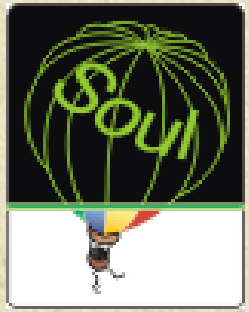
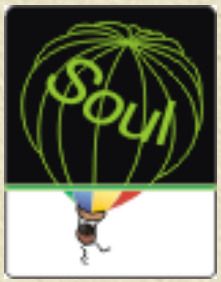


Logic Meta Programming in SOUL

Prof. Dr. Kim Mens
INGI — UCL
Belgium

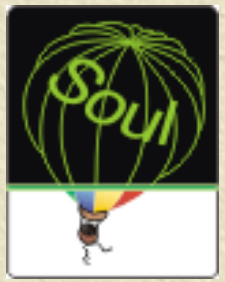
Johan Brichau, Tom Tourwé,
Dr. Tom Mens
PROG — VUB
Belgium

Research Goal

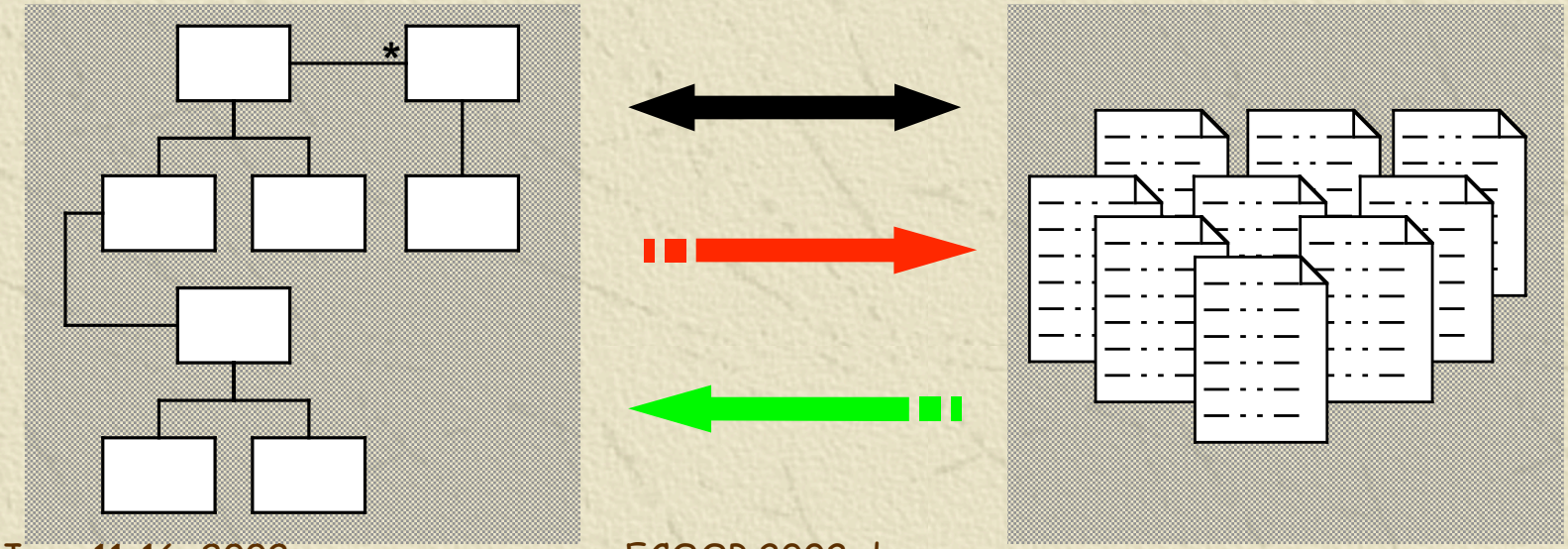


- Build sophisticated tools to support a variety of *software development* activities
  - *co-evolution* among different phases in the software life cycle
    - Code mining, conformance checking, synchronization, code generation
  - advanced *software engineering techniques*:
    - Code optimization, refactoring, change propagation, software metrics, aspect-oriented programming, guiding reuse

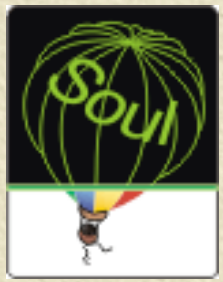
Why Logic Meta Programming?



