Enhancing Student Understanding of Formal Method through Prototyping

Andreea Barbu and Fabrice Mourlin

dreea@gmx.net and Fabrice.Mourlin@wanadoo.fr

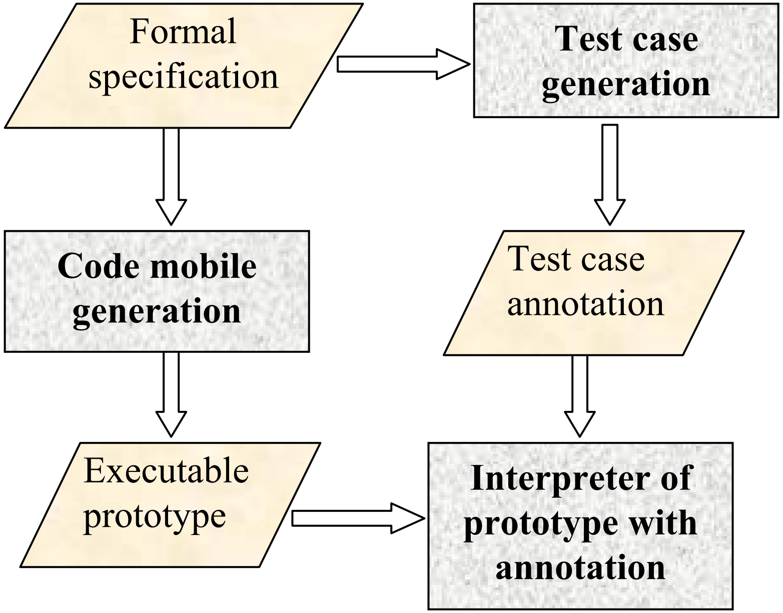
LACL, University Paris XII, France

Plan

- Academic context
- Introduction
- Architecture of our teaching approach
  - Related work
  - Context and tool
- Direct application of formal method
- Case studies
  - Examples
  - SLP
- Conclusion

Academic Context

- Subject : formal methods for mobility
- Objectives:
  - To model mobile agent systems
  - To express properties about MA system
  - To generate and to enrich executable prototype
  - To have unit tests about it



Introduction

- Message to students:
  - Formal methods are necessary in achieving correct software
    - software that can be proven to fulfill its requirements.
  - Formal specifications are unambiguous and analyzable.
    - Building a formal model improves understanding.
    - The modeling of no determinism, communication, mobility, and other features in formal steps, allows design and implementation decisions to be made when most suitable.

Introduction

- Answer from students
  - Formal methods are not suitably supported with development tools,
  - They did not use or observe formal methods in their own industrial experience
    - Formal Methods are not widely used in software development.
  - Formal methods are based on mathematical manipulation and reasoning,
    - They are not confident and skilled in the use of mathematical techniques
    - The previous results of these courses are not well known,

Architecture of our teaching approach

- Mobility description:
  - Formal languages: mobile Unity, HO-Pi Calculus, COOPN$_2$, Ambient calculus or join calculus, M-nets, etc
  - Tools (Related work):
    - mobility WorkBench (MWB for polyadic pi calculus),
    - COOPN plug-in (for Eclipse and NetBeans)
    - Bplug (Eclipse plug in for B specification)
    - Mython (Python tool for M-net specifier),
  - Structure
    - Tool = support of experience exchange
    - = ideal observer of student test

Architecture of our teaching approach

- 2002 first version of our plate form: HOPiTool
  - Formal language : Higher Order Pi Calculus
  - Key concepts :
    - Agent definition,
    - Higher order expression,
    - Exchange of terms between agents,
    - Operational semantics is clearly defined
    - Observations and equivalences are already defined,
    - Sorts and checking are also defined
  - Main constraints :
    - Open plate form for student extensions
    - Network tool for the managing of the agent hosts

