Editorial

The MoVES network on Modelling, Verification and Evolution of Software combines the leading Belgian research teams in software engineering, with recognised scientific excellence in model-driven engineering, software evolution, formal modelling and verification, and aspect-oriented software development.

This month’s issue of the MoVES newsletter focusses on the work package “Software Decomposition, Composition and Recomposition” that explores building reusable support for self-adaptive and context-aware system software using context-oriented and aspect-oriented programming techniques. In particular, this newsletter contains contributions by Jorge Vallejos et al. on support for context-dependent method dispatch (a collaboration between VUB and UCL), by Dave Clarke et al. on the semantics of context-oriented programming languages (a collaboration between K.U.Leuven, VUB and University of Chile), by Pascal Costanza et al. on safe reconfiguration in the presence of dynamically scoped adaptations (a collaboration between VUB and K.U.Leuven), and by Arnaud Hubaux et al. on support for concurrent feature-based configuration (a contribution by FUNDP).

It also zooms in on a selection of recent research contributions of the DistriNet research group at K.U.Leuven. These works are a representative sample that covers the current focus and scope of the research group: distributed software, secure software and research on software engineering techniques in these two domains. In particular, the following contributions have been included in this newsletter: Energy-efficient Semantic Publish/Subscribe for Battery-powered Systems by Davy Preuveneers et al., Decentralized Coordination of Plug-in Hybrid Vehicles for Imbalance Reduction in a Smart Grid by Stijn Vandael et al., Expressive Modular Fine-Grained Concurrency Specification by Bart Jacobs et al., Non-interference Through Secure Multi-Execution by Dominique Devriese et al., From Modularized Design to Code: The cost of Maintenance by Aram Hovsepyan et al., and Domain-driven Discovery of Stable Abstractions for Aspectual Interfaces by Dimitri Van Landuyt et al.

Eddy Truyen, Stefan Van Baelen and Wouter Joosen

Upcoming Events & Publications

- Recent and Upcoming Publications:

- Upcoming Events
  - LICIT (LEUVEN Center on Information and Communication Technology) workshop: Workshop on Software Product Line Engineering, May 10, 2011
  - Variability@ER2011: Workshop on Software Variability Management, co-located with ER 2011, 31 October – 3 November, 2011
Predicated Generic Functions: Enabling Context-Dependent Method Dispatch

Software services should perceive and adapt to changes in their execution environment, so that they match client expectations and needs as closely as possible. Unfortunately, the lack of programming language support to encode context-dependent behaviour has a negative impact on software quality, in particular because it forces programmers to scatter the dependencies throughout application code in the form of conditional statements and design patterns. More advanced solutions still do not offer the necessary linguistic support to write clean context-aware code.

We have therefore developed a generic function-based multiple dispatch mechanism, called predicated generic functions[1]. This solution builds on the power and generality of predicate dispatch[2], which offers fine-grained control on method applicability by means of logical predicates, albeit with certain limitations regarding method overriding ambiguities. Predicated generic functions alleviate these limitations. Namely, instead of requiring a logical implication order between predicates, predicated generic functions enable users to manually establish an order between logically unrelated predicates. As illustrated in the figure, predicated generic functions contain not only the methods with a common name and argument structure, but also the predicates on which such methods can be specialised. A method is selected for execution when its predicate expression is satisfied, and the order of the predicates specified in a generic function determines the order of applicability of its methods.

The mechanism of predicated generic functions has been implemented in Lambic[3], a prototype extension of the Common Lisp programming language. Lambic allows application developers to modularise behavioural adaptations in methods and declaratively specify the context conditions for these adaptations as predicates. Manual ordering of predicates in generic functions provides fine-grained control over the composition of adaptations, ensuring that the “most suitable” composition of behaviour is selected by the dispatch mechanism, thus complying with any domain-specific constraint.

Further Information: [http://soft.vub.ac.be/lambic](http://soft.vub.ac.be/lambic)
Contact: jvallejo@vub.ac.be

Semantics of Context-Oriented Programming Languages

In context-oriented programming (COP) the context in which expressions evaluate can be adapted as a program runs. COP provides a degree of flexibility beyond object-oriented programming, while arguably retaining more modularity and structure than aspect-oriented programming. Although many languages exploring the context-oriented approach exist, very little work has been done towards providing formal type-safe dynamic semantics of these languages. We address this shortcoming by providing a concise syntax-based formal semantics for context-oriented programming with layers, as witnessed by ContextL, ContextJ*, and other languages. Our language is based on Featherweight Java extended with layers and scoped layer activation and deactivation. As layers may introduce methods not appearing in classes, we also give a static type system that ensures that no program gets stuck (i.e., there exists a binding for each dispatched method call) [1].

