Model-Driven Engineering of Self-Adaptive Software

*UCT CS Colloquium*
University of Cape Town, South Africa, 19th August 2015

- Thomas Vogel
- @tomvog
- System Analysis and Modeling Group
- Hasso Plattner Institute
- University of Potsdam, Germany
Continuous Change

- **Software aging** [Parnas, 1994]
  - When not being adapted to changing user needs (lack of movement)
  - Adapting the software often violates the design (ignorant surgery)

- **Lehman’s laws of software evolution** (real-world applications)
  [Lehman and Belady, 1985, Lehman and Ramil, 2001]
  1. A “system must be continually adapted else it becomes progressively less satisfactory in use”
  6. “The functional capability of [...] systems must be continually increased to maintain user satisfaction over the system lifetime”

⇒ **Software Evolution and Maintenance**
  [Mens and Demeyer, 2008, Mens et al., 2010, Mens et al., 2014]
Software Evolution Process [Sommerville, 2010]

- Performed by different groups of people (support staff, developers, ...)
  [Kitchenham et al., 1999]
- Follows a higher-level management process [Kitchenham et al., 1999]
- Enacting a release during scheduled system downtimes (stop-and-go maintenance) [Pezzè, 2012]

⇒ Process is costly, introduces delays, and affects availability
Software systems that are...

- **context-aware** (pervasive computing [Weiser, 1991, Satyanarayanan, 2001], internet of things [Perera et al., 2014])
  - timely changes
  - individual changes
- **mission-critical/dependable** [Shaw, 2002]
  - high or permanent availability
- **complex** (ultra-large-scale [Northrop et al., 2006], system of systems [Valerdi et al., 2008])
  - costs
  - dynamic integration
  - shutdown not feasible
- ...

⇒ Efforts and feasibility of traditional software evolution process?
⇒ **Built-in evolution/adaptation process?**
Self-Adaptive Software [Cheng et al., 2009, de Lemos et al., 2013]

“systems that are able to modify their behavior and/or structure in response to their perception of the environment and the system itself, and their goals” [de Lemos et al., 2013, p. 1]

Observations:
- Self-*: configuring / optimizing / healing / protecting / managing / ...  
- Shift responsibility for adaptation from developers to the system  
- Shift software engineering activities from dev. time to runtime  
- Blurring boundary between development time and runtime

Goal:
- Automated and dynamic adaptation  
- Mitigating the growing costs, complexity, and diversity of adaptation
Feedback Loop [Kephart and Chess, 2003, Brun et al., 2009]

- Often inspired by control theory [Filieri et al., 2015]
- Turns an open-loop into a closed-loop system [Salehie and Tahvildari, 2009]
- **Architectural** blueprint: separating domain and adaptation concerns
  - Similar to computational reflection [Maes, 1987]
- Knowledge: policies and a representation (reflection) of the adaptable software [Huebscher and McCann, 2008]
  - e.g., event-condition-action rules and an architectural representation
Engineering Self-Adaptive Software

State of the Art

- Aims for reducing development efforts
- Typically, frameworks for feedback loops
  - Customization such as injecting policies and a representation
  - Partial generation of feedback loops based on policies

Some Drawbacks

- No explicit specification and design of the feedback loops
- Closed approaches
  - Prescribe the structure and number of feedback loops
  - Restrict the techniques/types of knowledge (policies, representation,...)
- Gap between the development and runtime environments
The term Model-Driven Engineering (MDE) is typically used to describe software development approaches in which abstract models of software systems are created and systematically transformed to concrete implementations.

[France and Rumpe, 2007]
# Engineering Self-Adaptive Software with EUREMA

## Side note: Model-Driven Engineering (MDE)

### Goals [France and Rumpe, 2007]
- Mitigating the gap between the problem and solution space
  - Avoiding accidental complexity of closing the gap manually
- Raise the level of abstraction (domain-specific languages & models)
- Automating development: transformation and generation
- Early analysis and quality assurance

### Promises
- “Industrializing” software development [Greenfield and Short, 2003]
- Improve developers’ productivity and software quality
- Reduce costs and time to market
### Side note: Model-Driven Engineering (MDE)

“In our broad vision of MDE, models [...] are also the primary means by which developers and other systems understand, interact with, configure and modify the runtime behavior of software.”

[France and Rumpe, 2007]

### Goals of “runtime models”

- Abstractions of runtime phenomena
- Automate runtime adaptation
- Analyze running software systems
EUREMA (Executable Runtime Megamodels)

**Domain-specific modeling language**
- Uses feedback loop concepts
  - MAPE activities, runtime models, ...
- Explicit design of feedback loops
- Allows freely modeling feedback loops
  - Structure and number of loops
  - Techniques and types of models

**Runtime Interpreter**
- EUREMA models are kept alive at runtime
- Directly executed by the interpreter
- No generation/translation steps
  - No gap between dev. and runtime env.
- Flexibility to adapt feedback loops
Language Overview

- Graphical modeling language
- Two kinds of diagrams

Feedback Loop Diagram (FLD)

- FLD: activities + control flow, runtime models + their usage (behavior)
Language Overview

