Languages, Composition and Verification @ DistriNet & SOM

Wouter Joosen, Adriaan Moors, Frank Piessens, Eddy Truyen, Stefan Van Baelen.
Overview

- Main actors
- Selected topics
  - Programming Language Research (Adriaan)
  - Modeling and MDD Research (Stefan)
  - Composition (Eddy)
  - Verification (Frank)
  - [Postponed: From requirements to design (Riccardo Sc.)]
- Concluding Remarks
Main Actors

- Faculty
  - Eric Steegmans (SOM)
  - Yolande Berbers (Embedded Systems), Tom Holvoet (Multi-Agent Systems), Wouter Joosen, Frank Piessens – all DistriNet

- PostDocs
  - Lieven De Smet, Bart De Win, Bart Jacobs, Riccardo Scandariato, Eddy Truyen, Stefan Van Baelen, Yves Vandewoude, Danny Weyns.
I. RESEARCH ON PROGRAMMING LANGUAGES (related to WP1)

Frank Piessens, Wouter Joosen, Eric Steegmans, Marko van Dooren, Adriaan Moors, Sven De Labey, Koen Vanderkimpen
High-level overview

- Goal of our programming languages research:
  - Detecting as many bugs as early as possible by leveraging the type system, with minimal effort by the programmer.
  - Bugs: security, application-specific, concurrency,…

- Approaches
  - Language Extensions
  - Extensible Languages
Language Extensions

- Higher-kindled Types for Scala 2.5 [Adriaan]
  - Abstracting over type constructors
- First-class Inheritance Relations [Marko’s PhD]
  - First-class code and subtyping inheritance to increase reuse potential
- Anchored Exception Declarations [Marko’s PhD]
  - Reducing the pain of checked exceptions
Extensible Languages

- Chameleon: framework for modelling OO languages [Marko’s PhD]
- Extensible type system for Scala [Adriaan]
  - Modular Type System
    - Software *engineering* on the level of types (Sheard: $\Omega$mega)
      - Apply same principles to develop type system modules
    - Pay-as-you-go (currency: type annotations)
  - Optional Type System (Bracha: “pluggable type system”)
    - Allow programmer to completely *turn off type system*
    - Type system must not influence execution

- What Java should have been (on JVM & .NET)
- Happy marriage of FP & OO (objects are core)
- Interesting type system
  - Essence of Haskell&SML ∩ OO (¬ purity, laziness,...)
- Unifies OO- & FP-style abstraction for types and values
  - Components
    - Provided interface: abstract class
    - Requirements: explicit self type, abstract type&value members
    - Composition: mixin composition
- DSL’s as libraries (parsers, actors, FRP, Prolog,...)
Higher-kindied Types in Scala

- With just parametric polymorphism:

```scala
trait Collection[a] {
  def map[b](f: a => b): Collection[b] = ... }

class List[a] extends Collection[a] { ... }
List(1, 2, 3).map(_ + 2) : Collection[Int]
```

- With type constructor polymorphism:

```scala
trait Collection[a, m[_]] {
  def map[b](f: a => b): m[b] = ... }

class List[a] extends Collection[a, List] { ... }
List(1, 2, 3).map(_ + 2) : List[Int]
```
Other Interests

- Theory of PL’s
  - Classic pencil&paper calculi: λ + ext’s, FJ, ClassicJava, vObj, FS,…
  - Mechanisation: Twelf (own calculus: Scalina)

- Advanced FP techniques in Scala
  - Datatype-Generic Programming [WGP06]
  - Combinator Parsing
    - E.g., \texttt{def target : Parser[Term] = ("new" ~! typeP ^^ New l ... )}
  - Idioms (Applicative Functors)
    - + combinator parsing: \texttt{def target = New ("new", typeP) l...}
Conclusions

- **Theory**
  - Investing in mechanisation of meta-theory
- **Practice**: extend existing languages
  - Chameleon as platform for exploration
  - Experience with extending the Scala compiler
- The middle road
  - E.g., Scala’s FP & OO marriage
- The road ahead: **extensible type systems**
  - Inspired by current FP research on type-level computation
II. Research on Model-Driven Development (related to WP2)

Stefan Van Baelen, Wouter Joosen, Yolande Berbers, Bert Vanhooff, Aram Hovsepyan, Didier Delanote (DistriNet)
Eric Steegmans, Geert Delanote (SOM)
Problem Statement

