Dear Reader,

Welcome to the eighth edition of the MoVES newsletter. The MoVES network on Modelling, Verification and Evolution of Software combines the strength of the leading Belgian research teams in software engineering. MoVES also has a number of foreign partners that complement the expertise of the Belgian research teams. This issue of the MoVES newsletter focuses on two of the foreign partners, namely INRIA Lille - Nord Europe (France) and the Delft University of Technology (The Netherlands).

With the MoVES project winding down in just a few months, we want to use the opportunity to inform you of some of the recent research that has been going on at the foreign institutions. In particular, at INRIA Lille investigations were conducted in the areas of: the creation of a reconfigurable middleware platform with aspects and components (Seinturier et al.), the checking of architectural and implementation constraints for domain-specific component frameworks using models (Loiret et al.), the integration of context for the adaptation of applications in ubiquitous environments (Romero et al.) and a context-aware dynamic software product line for mobile applications (Parra et al.).

At Delft University of Technology meanwhile research has focussed on exploring plug-in test suites (work by Greiler et al.), understanding of Ajax web applications (work of Matthijssen et al.), reverse engineering of spreadsheets (work by Hermans et al.), the creation of a domain specific language for mobile devices (mobi – work by Hemel and Visser) and work on integrated language definition testing (Kats et al.).

Enjoy reading!

Andy Zaidman and Laurence Duchien
Reconfigurable Middleware Platform with Aspects and Components

FraSCAti [1] is a middleware platform for developing Service Component Architecture (SCA) based distributed systems. SCA is a standard for distributed Service-Oriented Computing (SOC). The novelty of FraSCAti is to bring run-time adaptation and manageability properties to SCA applications and their supporting platform. As stated by Papazoglou et al., this is a key challenge in SOC research. With FraSCAti, an SCA application can be introspected to discover at run-time its structure, modified dynamically to add new services, reconfigured to take into account new operating conditions. Both the component-based architecture of the system and the binding of this system to external services can be reconfigured. This flexibility and openness at the application level is also offered at the platform level.

FraSCAti is based on three original characteristics. First, FraSCAti adopts a component-based structure for the platform itself, using the same component model as for SCA applications. Second, FraSCAti extends, in an upward compatible fashion, the SCA component model with reflective capabilities. Third, FraSCAti exploits aspect-oriented programming techniques for extending SCA components with nonfunctional services, themselves programmed as SCA components. This results in a component-based structure that is highly modular, extensible and dynamically reconfigurable.

FraSCAti is a project of the OW2 consortium for open source middleware.


Checking Architectural and Implementation Constraints for Domain-Specific Component Frameworks using Models

The Hulotte framework provides a general approach to design domain-specific component frameworks. The framework is based on state-of-the-art technologies, combining Component-Based Software Engineering, Model-Driven Engineering, and code generation techniques. The framework targets the design and the implementation of embedded systems. Component-Based Software Engineering (CBSE) emphasizes software architecture by decomposing systems into logical modules. Within this approach, the first-class design entities are the architectural artifacts, which comprise components, their attributes, and the bindings between their interfaces. The semantics attached to these architectural artifacts depend on the targetted application domain, exposing relevant Domain-Specific Concerns (DSCs) and patterns at the architectural level. The integration of domain-specific concerns is usually performed in an ad-hoc manner, thereby negating the benefits of reuse that component-based architectures are supposed to offer. In order to address this, we have designed the Hulotte framework.

Hulotte allows component framework developers to integrate and implement domain-specific concerns using a dedicated design process. In programming languages one of the roles of the compiler is to check that the input program respects the constraints of the language, giving helpful error messages when it is not the case. Similarly, an architect using a domain-specific component framework is expected to adhere to the rules of the framework. It is therefore necessary, when developing the DSCs, to also specify the domain-specific constraints that govern the use of those concerns.

In [1] we address the problem of the definition of domain-specific constraints with Hulotte. We have identified two kinds of domain-specific constraints: those that DSCs impose on other DSCs, and those that DSCs impose on the source code elements that implement them. We call the former architectural constraints and the latter implementation constraints. Constraints at the architectural level occur due to the extension/narrowing of the component semantics that the DSCs impose. In order to define and check constraints at these different abstraction levels, information not only on the architecture, but also on its implementation are needed. We achieved this by merging a model of the architecture with a model of the implementation using a pivot representation to bind them.

To achieve this, Hulotte proposes:

(i) A model that incorporates structural information on the architecture of the application and its implementation, providing an uniform way of defining domain-specific constraints as invariants,

(ii) A toolchain implementing the complete design process, easily extensible towards arbitrary domain-specific concerns.