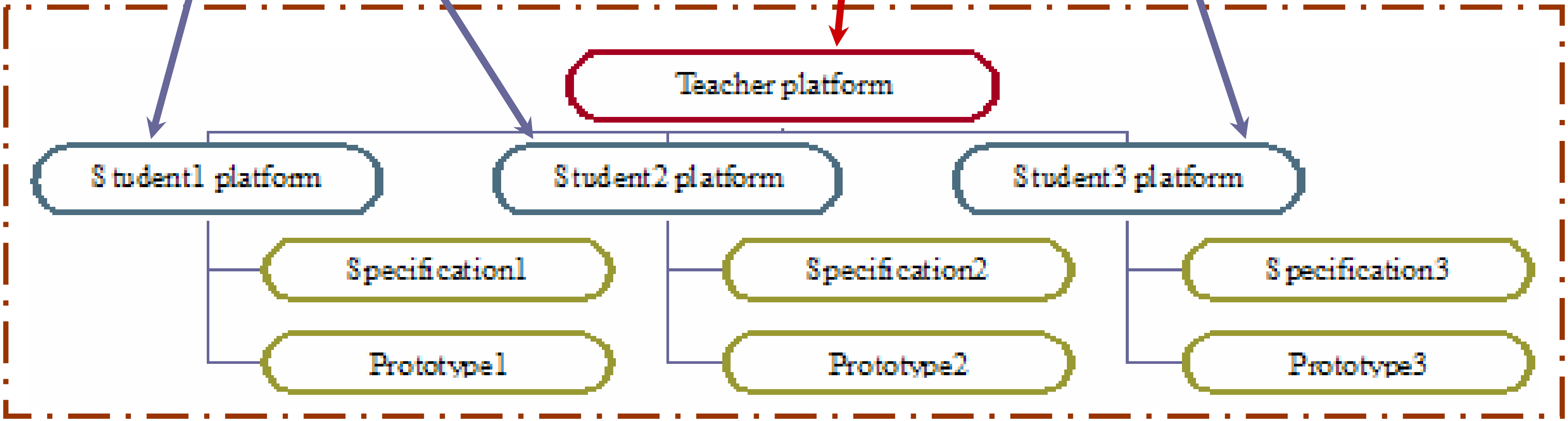
Architecture of our teaching approach

- Context of the course
  - Paris 12 university (computer science department), 35 hours
  - Formal specification to master degree Computer Science students,
  - 30 students
  - 10 Lessons, 1 project per student, a weekly evaluation, 1 exam,

- Structure of the course
  - an explanation of formal feature
    (i.e. deployment of an agent in a graph, etc.)
    - 1,5 hour
  - direct application of previous subjects
    (i.e. the specification of a system based agents which control telnet protocol and forward information)
    - 1,5 hour – 2 hours
  - subject of the evaluation

Direct application of formal method

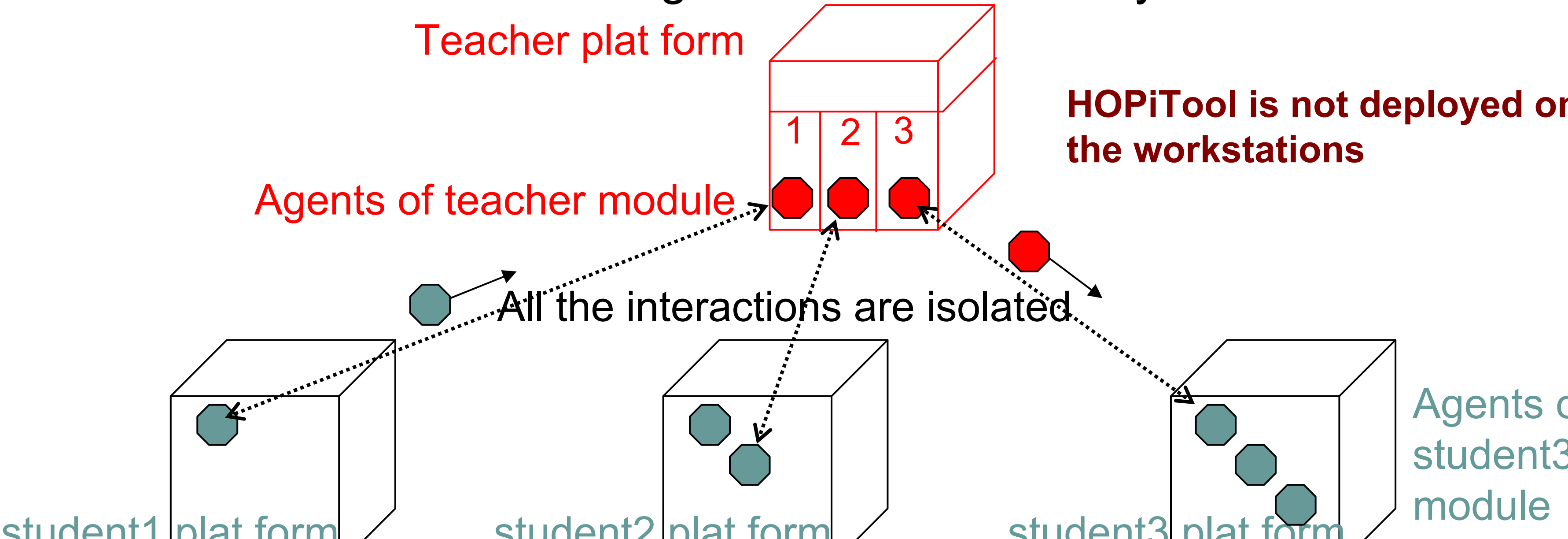
- From specific requirements to specification
  - Student writes its own specifications
  - A student agent checks the results of each students through interactions with a teacher agent,
    - Report is generated for each contribution



