UCL

Logic Metaprogramming in SOUL

Prof. Kim Mens
INGI — UCL
Belgium

Johan Brichau
PROG — VUB
Belgium

INGI
Département d'ingénierie informatique

Kim.Mens@info.ucl.ac.be,
Johan.Brichau@vub.ac.be

UCL

Why logic metaprogramming?

- Need for more sophisticated tools that support a variety of *software development* activities
  - *co-evolution* among implementation and information in earlier life-cycle phases:
    - Code mining, conformance checking, synchronization, code generation
  - advanced *software engineering techniques*:
    - Code optimization, refactoring, change propagation, software metrics, aspect-oriented programming, guiding reuse
- *Logic metaprogramming* is a unifying approach for building a wide variety of such tools

INGI

Département d'ingénierie informatique

UCL

What is logic metaprogramming?

- A kind of hybrid language symbiosis
- Combines a declarative language at meta-level with an object-oriented base language
  - base-level programs are expressed as logic terms, facts and rules at the meta level
  - meta-level programs can manipulate and reason about the structure of the base-level programs
- *We* use
  - Smalltalk as base language
  - a Prolog-derivative as meta language: SOUL

INGI

Département d'ingénierie informatique

Kim.Mens@info.ucl.ac.be, Johan.Brichau@vub.ac.be

Logic Metaprogramming

- We use a Prolog-like programming language
  - Logic programs are good at
    - metaprogramming, language processing, (multi-way) reasoning about knowledge, unification, backtracking
  - Focuses on "*what*" not on "*how*"
  - Maybe not best choice…but most people know it
    - Other declarative languages: (e.g. Gofer,…)
- Or rather, our Prolog-dialect "SOUL"
  - Slightly different syntax
  - With extra features to reason about and manipulate Smalltalk program structures

Département d'ingénierie informatique

A SOUL Program

SOUL syntax differs only slightly from Prolog syntax
(note the different syntax for variables and for lists too)

```
" List is the concatenation of L1 and L2
  append(?L1,?L2,?List)                    "

append(<>,?Lst,?Lst).

append(<?First|?Rest>,?L2,<?First|?Lst>) if
   append(?Rest,?L2,?Lst).

if append(<1,2,3>,<4,5>,<1,2,3,4,5>).
if append(<1,2,3>,<4,5>,?List).
if append(?L1,<4,5>,<1,2,3,4,5>).
if append(<1,2,3>,?L2,<1,2,3,4,5>).
if append(?L1,?L2,<1,2,3,4,5>).
```

Comment

Fact

Rule

Queries

Département d'ingénierie informatique

UCL

Logic *meta*programming

- We use a logic language…
- …as *meta*language to manipulate and reason about structure of programs written in the base language
- All relevant constructs in the base language are "reified" as rules and facts in the logic language
- SOUL = Smalltalk Open Unification Language
  - uses Smalltalk as base language
  - is a Prolog-dialect
  - featuring a strong symbiosis with the underlying Smalltalk

INGI

Département d'ingénierie informatique

SOUL: symbiosis with Smalltalk

- SOUL works directly on the Smalltalk image
- Smalltalk values and expressions can be used as constants in the logic language
- Logic facts, rules and queries can contain Smalltalk expressions…
- … that may be parameterised with logic variables
- And… SOUL is nicely integrated with the Smalltalk environment (see demo)

Département d'ingénierie informatique

Using Smalltalk values in SOUL

- *Integers*:
  - Shortcut:

```
if factorial([4],?X)
if factorial(4,?X)
```

- *Symbols*:
  - Shortcut:
  - Note: `Symbol` is not a variable but a constant!

```
if write([#Symbol])
if write(Symbol)
```

- *Strings*:
  - Shortcut: no shortcut for stings (yet)

```
if write(['This is a string!'])
```

- *Classes*:

```
if class([Object])
```

- *Other* Smalltalk objects: `[` *some Smalltalk object* `]`

Using Smalltalk expressions as logic *terms*

- "Smalltalk terms" `[` *some Smalltalk expression* `]`
  - "Reify" Smalltalk objects into logic *terms*
  - Can contain any Smalltalk expression
    - not only Smalltalk constants
    - expression should evaluate to an object
    - expression may be parameterized with logic variables which
      - are supposed to be bound upon evaluation of the expression
      - Are substituted by their value before evaluating the expression

- Examples:
  - ST constant:
  - ST expression:
  - Parameterized expr.:

```
if class([Array])
allClasses([Smalltalk allClasses])
plus(?x,?y,[ ?x + ?y ])
```