- Improving MDD Process
  - Large, Complex, ‘Hard-Coded’ Transformations
Problem Statement

- Improving MDD Process
  - Large, Complex, ‘Hard-Coded’ Transformations
  - Often restricted to a 3-Step Approach
Problem Statement

• Improving MDD Process
  • Large, Complex, ‘Hard-Coded’ Transformations
  • Often restricted to a 3-Step Approach

• Difficult to Write, Adapt, Reuse Transformations
Problem Statement

• Improving MDD Process
  • Large, Complex, ‘Hard-Coded’ Transformations
  • Often restricted to a 3-Step Approach

Application PIM

J2SE-RMI PSM

.NET PSM

Java Code

C# Code

VB Code

• Difficult to Write, Adapt, Reuse Transformations
• Explicit Stereotyping & Tagging of Model Elements
Problem Statement

- **Improving MDD Process**
  - Large, Complex, ‘Hard-Coded’ Transformations
  - Often restricted to a 3-Step Approach

- **Difficult to Write, Adapt, Reuse Transformations**
- **Explicit Stereotyping & Tagging of Model Elements**
- **Lack of Traceability**
Problem Statement

- Improving MDD Process
  - Large, Complex, ‘Hard-Coded’ Transformations
  - Often restricted to a 3-Step Approach

- Difficult to Write, Adapt, Reuse Transformations
- Explicit Stereotyping & Tagging of Model Elements
- Lack of Traceability
- Non-Functional Properties often only realized at Code Level
Goals

- Development of a Transformation Chain
  - Fine-Grained Reusable Transformations
  - Gradually introducing Non-Functional Aspects
  - Transformation Modeling instead of Transformation Coding
Goals

- Development of a Transformation Chain
  - Fine-Grained Reusable Transformations
  - Gradually introducing Non-Functional Aspects
  - Transformation Modeling instead of Transformation Coding

- Transformation Traceability
  - End2End Traceability from PIM to code
  - Generic and Specific Traceability Information
Transformation Chain

- Gradually Introducing, Refining & Optimizing Application Properties
  [Vanhooff et al, ECMDA-CMT06 & SAMOS06]
Transformation Chain

- Gradually Introducing, Refining & Optimizing Application Properties
  [Vanhooff et al, ECMDA-CMT06 & SAMOS06]

Application (Functional)
Transformation Chain

- Gradually Introducing, Refining & Optimizing Application Properties
  [Vanhooff et al, ECMDA-CMT06 & SAMOS06]
Transformation Chain

- Gradually Introducing, Refining & Optimizing Application Properties
  [Vanhooff et al, ECMDA-CMT06 & SAMOS06]
Transformation Chain

- Gradually Introducing, Refining & Optimizing Application Properties
  [Vanhooff et al, ECMDA-CMT06 & SAMOS06]
Transformation Chain

- Gradually Introducing, Refining & Optimizing Application Properties
  [Vanhooff et al, ECMDA-CMT06 & SAMOS06]
Transformation Chain

- Gradually Introducing, Refining & Optimizing Application Properties
  [Vanhooff et al, ECMDA-CMT06 & SAMOS06]
Transformation Chain

- Gradually Introducing, Refining & Optimizing Application Properties
  [Vanhooff et al, ECMDA-CMT06 & SAMOS06]

Application (Functional) → Application with Persistency → Application with Logging

Application for Java → Application with MI removed

Java Code for Application

TF1 → TF2 → TF3 → TFn → TFn-1
Generic Upsilon Transformation (GUT)
Generic Upsilon Transformation (GUT)
Generic Upsilon Transformation (GUT)

- Model Transformation as a UML Diagram
Generic Upsilon Transformation (GUT)

- Model Transformation as a UML Diagram
- Mapping of Source Model on Transformation
Generic Upsilon Transformation (GUT)

- Model Transformation as a UML Diagram
- Mapping of Source Model on Transformation
- Use a Generic Transformation Engine
Generic Upsilon Transformation (GUT)

• 3 Input Models
  • 2 Base Models
    • Application & Transformation Model
    • SW & HW Model
    • Functional Model & Non-Functional Aspect
  • Mapping Model (using GUT Profile)
Generic Upsilon Transformation (GUT)

- 3 Input Models
  - 2 Base Models
    - Application & Transformation Model
    - SW & HW Model
    - Functional Model & Non-Functional Aspect
  - Mapping Model (using GUT Profile)
- Several Mapping Strategies available
  - Link / Filter / Merge / Inheritance Merge / Copy / Move / Rename / Remove
    - Elements not involved are Preserved
Generic Upsilon Transformation (GUT)