HOPiTool is deployed on all the workstations of this teaching network

Direct application of formal method

- From specification to prototype (if previous step is OK)
  - Student generates code through HOPiTool and add some behavioral features (watch point, I / O, etc),
    - Compilation, deployment and configuration
    - Execution of the agents of the student system interaction with the agents of the teacher system.



Teacher plat form

Agents of teacher module

HOPiTool is not deployed or the workstations

All the interactions are isolated

Agents o student3 module

student1 plat form

student2 plat form

student3 plat form

Direct application of formal method

- Observations:
  - For students
    - Interpretation of a scenario
    - Application of observations (Parrow, Sangiorgi)
    - Construction of inference tree for any agents
    - Report about the firing event

- Observations
  - For teacher
    - Timing of the student work,
    - Bug tracking
    - Measure about all the resu of a student group (statistics on difficulties, etc
    - Definition of new metrics : equivalence relations, etc.

Case studies

- Student project examples
  - SLP protocol simulation
    - (Service Location Protocol)
  - Intrusion detection system
    - Login protocol is observed by agent which filters users
  - Mobile computing
    - Pi number calculus with BBP formula
    - Parallel bubble sort
    - Matrix computation

- Teacher deliverable
  - Requirements
  - A part of specification
    - The teacher agent
  - A register for the subscription of the students
    - All time events are saved
  - A teacher module of agents
    - Agents for the case study
    - Agents for student evaluation

Case studies

- Service Location Protocol
  - Subject: agent exportation and local activity
  - Requirement: 5 agents are defined
    - 3 agents are specified by teacher (DA, DA_{Mem}, $IdleDA_{Mem}$)
    - 2 agents have to be specified by student (SA, UA)
  - First evaluation of specifications
  - HOPiTool code generation
    - Java, Jini API
  - Deployment over the network:
    - Lookup service are started
    - Teacher agent module is started
    - Student agent module is started
  - Second evaluation of multi agent module

SLP Case study

- Scenario
  - Set of interactions between SA and DA
    - SA wants to publish a print service and a mail service (for the session) : first request
    - Sa sends both services to DA and receives acknowledge
  - Set of interactions between UA and DA
    - UA looks for a print service : first request
    - UA receives a service from SA and uses it for printing a quiz

$$A(SrvRqst) = (vSrvRply)SrvRply(Service(print, msg), SrvRply).SrvRply(S(name, f)).UA(SrvRq$$