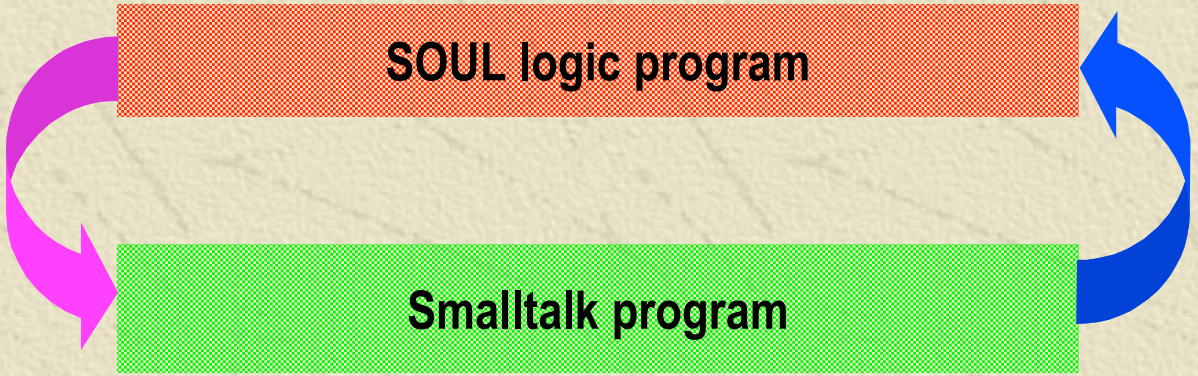
- *A* unifying approach to support such sophisticated tools
  - **Conformance** of design and implementation
  - **Generation** of implementation from design
  - **Extraction** of design from implementation



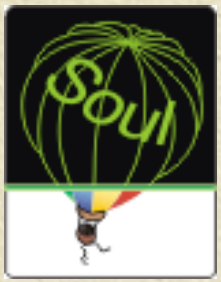
What is Logic Meta Programming?



- Combination of a logic language at the meta level and an object-oriented language at the base level
  - meta-level programs manipulate and reason about the (static) structure of base-level programs

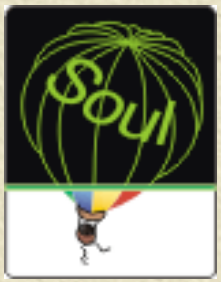


Why **Logic** programming?



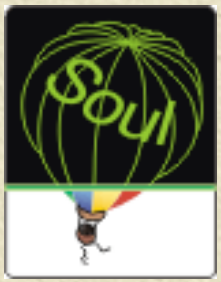
- Logic languages are good at
  - Metaprogramming,
  - (design-) knowledge representation
  - (multi-way) reasoning about knowledge
- Declarative
  - Focuses on "***what***" not on "***how***"
- SOUL
  - a Prolog-like logic programming language
  - Special features for code generation and inspection

Why **Meta** programming?



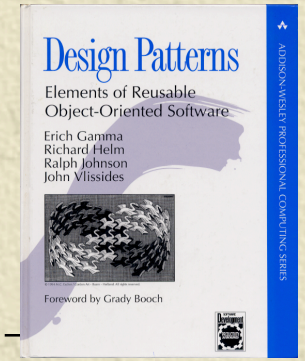
- A specific need to
  - Reason about programs
  - Manipulate programs
- This requires
  - Representation of base-level programs
    - Logic facts & rules
  - Tight integration with development environment
    - Always in sync with implementation
    - Source code generation