- 3 Input Models
  - 2 Base Models
    - Application & Transformation Model
    - SW & HW Model
    - Functional Model & Non-Functional Aspect
  - Mapping Model (using GUT Profile)
- Several Mapping Strategies available
  - Link / Filter / Merge / Inheritance Merge / Copy / Move / Rename / Remove
    - Elements not involved are Preserved
  - Approach applied in ITEA Project on MDD for Embedded System Development
    - http://www.martes-itea.org
Transformation Traceability

• Traceability is Important to
  • Track Requirements
  • Keep System Parts Consistent
  • Propagate Changes

• Traces contain useful Information
Transformation Traceability

• Traceability is Important to
  • Track Requirements
  • Keep System Parts Consistent
  • Propagate Changes

• Traces contain useful Information

• Automatic Model Transformations
  • Can produce Trace Models ‘for free’
Transformation Traceability

- How to exploit Traceability Information?
  - Find out ‘Hidden’ Relations via Indirect Traces
  - Transformation is unaware of the Exact Traces
  - Use Tagged Traceability

[Vanhooff et al., ECMDA07 Traceability Workshop]
Transformation Traceability

- How to exploit Traceability Information?
  - Find out ‘Hidden’ Relations via Indirect Traces
  - Transformation is unaware of the Exact Traces
  - Use Tagged Traceability

[Vanhooff et al., ECMDA07 Traceability Workshop]
III. Research on software composition @ DistriNet (related to WP3)

Eddy Truyen, Bert Lagaisse, Kris Verlaenen, Nico Janssens, Sam Michiels, Frans Sanen, Tom Govaerts, Yves Vandewoude, Wouter Joosen, Frank Piessens, Yolande Berbers
Helicopter overview
Helicopter overview

- GOAL of our software composition research

*Improving software adaptability and software reuse by improved separation of concerns, higher-level composition support, and dynamic configuration*
Helicopter overview

- GOAL of our software composition research
  
  Improving software adaptability and software reuse by improved separation of concerns, higher-level composition support, and dynamic configuration

- Application domains
  
  - E-Health, Mobility, Security, E-Government, Digital Media
Helicopter overview

• GOAL of our software composition research
  Improving software adaptability and software reuse by improved separation of concerns, higher-level composition support, and dynamic configuration

• Application domains
  • E-Health, Mobility, Security, E-Government, Digital Media

• Distributed Software Systems
  • Typically .NET / J2EE middleware environments
  • Service-Oriented Architectures
  • Open and Reflective systems
TECHNOLOGIES
TECHNOLOGIES

• Dynamic reconfiguration
  • Component-based software systems
  • Safe reconfiguration
TECHNOLOGIES

• Dynamic reconfiguration
  • Component-based software systems
  • Safe reconfiguration

• Aspect-oriented middleware
  • Improved separation of concerns
  • Dynamic (re-)composition of components and aspects
  • Safety
TECHNOLOGIES

- Dynamic reconfiguration
  - Component-based software systems
  - Safe reconfiguration
- Aspect-oriented middleware
  - Improved separation of concerns
  - Dynamic (re-)composition of components and aspects
  - Safety
- Higher-level composition support
  - Policies
  - Business Processes / Service Orchestration
  - Feature interaction models

Y. Vandewoude, Dynamically updating component-oriented systems, Ph.D., 2007
DYNAMIC RECONFIGURATION

- Component-connector decomposition


Y. Vandewoude, Dynamically updating component-oriented systems, Ph.D., 2007
DYNAMIC RECONFIGURATION

- Component-connector decomposition
- Example reconfiguration Scenarios
  - injection of protocol boosters in case of congestion
  - injection of compression in case of packet loss


Y. Vandewoude, Dynamically updating component-oriented systems, Ph.D., 2007
DYNAMIC RECONFIGURATION

- Component-connector decomposition
- Example reconfiguration Scenarios
  - injection of protocol boosters in case of congestion
  - injection of compression in case of packet loss
- Safe Reconfiguration
  - Structural integrity and global state consistency


