToM^3 is a prototype environment (shown in the figure) for modelling language engineering through meta-modelling and model-transformation. The environment allows us to try out new ideas in rule-based model transformation, modelling of user interface behaviour, (meta-)model evolution, domain-specific visual modelling, etc. The goal is to evolve Model-Driven Engineering into a well-founded Engineering discipline.



Meta-modelling refers to the specification of different formalisms used to model systems. We have focused on formalisms for analysis, simulation and synthesis of dynamical systems. Model-transformation refers to the (automatic) process of converting, translating or modifying a model in a given formalism, into another model, possibly in a different formalism. In AToM^3, all artefacts are internally represented as typed, attributed graphs.

The environment is now being redesigned from the bottom up, with for example a new graph rewriting kernel allowing different families of rule-based transformation languages, and a new web-browser-based user interface. The new environment is called AToMPM "A Tool for Multi-Paradigm Modelling", highlighting the focus on multi-formalism and multi-abstraction modelling.

Further Information: *http://atom3.cs.mcgill.ca/*
Contact: *Hans.Vangheluwe@ua.ac.be, Bart.Meyers@ua.ac.be*

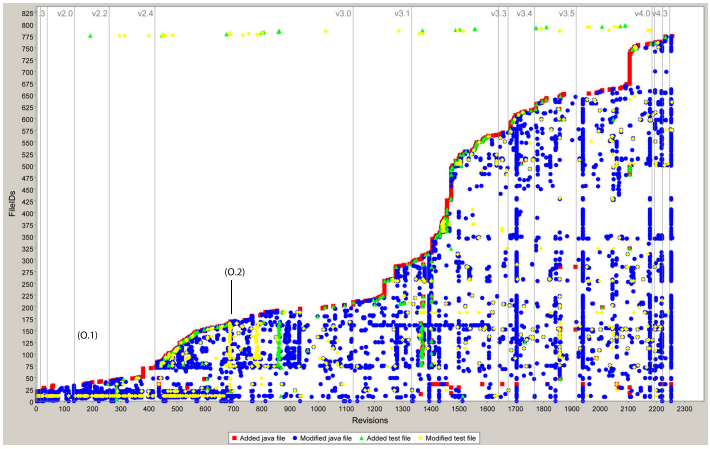
Serge Demeyer, Ahmed Lamkanfi, Andy Zaidman

Mining for Test Co-Evolution and Bug Tracking Databases

The term mining software repositories (MSR) has been coined to describe systematic investigations of artefacts that are produced and archived during software production: the information stored in source code version-control systems (e.g., the Concurrent Versions System (CVS)) requirements/bug-tracking systems (e.g., Bugzilla), and communication archives (e.g., e-mail). These repositories hold a wealth of information about the actual evolutionary path taken to realise a software system, and as such provide a unique opportunity for empirical research.

We mine version-control systems to gain insight in the co-evolution between test code and production code. [1] The figure below shows a partial result; a so-called change history view of CheckStyle. The X-axis represents time, the Y-axis represents test or production code. A dot in the view represents a change as recorded in the version control system: red is newly added production code; blue is modified production code; green is newly added test code and yellow represents modified tests. Views like these allow us to see whether test code changes when the corresponding production code is modified.

We apply text-mining algorithms on bug reports to support bug-triage. Terms like 'crash' or 'failure' serve as good indicators for the severity of a bug, hence we want to use them as indicators. Consequently, we envision a tool that – after a certain 'training period' – provides a second opinion to be used by the development team for classifying incoming bug reports. Based on three cases drawn from the open-source community (Mozilla, Eclipse and GNOME), we experience that given a training set of sufficient size (approximately 500 reports per severity), it is possible to predict the severity with a reasonable accuracy.[2]



[1] Andy Zaidman, Bart Van Rompaey, Serge Demeyer, and Arie van Deursen. Mining software repositories to study co-evolution of production and test code. In Proceedings of the 1st International Conference on Software Testing, Verification and Validation (ICST), pages 220-229. IEEE Computer Society, 2008.

[2] Ahmed Lamkanfi, Serge Demeyer, Emanuel Giger, and Bart Goethals, Predicting the Severity of a Reported Bug, In Proceedings MSR'10 (7th IEEE Working Conference on Mining Software Repositories), May, 2010

Further Information: *http://swerl.tudelft.nl/testhistory*
Contact: *serge.demeyer@ua.ac.be, Ahmed.Lamkanfi@ua.ac.be, A.E.Zaidman@tudelft.nl*

Partners