Further Information: http://adam.lille.inria.fr/
Contact: Lionel.Seinturier@inria.fr
**SPACES: Integration of Context for the Adaptation of Applications in Ubiquitous Environments.**

The advent and widespread usage of Smartphones combined with the presence of services in Ubiquitous Environments enable the creation of applications able to adapt according to the current environment state. Nevertheless, these adaptations required the integration of information (called context information) regarding the heterogeneity in terms of resources (e.g., execution platforms and protocols) and the user mobility, in order to distribute the adaptation responsibilities.

In order to face the previous issues, we propose SPACES [1], a middleware platform for the integration of context information in ubiquitous environments. Based on the Representational State Transfer (REST) principles, SPACES fosters the notion of Context as a Resource to enable an easy usage of this information. This means that context can be accessed by using standard identifiers (e.g., URLs), addressed via simple interfaces (e.g., GET, POST, PUT and DELETE from HTTP) and represented by applying standard formats (e.g., JSON and XML). Furthermore, SPACES complements REST by bringing discovery capabilities (based on standard protocols such as UPnP and SLP) into the context integration in order to deal with dynamicity in ubiquitous environments.

SPACES is employed to build Ubiquitous Feedback Control Loops (FCLs) [1]. Based on the Autonomic Computing Paradigm, these FCLs permit the context-based adaptation of applications by providing the flexibility required to integrate new participants in the adaptation process (e.g., context-aware applications, services and legacy systems) and the incorporation of new communication mechanisms when required. In the core of the Ubiquitous FCLs, i.e., the decision-making, we employ constraint programming techniques to optimize the selected configuration regarding aspects for providing a better user experience, such as the cost associated with the adaptation, the resources consumed or the QoS offered by the new configuration.


**Further Information:** http://adam.lille.inria.fr/

**Contact:** Laurence.Duchien@inria.fr

---

**CAPucine: A Context-Aware Dynamic Software Product Line for Mobile Applications**

In the recent years, we have witnessed major advances in mobile computing. Modern devices are equipped with a variety of sensors and network interfaces that make them quite versatile. In order to take advantage of all the hardware capabilities and provide a better user experience, software has to be context aware, i.e., it has to monitor the events and information coming from its environment and react accordingly. At the same time, we notice that an important number of such mobile applications share several characteristics regarding its architecture, communication, storage and interfaces. This leads us to consider that context-aware systems can also benefit from the Software Product Line (SPL) paradigm. SPLs were defined to take advantage of commonalities through the definition of reusable artifacts, in order to automate the derivation of multiple products. Nevertheless, SPLs are limited regarding the runtime modifications implied by context awareness. For that, we investigate on Dynamic Software Product Lines (DSPL). A DSPL extends a classic SPLs by providing mechanisms to adapt products at runtime to cope with dynamic changes imposed by context awareness. Our main goal is to unify design and runtime adaptations under the same definition through high-level artifacts.

Concretely, we introduce both: a simple—yet complete—variability model, and a composition model that realizes variability. With the variability model we aim at defining a family of products and at identifying commonalities and variabilities among those products using variants. The composition model on the other side, is based on ideas from Aspect Oriented Software Development (AOSD). We propose two processes of product derivation: design weaving and runtime weaving. Both processes use the same variability and aspect models. We thus allow developers to reuse the same artifacts used for building a software product to adapt it dynamically among various configurations. For the design weaving, we base ourselves on a model driven approach where transformations and code generation are employed to obtain source code from a set of models. For the runtime weaving, we use FraSCAti, a service and component based platform with dynamic properties, to execute reconfigurations during the execution of products. We also use a context manager to process events coming from the environment and make decisions about the adaptation [1].

To validate our approach we define and implement a DSPL for a retail case study. We successfully cover the whole cycle of design derivation and adaptation of software products. The scenario demonstrates the versatility of our approach and in particular the unification achieved through the aspect models used at design time as well as at runtime [2].


**Further Information:** http://adam.lille.inria.fr/

**Contact:** Laurence.Duchien@inria.fr
Spreadsheets are widely used in industry: estimations indicate that 90% of all analysts in industry perform calculations in spreadsheets. Their use is diverse, ranging from inventory administration to educational applications and from scientific modeling to financial systems. A study showed that 95% of U.S. firms, and 80% in Europe, use spreadsheets in some form for financial reporting. Business analysts using spreadsheets usually have very limited training as a programmer. They are end-user programmers, and as such face many of the challenges of professional developers, such as identifying faults, debugging, or understanding someone else’s code.

In our study with the spreadsheet users and developers of the Dutch asset management company Robeco, 70% of the subjects expressed that they have difficulties with understanding spreadsheets received from a colleague [1]. Furthermore, the study showed that most important information needs concern the understanding of the dependencies between worksheets, blocks and cells as inferred by spreadsheet formulas. The support provided by spreadsheets, e.g., Excel’s Audit Toolbar, have been rated insufficient.