$$(SrvReg, SrvAck) = SrvReg(Service(print, msg).SrvAck.SA(SrvReg, SrvAck)$$

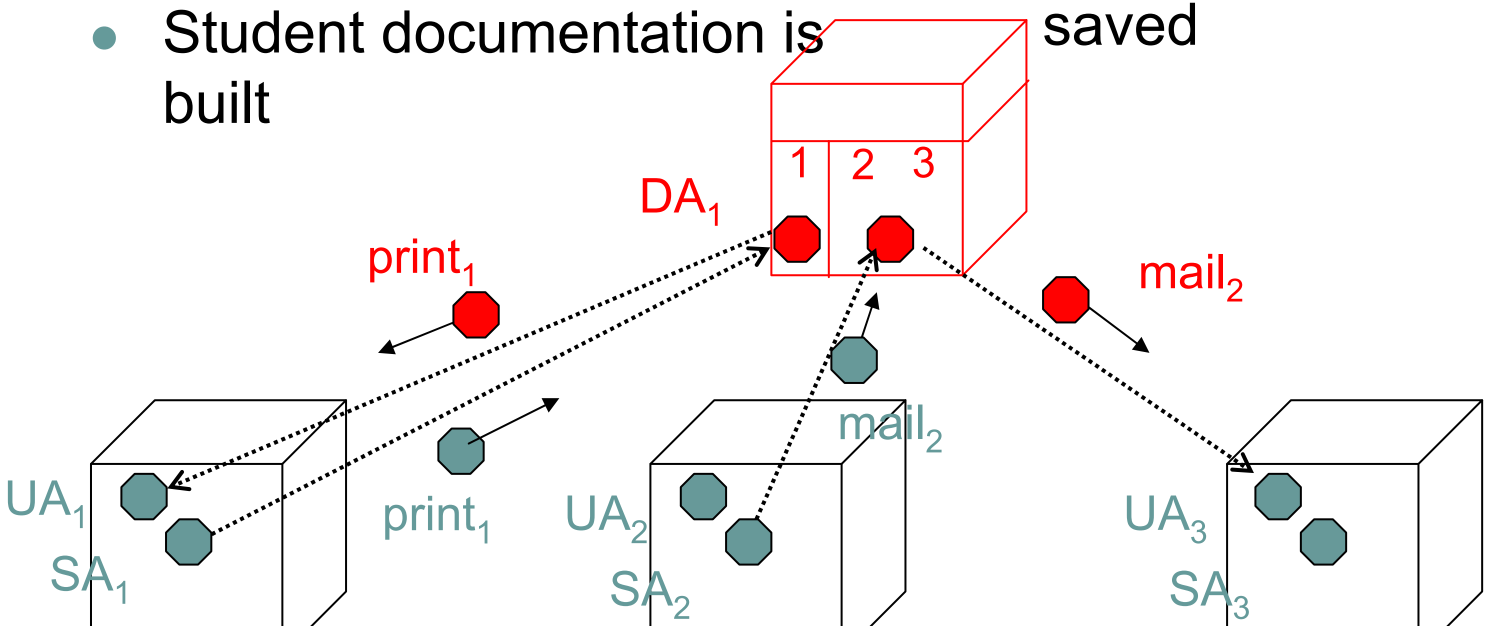
$$A(SrvReg, SrvRqst, SrvAck) = (SrvReg.(S(name, f).input(S(name, f).SrvAck)))$$

$$|(SrvRqst(S(name, f), SrvRply).name(SrvRply)).DA(SrvReg, SrvRqst, SrvA$$

SLP Case study

- From specification
  - Mobile code is generated
  - Unit test cases are defined (JUnit and JDepend)
  - Student documentation is built