- Graphical modeling language
- Two kinds of diagrams

**Feedback Loop Diagram (FLD)**

- FLD: activities + control flow, runtime models + their usage (behavior)
- LD: layers, white/black-box modules + their relationships (structure)
  - Trigger of modules: <events>; <period>; <initialState>;
Language Overview

- Graphical modeling language
- Two kinds of diagrams

**Feedback Loop Diagram (FLD)**

**Layer Diagram (LD)**

- FLD: activities + control flow, runtime models + their usage (behavior)
- LD: layers, white/black-box modules + their relationships (structure)
  - Trigger of modules: <events>; <period>; <initialState>;
- FLDs and LD are kept alive at runtime and executed by an interpreter
Modularity

- Multiple FLDs for one feedback loop
- **Complex model operation** to invoke an FLD (entries and exists)
- Binding in the LD
Modularity

- Multiple FLDs for one feedback loop
- **Complex model operation** to invoke an FLD (entries and exists)
- Binding in the LD
Variability

- Alternative modules as **variants**
- Rebinding to switch between alternatives
- Design-time and runtime
- Example: different analysis techniques

- The same applies to implementations (black-box modules) of basic model operations
- Example: different monitoring techniques
Multiple Feedback Loops

- Multiple concerns to be managed.
- Competing concerns and interferences → **coordination**

**EUREMA**

- Modeling the synchronized execution of feedback loops.
- Model operation implementation realizes the coordination mechanism (e.g., utility functions or voting).

---

**Multiple Feedback Loops**

- Multiple concerns to be managed.
- Competing concerns and interferences ⇒ **coordination**
**Multiple Feedback Loops II**

**Independent execution**

- Individual trigger for each feedback loop
- Potentially, concurrent execution of different feedback loops
- Possibility to implicitly synchronize the execution by triggers (e.g., appropriate frequencies of execution runs)
Multiple Feedback Loops III

Sequencing Complete Feedback Loops

- Explicitly modeling the synchronized execution
- MAPE for self-repair $\rightarrow$ MAPE for self-optimization
Multiple Feedback Loops IV

Sequencing Adaptation Activities of Feedback Loops

- More fine-grained synchronization (activities vs. whole feedback loop)
- Interleaved execution of different feedback loops
- $M \rightarrow A+P$ for self-repair $\rightarrow A+P$ for self-optimization $\rightarrow E$

Thomas Vogel | Model-Driven Engineering of Self-Adaptive Software | UCT CS Colloquium | 19th Aug 2015
Evaluation

- mRUBiS as a playground
- Two cases
  - Self-healing
  - Self-optimization
- Compare alternative solutions
  - Models vs. code
  - State- vs. event-based loops

  with respect to
  - Development costs
  - Runtime efficiency
- Applied EUREMA to other approaches
  - Rainbow, DiVA, PLASMA
Conclusion

Summary and contributions of EUREMA

1. Integrated MDE approach
2. Open approach
3. Seamless Integration of Development and Runtime Environment
4. Adaptation and Evolution of Feedback Loops
5. State- and Event-Based Feedback Loops

Future Work

- Distributed feedback loops and decentralized adaptation
- Concurrent execution of interdependent feedback loops
- Model-based techniques to analyze and test EUREMA models

Cyber-physical systems: Driving force for innovation in mobility, health, energy and production.
acatech (National Academy of Science and Engineering) Position Paper,
http://www.acatech.de/de/publikationen/stellungnahmen/acatech/detail/artikel/cyber-physical-systems.html.

Engineering Self-Adaptive Systems through Feedback Loops.


Software Engineering meets Control Theory.
to appear.

Model-driven development of complex software: A research roadmap.
In 2007 Future of Software Engineering, FOSE ’07, pages 37–54. IEEE.

Rainbow: Architecture-Based Self-Adaptation with Reusable Infrastructure.

Software factories: Assembling applications with patterns, models, frameworks and tools.
In Companion of the 18th Annual ACM SIGPLAN Conference on Object-oriented Programming, Systems, Languages, and Applications, OOPSLA ’03, pages 16–27. ACM.

A survey of autonomic computing.
References II

   The Vision of Autonomic Computing.

   Towards an ontology of software maintenance.

   Program evolution: processes of software change.
   Academic Press Professional, Inc., San Diego, CA, USA.

   Rules and tools for software evolution planning and management.

   Concepts and experiments in computational reflection.

   Software Evolution.
   Springer.

   Guest editors’ introduction: Software evolution.

   Evolving Software Systems.
   Springer.

   Models@ Run.time to Support Dynamic Adaptation.
   Computer, 42(10):44–51.

   Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA.

   Software aging.

   Context aware computing for the internet of things: A survey.
References III

From off-Line to continuous on-line maintenance.
In 28th IEEE International Conference on Software Maintenance, ICSM ’12, pages 2–3. IEEE.

Self-adaptive software: Landscape and research challenges.

Pervasive Computing: Vision and Challenges.

Everyday dependability for everyday needs.
In Supplemental Proceedings of the 13th International Symposium on Software Reliability Engineering, ISSRE ’02, pages 7–11. IEEE.
(keynote).

Software Engineering.
Addison-Wesley, 9 edition.

Plasma: A plan-based layered architecture for software model-driven adaptation.

A research agenda for systems of systems architeccting.


Model-Driven Engineering of Self-Adaptive Software with EUREMA.

The Computer for the 21st Century.