Y. Vandewoude, Dynamically updating component-oriented systems, Ph.D., 2007
DYNAMIC RECONFIGURATION

• Component-connector decomposition
• Example reconfiguration Scenarios
  • injection of protocol boosters in case of congestion
  • injection of compression in case of packet loss
• Safe Reconfiguration
  • Structural integrity and global state consistency
• Example systems
  • NeCoMan/DIPS: Dynamic reconfiguration middleware layer for network services
  • DRACO: Context-aware reconfiguration of embedded software
ASPECT-ORIENTED MIDDLEWARE

- Middleware needs to support complex compositions
  - Between different components in the network
  - Between components and local services

G. Kiczales et al, Aspect-oriented programming, ECOOP’97
ASPECT-ORIENTED MIDDLEWARE

- Middleware needs to support complex compositions
  - Between different components in the network
  - Between components and local services

- Problem: Tangled composition
  - Decreases reusability,
  - Higher complexity
  => More powerful composition mechanisms needed

G. Kiczales et al, Aspect-oriented programming, ECOOP’97
Aspect-oriented programming

Authentication

Aspect: Authenticate(...)
Before: Execution( *.Main(..) )

Authorization

Aspect: Authorize(...)
Before: Execution( *.set*(..) )

Components/classes

Aspect-oriented programming

An aspect …
“whenever condition X, do Y”

- Defines behaviour Y (advice)
- Defines composition logic (pointcut)
  - binds that behaviour to the rest of the system.
  - Specifies a set of joinpoints

Aspect-oriented programming

- Authentication
  - Aspect: Authenticate(…)
  - Before: Execution( *Main(… ) )

- Authorization
  - Aspect: Authorize(…)
  - Before: Execution( *set*(.. ) )

- An aspect …
  "whenever condition X, do Y"
  - Defines behaviour Y (advice)
  - Defines composition logic (pointcut)
    - binds that behaviour to the rest of the system.
    - Specifies a set of joinpoints

Aspect-oriented middleware

Mainstream programming language

Aspect-oriented composition specification

AO middleware

Dynamic composition of aspects and components


B. Lagaisse et al., True and Transparent Distributed Composition of Aspect-Components, Middleware 2006
DyMAC: True distributed aspects

- **Example: Remote pointcut expressions**
  - Evaluate on **kind** and **context** of joinpoints
  - **Kind:** call and execution of method invocations between components
  - **Contextual properties:**

  - **Caller (sending component)**
    - Name, Application
  - **Dependency**
    - Deployment infrastructure:
      - Host, hostgroup, framework, appdomain
  - **Callee (Receiving Component)**
    - Name, Application
    - Provided interface
      - Deployment infrastructure:
        - Host, hostgroup, framework, appdomain
ASPECT-BASED RECONFIGURATION

E. Truyen et al., Run-Time and Atomic Weaving of Distributed Aspects, Transactions on Aspect-Oriented Software Development II, 2006
N. Janssens et al., Adding Dynamic Reconfiguration Support to JBoss AOP, Workshop on Middleware-Application Interaction (MAI), 2007
ASPECT-BASED RECONFIGURATION

- Lasagne: Concurrent customization
  - Client-specific activation of features
  - Atomic weaving to guarantee behavioral integrity

N. Janssens et al., *Adding Dynamic Reconfiguration Support to JBoss AOP*, Workshop on Middleware-Application Interaction (MAI), 2007
ASPECT-BASED RECONFIGURATION

- **Lasagne:** Concurrent customization
  - Client-specific activation of features
  - Atomic weaving to guarantee behavioral integrity

- **NeCoMan/AOP:** Making JBoss AOP safe
  - JBoss AOP = popular AO framework in industry
    - Support for dynamic aspect weaving but …
  - NeCoMan ported on top of JBoss/AOP …

E. Truyen et al., *Run-Time and Atomic Weaving of Distributed Aspects, Transactions on Aspect-Oriented Software Development II*, 2006

N. Janssens et al., *Adding Dynamic Reconfiguration Support to JBoss AOP, Workshop on Middleware-Application Interaction (MAI)*, 2007
HIGHER-LEVEL COMPOSITION SUPPORT
HIGHER-LEVEL COMPOSITION SUPPORT

- CIA: Management of concern interactions in middleware [Sanen, DAIS 2007]
HIGHER-LEVEL COMPOSITION SUPPORT

- **CIA:** Management of concern interactions in middleware [Sanen, DAIS 2007]
- **ACI:** Aspect Control Interface for evolvable application-specific security policies [Verhanneman, Distr. Computing 2006]
HIGHER-LEVEL COMPOSITION SUPPORT