Département d'ingénierie informatique

Using Smalltalk expressions as logic *clauses*

- Smalltalk clauses [*some Smalltalk expression*]
  - Same syntax and semantics as Smalltalk terms
  - Execute parameterized Smalltalk expressions
  - Difference in usage:
    - Used in the position of logic clauses (instead of logic terms)
    - Should always return true or false after evaluation

- Examples:

```
write(?text) if
    [Transcript show: (?text asString). true].

smallerThan(?x,?y) if
    atom(?x), atom(?y), [?x < ?y].
```

Département d'ingénierie informatique

Using Smalltalk expressions to generate multiple results

- Logic queries can return multiple results
- Smalltalk expressions produce unique answers
- (How) can we use Smalltalk terms to write logic rules that produce multiple results?
- Answer: use `generate/2` predicate
  - First argument is unbound logic variable
  - Second argument is ST term returning a ST collection
  - All elements of the collection are unified one by one with the variable thus producing multiple results
- Example:

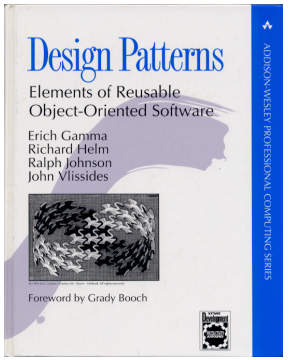
```
class(?C) if
   generate(?C,[SOULExplicitMLI current allClasses])
```

