A Reflective Approach to Dynamic Software Evolution

Peter Ebraert

Programming Technology Lab
Faculty of Sciences - Vrije Universiteit Brussel
Overview

- Problem Statement
- Towards Separated Concerns
- Towards Dynamic Software Evolution
- Conclusion
- Issues
Problem Statement

- **Software Evolution is Unavoidable**
- **Critical Systems**
  - Systems that “cannot” be shut down
  - Web services, Telecommunication switches, Banking Systems, Airport Traffic control systems, GPS satellite update...

**What if a small part of those systems has to evolve?**
Towards Separated Concerns

- Every concern implemented as a separate entity.
  - Function, ADT, class, component, aspect, ...

- Advantages
  - No scattered code
  - Every entity can evolve separately
  - Easier to maintain

```java
tomtourwe.doResearch(loadsOfMoney);
```
Towards Dynamic SW Evolution

Goal

- **Divide and conquer!**
- **Allowing every entity to evolve separately**
- **System evolution -> Entity evolution**
  - Entity addition
  - Entity removal
  - Entity modification
- **Dynamic comes in when this is done at runtime**
Towards Dynamic SW Evolution
Reflective Systems

- **Able to reason about itself**
  
  - 2 levels of calculation: base level, metalevel
  
  - Causal connection between the 2 levels
    
    - Base level application has access to its metalevel representation on the base level
    
    - A change of the metalevel representation impacts the base level application.

---

Peter Ebraert - Vrije Universiteit Brussel

6 / 10
Towards Dynamic SW Evolution
The Framework

- **Two layered architecture**
  - Base level: instrumented application
  - Metalevel: monitor
    - Gets control on every inter-entity communication
    - Propagates control to the adequate base-level entity
Towards Dynamic SW Evolution
The Runtime API

- Allows runtime interaction with the system
  - Modification of the base-level applications representation
- API functions on the Monitor
  - Adding an entity
  - Removing an entity
  - Modifying an entity
    - Deactivate the entity -> Queue all messages to it
    - Transfer the state -> Programmers decision
    - Activate the entity -> Execute all queued messages
Conclusion

○ **Two-step solution for dynamic evolution**

1. Make the system well modularized
2. Control the instrumented base application by a metalevel monitor
   -> Use reflective programming capabilities

○ **Works for a lot of programming styles**

   *Object-oriented, aspect-oriented or any other, as long as it is well modularized.*
Issues

○ Issues
  
  Do we really want D.E.?
  Are there good alternatives?
  Does DAOP allow D.E.?
  Does Reflection allow D.E.?

  Existing Instances
  State Mapping
  Running Threads with D.E.
  Aspect Composition

○ References

  ○ Peter Ebraert and Tom Tourwé
    A Reflective Approach to Dynamic Software Evolution
    In the proceedings of the Workshop on Reflection, AOP and Meta-Data for Software Evolution (RAM-SE’04) in conjunction with the European Conference on Object Oriented Programming (ECOOP 2004), 15th of June 2004, Oslo Norway

  ○ Peter Ebraert and Eric Tanter
    A Concern-based Approach to Dynamic Software Evolution
    In the Dynamic Aspects Workshop (DAW) proceedings in conjunction with the conference on Aspect Oriented Software Design (AOSD 2004), March 22-26 2004, Lancaster UK