- CIA: Management of concern interactions in middleware [Sanen, DAIS 2007]
- Simplified policy specification in multiple domains [Verlaenen, IM 2007]
HIGHER-LEVEL COMPOSITION SUPPORT

- CIA: Management of concern interactions in middleware [Sanen, DAIS 2007]
- Simplified policy specification in multiple domains [Verlaenen, IM 2007]
- Policy interactions [Verlaenen, IM 2007] [Verlaenen, Policy 2007]
HIGHER-LEVEL COMPOSITION SUPPORT

- CIA: Management of concern interactions in middleware [Sanen, DAIS 2007]
- Simplified policy specification in multiple domains [Verlaenen, IM 2007]
- Policy interactions [Verlaenen, IM 2007] [Verlaenen, Policy 2007]
- Padus: Aspect support for Service Orchestration [Braem, BPM 2006]
IV. RESEARCH ON VERIFICATION (related to WP6)

Frank Piessens, Wouter Joosen, Pierre Verbaeten, Bart Jacobs, Lieven Desmet, Jan Smans, Dries Vanoverberghe, Pieter Philippaerts
Helicopter overview

- **GOAL** of our verification research:
  - *Improving software quality and security by providing high assurance techniques for dealing with implementation-level vulnerabilities and bugs*

- **TECHNOLOGIES**:
  - **Static verification**: classic Hoare-logic based program verification tuned for specific concerns such as: sandboxing, concurrency, data dependencies
  - **Run time verification**: program monitoring and run time verification of compliance with security policies
Static verification

• Research on verification technology
  • Starting point is the Spec# - ESC/Java line of verifiers
  • Contributions:
    • Sound verification of concurrent programs
    • Better support for specifying and verifying frame conditions
    • Better support for data abstraction in specifications

• Research on applications
  • Motto: Verification as an improved type system
  • Contributions:
    • Verifying absence of race conditions and deadlocks
    • Verifying absence of broken data dependencies
    • Verifying absence of Security Exceptions
Run time verification

- Research on verification technology
  - Starting point is the technique of *Inlined Reference Monitoring*
  - Contributions:
    - Providing support for monitor-aware development
    - Compiling policy languages to run time monitors

- Research on applications
  - Run time monitoring of compliance with device policies of .NET Compact Framework mobile applications
Research Sample: data dependencies

- Web applications
  - Process sequences of user requests
    - Interactive, non-deterministic applications
  - Maintain server-side state to support the notion of **sessions**
  - Maintaining the consistency of that state is hard in the presence of:
    - Naïve users using back-buttons, bookmarking intermediate URL’s,…
    - Malicious users messing with URL’s

- The solution discussed is part of the PhD Thesis of Lieven Desmet
  - Joint work with Wouter Joosen, Pierre Verbaeten
Example web application

- E-commerce site bundled with the J2EE 1.4 tutorial
- Reactive client/server interaction
Desired composition property

- **No broken data dependencies on the shared repository**
  - A shared data item is only read after being written on the shared repository

- For each read interaction, the data item present on the shared repository is of the type expected by the read operation
Solution overview

- Checking specification – implementation compliance
- Application-specific protocol verification
- Run-time protocol enforcement
Solution overview

- Application implementation
- Application specification
- Checking specification – implementation compliance
- Application-specific protocol verification
- Run-time protocol enforcement
Solution overview

Application implementation

Application specification

Checking specification – implementation compliance

Application-specific protocol verification

Run-time protocol enforcement
Solution overview

- Application implementation
- Application specification

- Deployment information
- Intended client/server protocol

- Checking specification – implementation compliance
- Application-specific protocol verification
- Run-time protocol enforcement

Input artifact
Generated artifact
Solution overview

Application implementation

Application specification

Checking specification – implementation compliance

Deployment information

Application-specific protocol verification

Intended client/server protocol

Run-time protocol enforcement

Online web traffic

Input artifact

Generated artifact
Solution overview

- Application implementation
- Application specification
- Deployment information
- Intended client/server protocol
- Online web traffic
- Checking specification – implementation compliance
- Application-specific protocol verification
- Run-time protocol enforcement

Input artifact
Generated artifact
Experimental results

- **Annotation overhead:**
  - At most 4 lines per component

- **Verification performance:**
  - Static verification took at most 4 minutes per component