We have developed an approach and a tool called GyroSAT that extracts leveled data-flow diagrams from Excel spreadsheets. The approach first determines the cell type of all cells in the spreadsheet: Data, Formula, Label or Empty. Based on this information, data-blocks, maximally connected groups of cells surrounded by empty cells, within a worksheet are identified. After that, the tool creates a data-flow diagram by creating nodes for each cell of type Data and Formula and edges to represent the formula dependencies. Next, the labels describing Data and Formula cells are used to label each node in the diagram. The final step adds levels to the data-flow diagram. A level is introduced for each worksheet within the spreadsheet and for each data block within every worksheet.

![Diagram of spreadsheet analysis](http://www.se.ewi.tudelft.nl)

Further Information: http://www.se.ewi.tudelft.nl
Contact: f.f.j.hermans@tudelft, m.pinzger@tudelft, arie.vandeursen@tudelft.nl

Understanding Ajax web applications

Over the last decade, web development has evolved from creating static web sites to creating rich and highly interactive web applications. One of the enabling technologies for creating these interactive web applications is Ajax (Asynchronous Javascript and XML). A well-known example of an Ajax application is Gmail, which uses Ajax technologies to update only a part of the page when you open an email conversation, and to suggest email addresses of recent correspondents as you type.

Unfortunately, Ajax also makes web development more complex. This motivated us to examine ways to support web developers in maintaining this new breed of web applications. In particular, we constructed a tool called FireDetective that leverages dynamic analysis and helps web developers to better understand Ajax applications [2].

Specific to FireDetective is that it traces both at the client-side (the browser) and the server-side (a J2EE server). The FireDetective visualizer then combines trace information from both sides in one visualization. This setup effectively reduces the number of context switches for the software developer because now he can use one tool to see the client and server interactions instead of two tools.

Through a user study we assessed FireDetective [1]. The conclusion of this study is that FireDetective allows them to understand Ajax applications more effectively, more efficiently and with more confidence. We are currently following up on this research by performing an additional experiment with a company specialized in developing Ajax applications.

![Diagram of understanding Ajax web applications](http://www.se.ewi.tudelft.nl)

Further Information: http://www.se.ewi.tudelft.nl
Contact: a.e.zaidman@tudelft, arie.vandeursen@tudelft.nl


A new generation of mobile touch devices, such as the iPhone, iPad and Android devices, are equipped with powerful, modern browsers. However, regular websites are not optimized for the specific features and constraints of these devices, such as limited screen estate, unreliable Internet access, touch-based interaction patterns, and features such as GPS. While recent advances in web technology enable web developers to build web applications that take advantage of the unique properties of mobile devices, developing such applications exposes a number of problems, specifically: developers are required to use many loosely coupled languages with limited tool support and application code is often verbose and imperative. We have developed mobl, a new language designed to declaratively construct mobile web applications [1]. Mobl integrates languages for user interface design, styling, data modeling, querying and application logic into a single, unified language that is flexible, expressive, enables early detection of errors, and has good IDE support. Mobl is our second case study in the design and implementation of syntactically integrated DSLs, DSLs that integrate sub-languages for multiple application aspects, enabling static verification of the entire application. Previously, we developed WebDSL, a DSL to develop data-driven web applications.

Plug-in architectures are widely used for complex systems such as browsers, development environments, or embedded systems, since they support modularization, product extensibility, and runtime product adaptation and configuration. Such products are assembled from plug-ins, and their functionality can be enriched by adding or configuring plug-ins. The plug-ins themselves consist of multiple plug-ins, and offer dedicated points through which their functionality can be influenced. A well-known example of such an architecture is Eclipse, best known for its use to create a series of extensible IDEs. In order to test systems built from plug-ins developers use extensive automated test suites. Unfortunately, current testing tools offer little insight in which of the many possible combinations of plug-ins and plug-in configurations are actually tested. To remedy this problem, we developed five architectural views that provide an extensibility perspective on plug-in-based systems and their test suites [1]. The views are implemented in ETSE, the Eclipse Plug-in Test Suite Exploration tool, and combine static and dynamic information on plug-in dependencies, extension initialization, and extension and service usage. The views can be used by developers to understand how the integration of multiple plug-ins is tested. We evaluated the proposed views by analyzing three open source Eclipse plug-in systems: eGit, Mylyn, and a Trac-connector for Mylyn.


Further Information: http://swerl.tudelft.nl/bin/view/Main/ETSE
Contact: m.s.greiler@tudelft.nl, arie.vandeursen@tudelft.nl, h.g.gross@tudelft.nl