Département d'ingénierie informatique

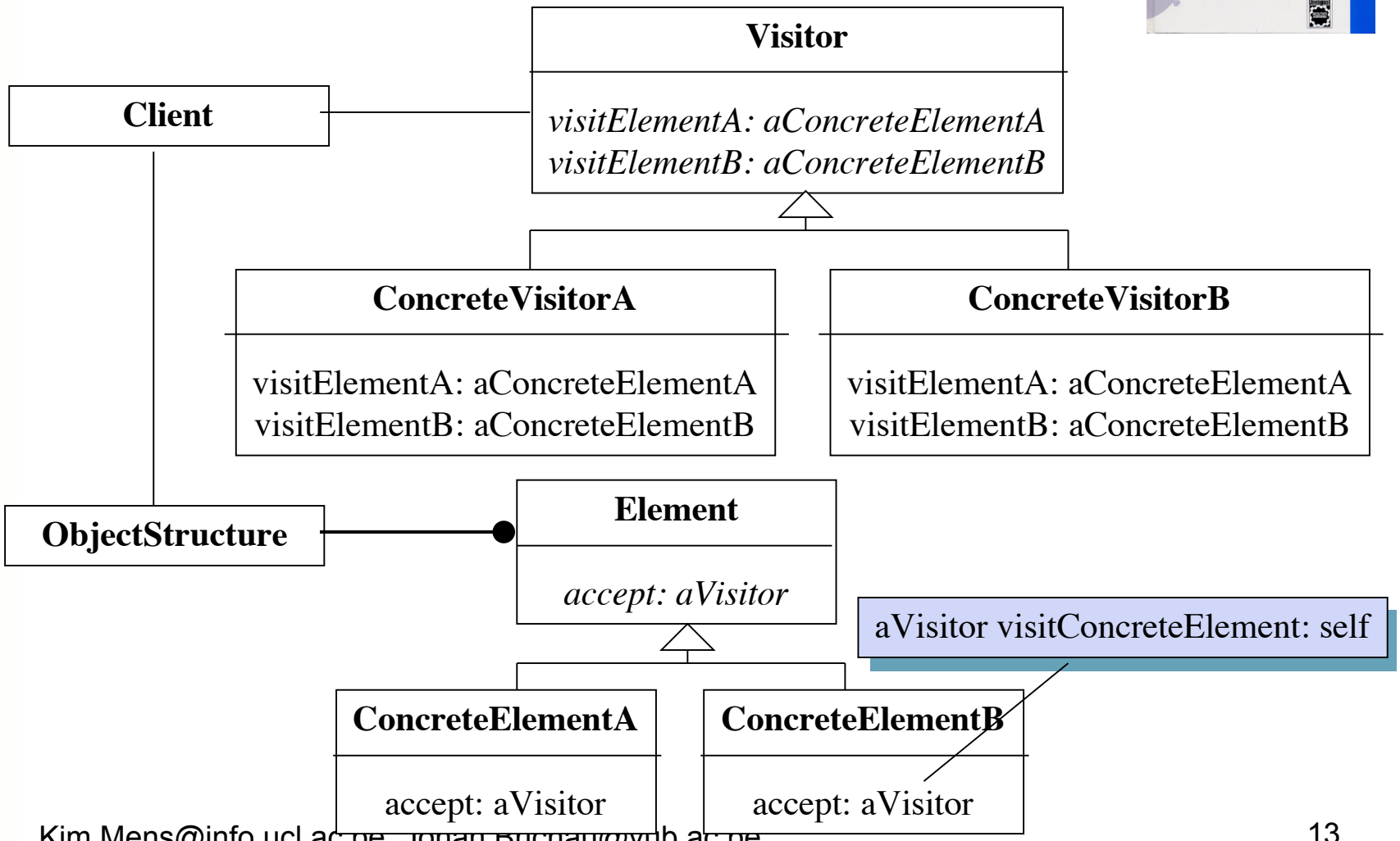
Quasi-quoting

- Quasiquoted Code Term
  - Enclosed between '{' and '}'
  - Can contain logic variables
  - Is **not** executed by SOUL
- Examples
  - {Array at: 1 put: ?x}
  - {boolean ifTrue:[?trueC] ifFalse:[?falseC]}
  - {<html> ?htmlheader ?htmlbody </html>}
- compileMethod(?class,?code) if
  [?class compile: ?code. true]