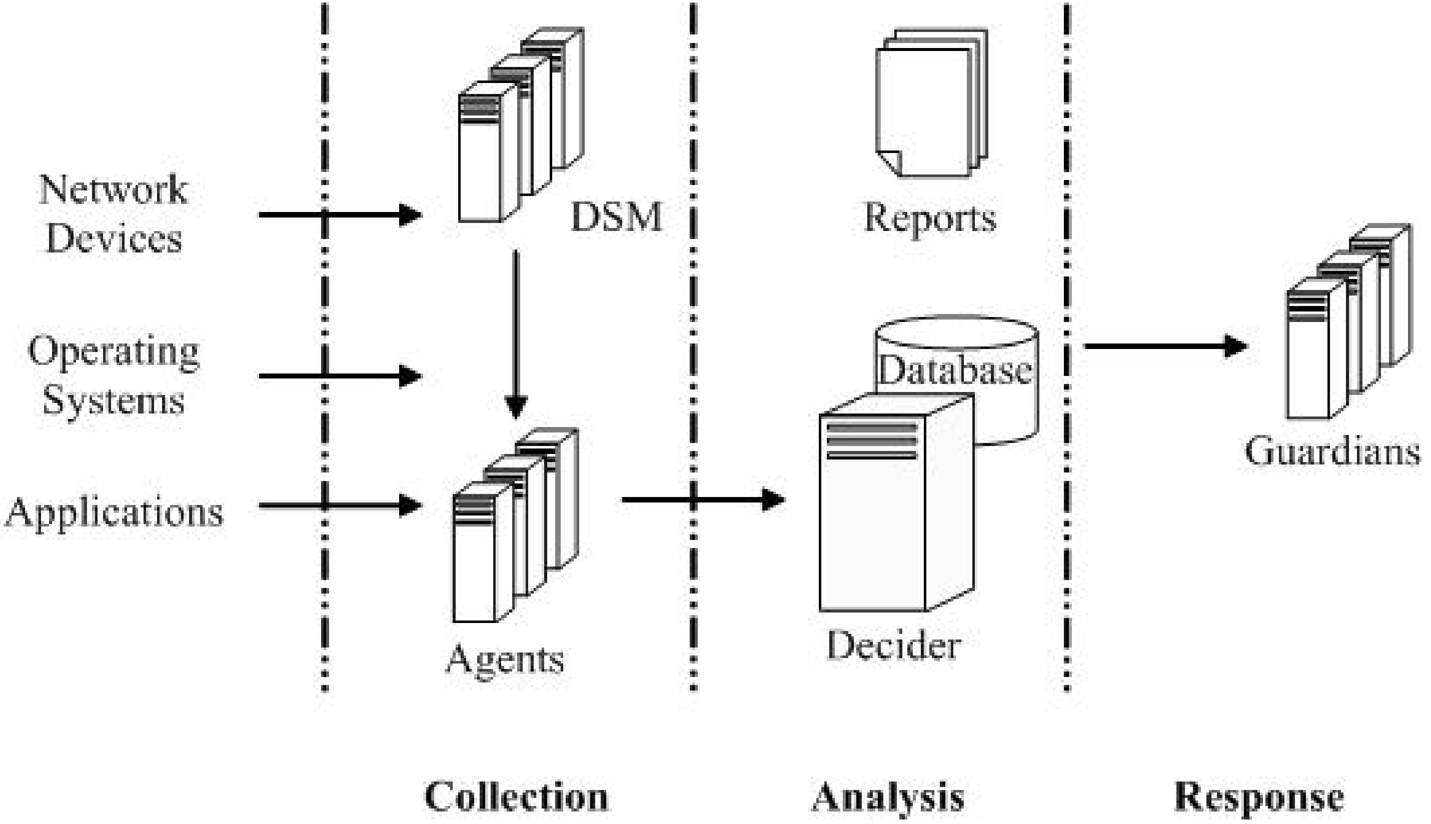
- From student mobile code
  - Services are published into global lookup service of HOPiTool
  - Results of test cases are saved



Conclusion

- Our teaching approach
  - Direct feedback : direct measure about student understanding
  - Same tool is used for direct application and final evaluation
  - Student projects bring new contribution to specification repository
  - Teacher contribution improves HOPiTool
    - new formal observations
    - New features like test cases or replay.
  - Tier-3 : 3 students work on Huntsman project
    - detection and denial of intruder attacks
    - www.tier-3.com

IDS – Architecture



BBP

Pi number formula

$$\pi = \sum_{i=0}^{\infty} \frac{1}{16^i} \left(\boxed{\frac{4}{8i+1}} - \boxed{\frac{2}{8i+4}} - \boxed{\frac{1}{8i+5}} - \boxed{\frac{1}{8i+6}} \right)$$

4 agents: one per contribution

A collector agent picks up each result and computes the value of the iterati

A iterator agent computes le global approximation of all the collector