- **Run-time overhead:**
  - **Experiment:**
    - sequence of 1000 visitors
    - on average 6 requests per session
    - 2% of the users applied forceful browsing
  - Measured run-time overhead of 1.3%
Conclusion

- We are able to guarantee the desired composition properties in a web application
  - With minimal formal specification
  - Using existing verification tools
  - In a reasonable amount of time

- Proposed solution
  - Applicable to real-life applications
  - Scalable to larger applications (if the complexity of the individual components and the protocol remains equivalent)
Research Sample: concurrency in C#

- Multithreaded programs in Java or C# are hard to get right
  - Data races: two threads accessing the same memory location at the same time, and at least one of the accesses is a write
  - Race conditions on composite data structures
  - Deadlocks

- Moreover, testing for concurrency bugs is hard
  - Because of the non-deterministic nature of these bugs

- The solution discussed here is part of the PhD Thesis of Bart Jacobs
  - Joint work with Wolfram Schulte, Rustan Leino, Jan Smans
The Problem

class Range {
    int a, b;
    invariant a <= b;

    Range(int a, int b)
        requires a <= b;
        { this.a = a; this.b = b; }
}
The Problem

class Range {
    int a, b;
    invariant a <= b;
    Range(int a, int b) {
        requires a <= b;
        { this.a = a; this.b = b; }
    }
    Range r = new Range(2, 5);
    Thread 1
    int a1 = r.a;
    Thread 2
    int b2 = r.b;
    r.b = a1;
    r.a = b2;
}
The Problem

class Range {
    int a, b;
    invariant a <= b;

    Range(int a, int b) {
        requires a <= b;
        { this.a = a; this.b = b; }
    }
}

Range r = new Range(2, 5);

Thread 1
int a1 = r.a;
range b = a1;

Thread 2
int b2 = r.b;
range a = b2;

Each thread’s program, when run separately, maintains the invariant
The Problem

class Range {
    int a, b;
    invariant a <= b;

    Range(int a, int b) {
        requires a <= b;
        { this.a = a; this.b = b; }
    }

    Range r = new Range(2, 5);

    Thread 1
    int a1 = r.a;
    r.b = a1;
    r.a = b2;

    Thread 2
    int b2 = r.b;

    Each thread’s program, when run separately, maintains the invariant

    But in a concurrent setting, the invariant is not maintained

    r.a == 5 && r.b == 2
Overview of the solution

• A programming regime (or *programming model*)
  • that prevents data races
  • and deadlocks,
  • and enables local reasoning in the presence of object aliasing.

• An annotation syntax and verification approach
  • that enables modular static verification of compliance with the programming model.
Small example

class Counter {
    int count;
}

class Session implements Runnable {
    Counter counter;
    public void run()
    {
        synchronized (counter) {
            counter.count++;
        }
    }
}

Counter counter = new Counter();
Session session1 = new Session();
session1.counter = counter;
new Thread(session1).start();
Session session2 = new Session();
session2.counter = counter;
new Thread(session2).start();
Small example

class Counter {
    int count;
}

class Session implements Runnable {
    shared Counter counter;
    public void run()
        requires this accessible and this unshared;
    {
        synchronized (counter) {
            counter.count++;
        }
    }
}

Counter counter = new Counter();
share counter;
Session session1 = new Session();
session1.counter = counter;
new Thread(session1).start();
Session session2 = new Session();
session2.counter = counter;
new Thread(session2).start();
Conclusion

- Application-driven research on application of formal verification methods to software quality and security
- Formal soundness proofs for the key ideas …
- … and working implementations that scale to full Java or C#
- Key Papers for research samples:
Collaborations

- **WP 3:**
  - From requirements to design,
    - Collaboration with FUNDP
    - Collaboration with UCL (Axel)
  - Composition technology
    - Collaboration with VUB (related to SBO AspectLab)

- **WP 2:** Collaboration with UA

- **WP 1 and 6:** looking for opportunities

- **Additional:**
  - Interest to follow up on WP 5, 7 (Yolande)
Concluding remarks

- Effective kick off of WP3 scheduled after the summer.
- DistriNet budget increased from 400 to 500 kEuro
  - Tx Yolande:-)
- Open position (faculty member) @DistriNet,
  - Domain: secure software
  - Deadline: October 07
- June 5, 2007: PhD Marko Van Dooren – 17h!
  - Abstractions for Improving, Creating and Reusing Object-Oriented Programming Languages
Thank you!