Context-oriented programming treats execution context explicitly and provides means for context-dependent adaptation at runtime. This is achieved, for example, using dynamic layer activation and contextual dispatch, where the context consists of a layer environment of a stack of active layers. Layers can adapt existing behaviour using proceed to access earlier activated layers. A problem arises when a call to proceed is made from within a closure that escapes the layer environment in which it was defined. It is not clear how to interpret proceed when the closure is subsequently applied in a different environment, because the layers it implicitly refers to (such as the original layer and/or the remaining layers) may no longer be active. In [2], we describe the problem in detail and present some approaches for dealing with it, though ultimately we leave the question open.

Contact: dave.clarke@cs.kuleuven.be
Ensuring that software can display different behaviour in different use contexts requires adapting software at runtime in dynamically created scopes (e.g. in a thread, in a client session, in a collaboration). Context-Oriented Programming (COP) offers dedicated language constructs for performing such dynamically scoped adaptations.

COP supports powerful customizations such as concurrent customization where independent clients can concurrently customize the functionality of a shared component, and system-wide customizations that cross technical boundaries. However, like any dynamic software adaptation technique, COP hits a conceptual barrier when new variations of existing program entities are integrated into a running system: if applied uncarefully, dynamically scoped adaptations may lead to faulty behaviour especially when the adaptation is activated at stages where the affected system components themselves are in a transitional, inconsistent state. This remains a critical consistency issue, especially in distributed systems. For example, dynamically adapting the scheduling algorithm for processing stock quotes in a financial service line architecture depending on the user class must be coordinated across the entire service line (see Figure).

The FWO project RECOCO (Reflective Reconfiguration Support for Context-Oriented Programming) is a joint collaboration between the VUB and K.U.Leuven to create a consistency management framework on top of COP that ensures system-wide reconfiguration consistency.

A key hypothesis of the work is that the manner of preserving system consistency should be an application-specific endeavour. Inherently, the ways of managing state consistency is highly specific to the application at hand and therefore requires additional application-specific logic from the system itself.

In other words, not a "one-size-fits-all" solution should be hardcoded inside the consistency management framework. Instead, the consistency management framework needs to offer a set of abstractions that can be used to express application-specific policies for dealing with state inconsistencies.

Towards Support for Concurrent Feature-based Configuration

In software product lines, engineers derive products using feature-based configurators. Such tools do not scale well to complex (non-linear, multi-user) configuration processes. We have addressed this issue by extending a feature-based configurator (SPLOT) with multi-view support and by integrating it with a workflow management tool (YAWL). One of the challenges of non-linear multi-user configuration is conflict management. As users concurrently configure their views of the feature diagram (FD), inconsistencies between their respective decisions can occur.

At the time being, our toolset prevents inconsistencies during configuration with a lock on the configuration space. The underlying assumption here is that views are configured synchronously. Yet, other policies are needed in case this assumption does not hold. In practice, the requirements for the configuration environment are many and can diverge. Different usage scenarios, different reporting and resolution policies, multiple perspectives on the same FD, and controlled configuration processes all impose different constraints.

Arguably, most of these requirements correspond to high-level constraints on the configuration front-end. At a lower-level though, these different use cases converge upon the same requirements for the back-end, i.e. the reasoning engine. The duty of the front-end is to process the raw results returned by the back-end to meet the specific requirements of a configuration environment. In other words, different front-ends implementing distinct requirements can be plugged onto the same back-end.

Our current research focuses on the low-level concepts of concurrent feature-based configuration. Its objective is to provide (1) a precise specification of these concepts and (2) efficient algorithms for conflict detection, reporting and resolution that can serve as a robust back-end for concurrent configuration.
**µC-SemPS: Energy-efficient Semantic Publish/Subscribe for Battery-powered Systems**

Our lightweight semantic publish/subscribe system µC-SemPS [1] is targeted towards battery-powered micro-controllers and wireless sensor nodes in ubiquitous computing environments. The key challenge that we address is to minimize the overall energy consumption for subscription matching and delivery. Our system relies on an efficient representation of semantic subscriptions and favours computation over the more expensive wireless communication for the semantic matching and routing of events. When compared to more conventional pub/sub routing implementations, experimental results from network and energy simulations with MiXiM, an OMNeT++ modelling framework for wireless mobile networks, show that our approach for routing semantic events significantly reduces energy consumption.


**Further Information:** [http://distrinet.cs.kuleuven.be/people/davy](http://distrinet.cs.kuleuven.be/people/davy)
**Contact:** davy.preuveneers@cs.kuleuven.be

---

**Decentralized coordination of plug-in hybrid vehicles for imbalance reduction in a Smart Grid**

Intelligent electricity grids, or ‘Smart Grids’, are being introduced at a rapid pace. Smart grids allow the management of new distributed power generators such as solar panels and wind turbines, and innovative power consumers such as plug-in hybrid vehicles. One challenge in Smart Grids is to fulfill consumer demands while avoiding infrastructure overloads. Another challenge is to reduce imbalance costs: after ahead scheduling of production and consumption (the so-called ‘load schedule’), unpredictable changes in production and consumption yield a cost for repairing this balance.