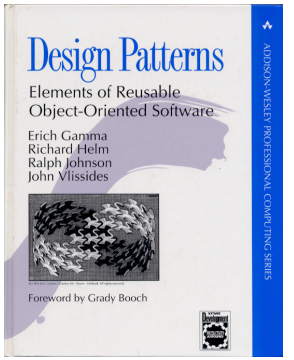
Département d'ingénierie informatique

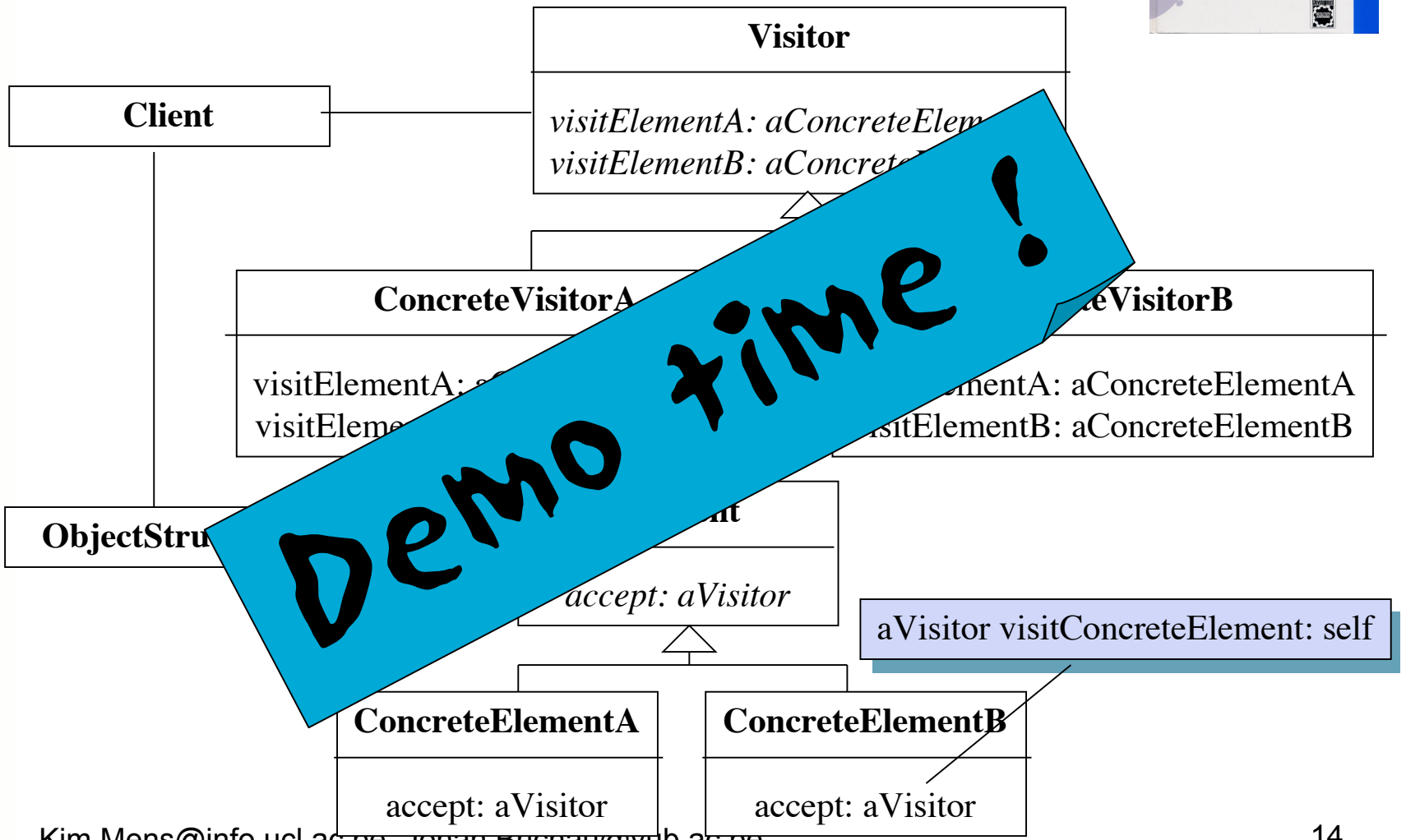


Example: Visitor pattern



Example: Visitor pattern





UCL

Conclusion: LMP

- LMP = using a logic metalanguage to reason about and manipulate programs written in an (object-oriented) base language
- LMP is
  - A unifying approach that combines the research of a growing group of researchers
  - A laboratory for conducting our software engineering experiments
  - A technique to build state-of-the art software development tools

INGI

Département d'ingénierie informatique

Conclusion: SOUL

- Several LMP tools and environments already exist:
  - SOUL, TyRuBa, C2C, QSOUL
  - Most recent tool = SOUL 2.8.x for VW5i.4
- SOUL
  - Is *free : http://prog.vub.ac.be/research/DMP*
  - Is getting quite stable and efficient
  - Is well-integrated with the Smalltalk IDE
  - Has a growing user community
  - Is being documented (automatically…)

Département d'ingénierie informatique

UCL

Conclusion: Usability

- During a 4-day course on DMP, the students successfully solved following assignments:
  - Detect violations of the law of Demeter
  - Verify & detect occurrences of Adapter and Bridge pattern
  - Find all valid/invalid constructor/initializer pairs in ST programs
  - Provide support for SOUL testing conventions
  - Extend the Smalltalk type inferencer
  - Calculate some software metrics
  - Detect code duplication in ST programs
  - Verify & detect occurrences of Decorator and Proxy pattern
- The (Master level) students had no prior knowledge of SOUL nor of Prolog

INGI

Département d'ingénierie informatique

UCL

Future work

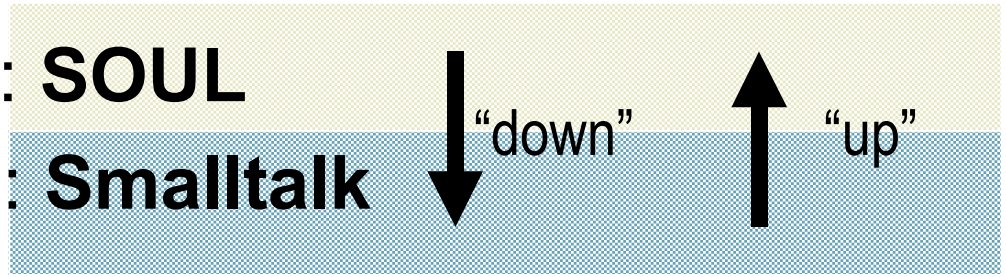
- Other reasoning engines, e.g.
  - Regular expressions
  - Forward chaining
  - Constraint languages
- Other base languages, e.g. Java
- Reasoning about dynamic aspects
- Enhance language symbiosis
  - Make use of SOUL from within Smalltalk more transparent
  - Towards *meta-circularity* and *linguistic symbiosis*

INGI

Département d'ingénierie informatique

Some technicalities: "Up-down" mechanism of SOUL

- Meta-language: **SOUL**
- Base-language: **Smalltalk**



- Symbiosis
  - Smalltalk values can be used in SOUL
    - 'up' of ST-values: explicit wrapper for objects defines the unification on ST-objects.
  - SOUL values can be used in Smalltalk
    - 'down' of 'upped ST-values': ST-value.
    - But: 'down' of SOUL-values: *ongoing research…*

Some technicalities: example of *up-down* mechanism