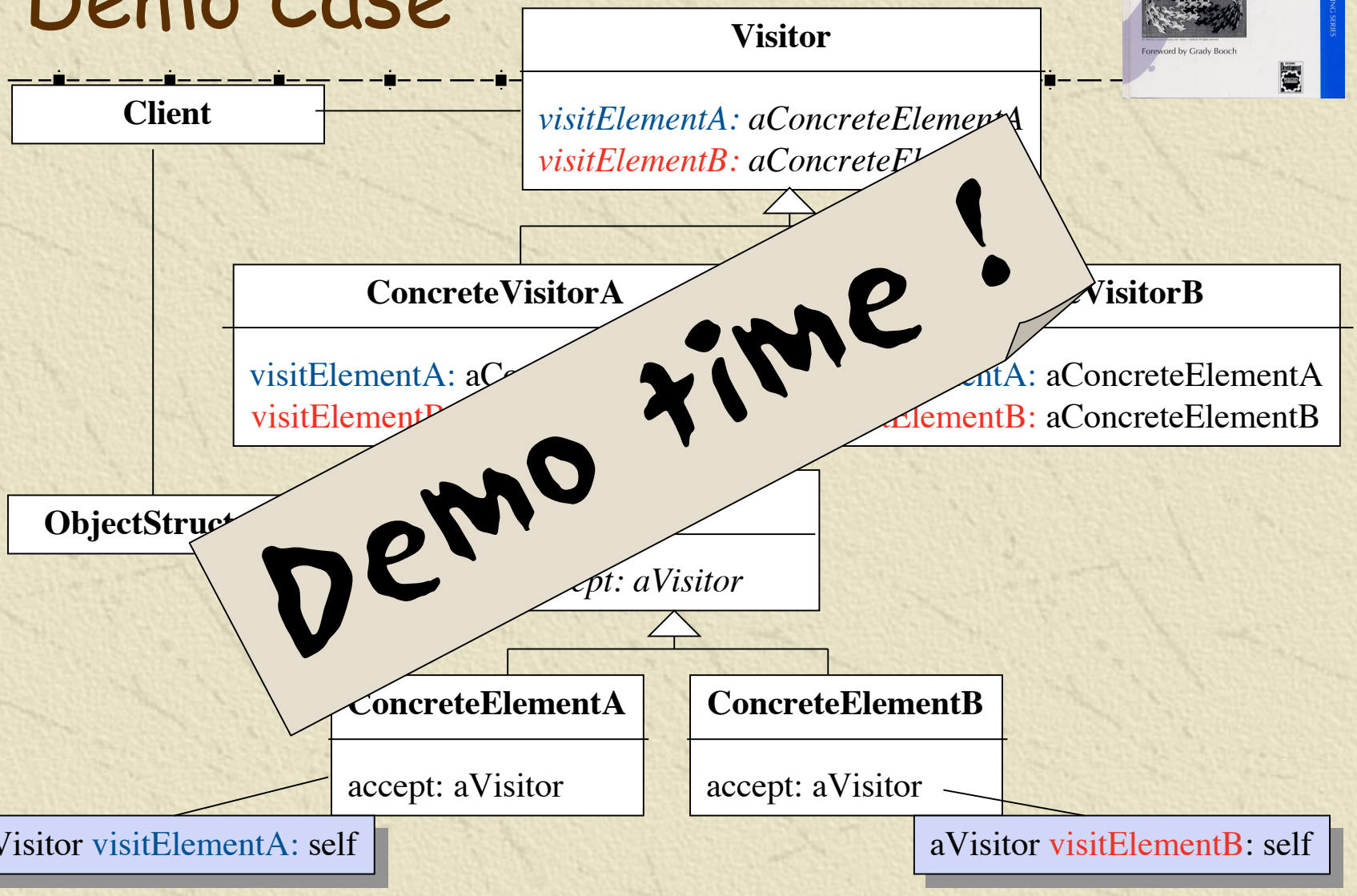
Demo Case



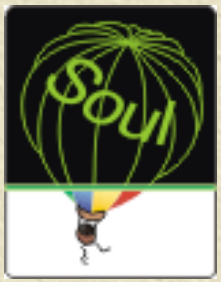
- Advanced tool support for design patterns
  - Generation
    - source code
    - documentation
  - Evolution
    - Keep documentation up to date
    - Detect design inconsistencies
  - Extraction
    - Retrieve documentation from implementation

Demo Case


Design Patterns
Elements of Reusable
Object-Oriented Software
Erich Gamma
Richard Helm
Ralph Johnson
John Vlissides
Foreword by Grady Booch
ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES


Visitor
visitElementA: aConcreteElementA
visitElementB: aConcreteElementB
Client
ConcreteVisitorA
visitElementA: aConcreteElementA
visitElementB: aConcreteElementB
ConcreteVisitorB
visitElementA: aConcreteElementA
visitElementB: aConcreteElementB
ObjectStructure
accept: aVisitor
ConcreteElementA
accept: aVisitor
aVisitor visitElementA: self
ConcreteElementB
accept: aVisitor
aVisitor visitElementB: self
Demo time !

Conclusion



- Logic meta programming is
  - Using a logic metalanguage to reason about and manipulate programs written in an (object-oriented) base language
  - A technique to build state-of-the art software development tools
  - A unifying approach that combines the research of a growing group of researchers
  - A laboratory for conducting our software engineering experiments
- For downloads and more information on SOUL see
  - *http://prog.vub.ac.be/research/DMP*