To cope with these risks and costs, we propose a decentralized, multi-agent system solution for coordinated charging of PHEVs in a Smart Grid [1]. Essentially, the MAS utilizes an “intention graph” for expressing the flexibility of a fleet of PHEVs. Based on this flexibility, charging of PHEVs can be rescheduled in real-time to reduce imbalances.

We discuss and evaluate two scheduling strategies for reducing imbalance costs: reactive scheduling and proactive scheduling. Simulations show that reactive scheduling is able to reduce imbalance costs by 14%, while proactive scheduling yields the highest imbalance cost reduction of 44%.


**Further Information:** [http://distrinet.cs.kuleuven.be/people/stijn](http://distrinet.cs.kuleuven.be/people/stijn)
**Contact:** stijn.vandael@cs.kuleuven.be

---

**Expressive Modular Fine-Grained Concurrency Specification**

Compared to coarse-grained external synchronization of operations on data structures shared between concurrent threads, fine-grained, internal synchronization can offer stronger progress guarantees and better performance. However, fully specifying operations that perform internal synchronization modularly is a hard, open problem. The state of the art approaches, based on linearizability or on concurrent abstract predicates, have important limitations on the expressiveness of specifications. Linearizability does not support ownership transfer, and the concurrent abstract predicates-based specification approach requires hardcoding a particular usage protocol. We propose a novel approach that lifts these limitations and enables fully general specification of fine-grained concurrent data structures [1]. The basic idea is that clients pass the ghost code required to instantiate an operation’s specification for a specific client scenario into the operation in a simple form of higher-order programming.

We machine-checked the theory using the Coq proof assistant. Furthermore, we implemented the approach in our program verifier VeriFast and used it to verify two challenging fine-grained concurrent data structures from the literature: a multiple compare-and-swap algorithm and a lock-coupling list.


**Further Information:** [http://distrinet.cs.kuleuven.be/people/bartj](http://distrinet.cs.kuleuven.be/people/bartj)
**Contact:** bart.jacobs@cs.kuleuven.be
A program is defined to be noninterferent if its outputs cannot be influenced by inputs at a higher security level than their own. Various researchers have demonstrated how this property (or closely related properties) can be achieved through information flow analysis, using either a static analysis (with a type system or otherwise), or using a dynamic monitoring system. We propose an alternative approach [1], based on a technique we call secure multi-execution. The main idea is to execute a program multiple times, once for each security level, using special rules for I/O operations. Outputs are only produced in the execution linked to their security level. Inputs are replaced by default inputs except in executions linked to their security level or higher. Input side effects are supported by making higher-security-level executions reuse inputs obtained in lower-security-level threads.

We show that this approach is interesting from both a theoretical and practical viewpoint. Theoretically, we prove for a simple deterministic language with I/O operations, that this approach guarantees complete soundness (even for the timing and termination covert channels), as well as good precision (identical I/O for terminating runs of termination-sensitively noninterferent programs).

On the practical side, we present an experiment implementing secure multi-execution in the mainstream Spidermonkey Javascript engine, exploiting parallelism on a current multi-core computer. Benchmark results of execution time and memory for the Google Chrome v8 Benchmark suite show that the approach is practical for a mainstream browser setting. Certain programs are even executed faster under secure multi-execution than under the standard execution.

We discuss challenges and propose possible solutions for implementing the technique in a real browser, in particular handling the DOM tree and browser callback functions. Finally, we discuss how secure multi-execution can be extended to handle language features like exceptions, concurrency or nondeterminism.

The benefits of defining explicit pointcut interfaces in aspect-oriented applications have been advocated by many. A pointcut interface represents an abstraction of the join points in the base application to which aspects are composed. Although many techniques and notations to model explicit pointcut interfaces exist, there is currently a lack of methodological guidance how to design pointcut interfaces which are (i) robust in light of software evolution and (ii) can be reused by multiple aspects.

In order to attain these qualities, a pointcut interface should preferably expose stable abstractions in the problem domain. We have proposed a top-down method for designing such stable pointcut interfaces [1]. The method builds upon state-of-the-art domain analysis techniques and employs systematic re-engineering of use case models to discover stable abstractions that are anchored in the domain model of the application. At architecture creation time, these stable domain abstractions are mapped onto pointcut interfaces.

Additionally, we provided algorithmic procedures for use case re-engineering and guidelines for architecture creation. Based on these, we have developed a prototype implementation [1], which in turn improves the ease of adoption. We conducted the validation of this method across three large-scale realistic case studies, where we observe that the resulting pointcut interfaces can be reused for implementing the composition logic of different aspects without requiring modification to their pointcut signatures.

These case studies are: (i) an electronic banking system, (ii) a car crash management system (CCMS), and (iii) a digital news platform. The CCMS was provided to us by a third party, and as such serves as an independent validation of our work [2]: we provide concrete evidence from the CCMS of how our method can be adopted to realize the above-mentioned goals. In [3], we have shown how a component-based version of the news publishing platform is refactored using AOSD techniques, and how this improves variability and evolvability of the publishing infrastructure.