An Open Unification Language to Express Design Information

Roel Wuyts

FNRS Meeting

May 6th, 1997

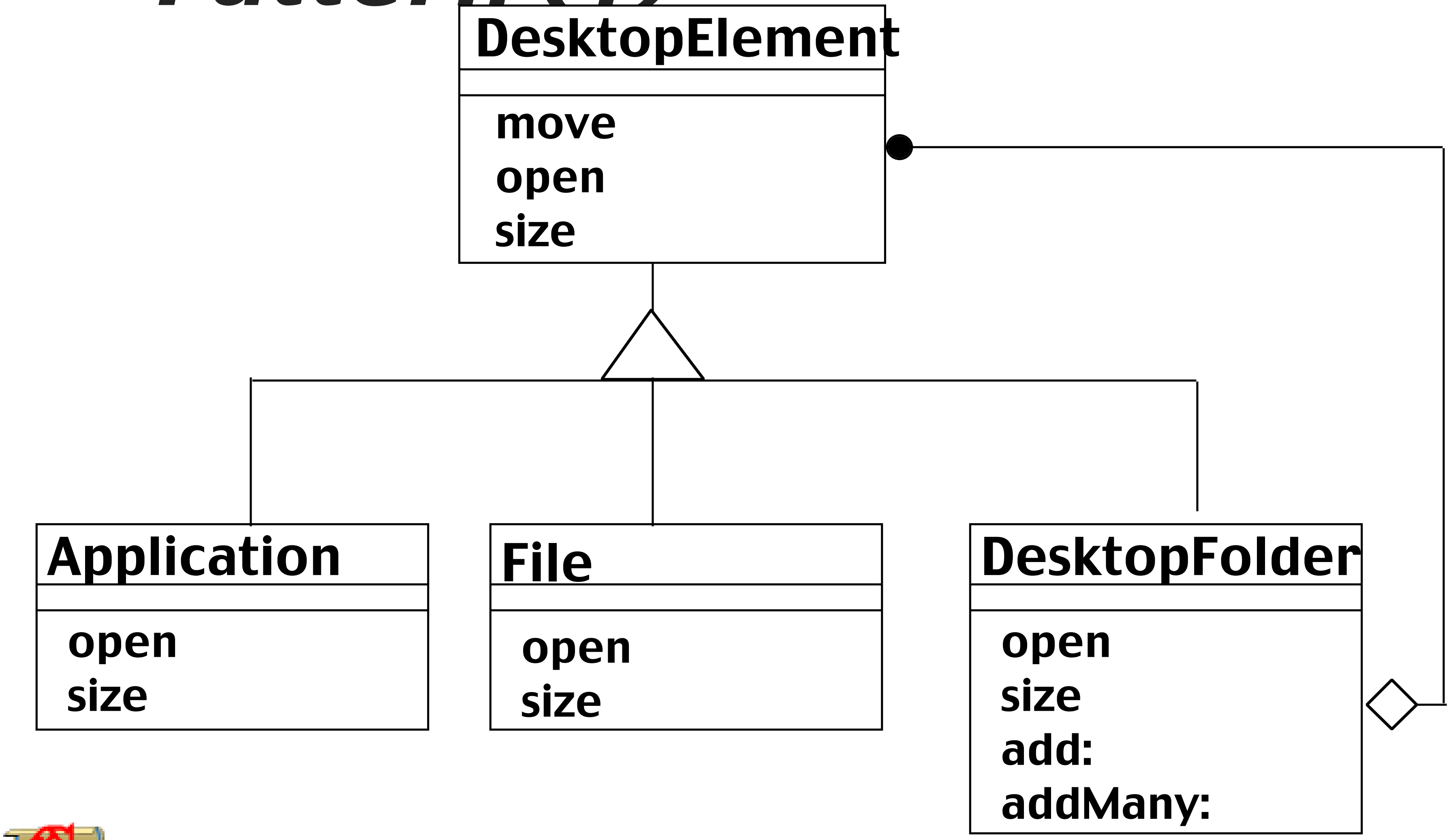
Contents

- Extracting and verifying design information
- Example : composite design pattern
- How to describe design information
- Domains
- An open unification language
- The composite design pattern
- Conclusion

Extraction/Verification of Design

- Formally Expressing Design Information.
- Automatically extracting and verifying.
- Reuse Contracts are just one approach.
- A general support mechanism to extract and verify different kinds of design information is needed.

Example : Composite Pattern (1)



DesktopElement

move
open
size

Application

open
size

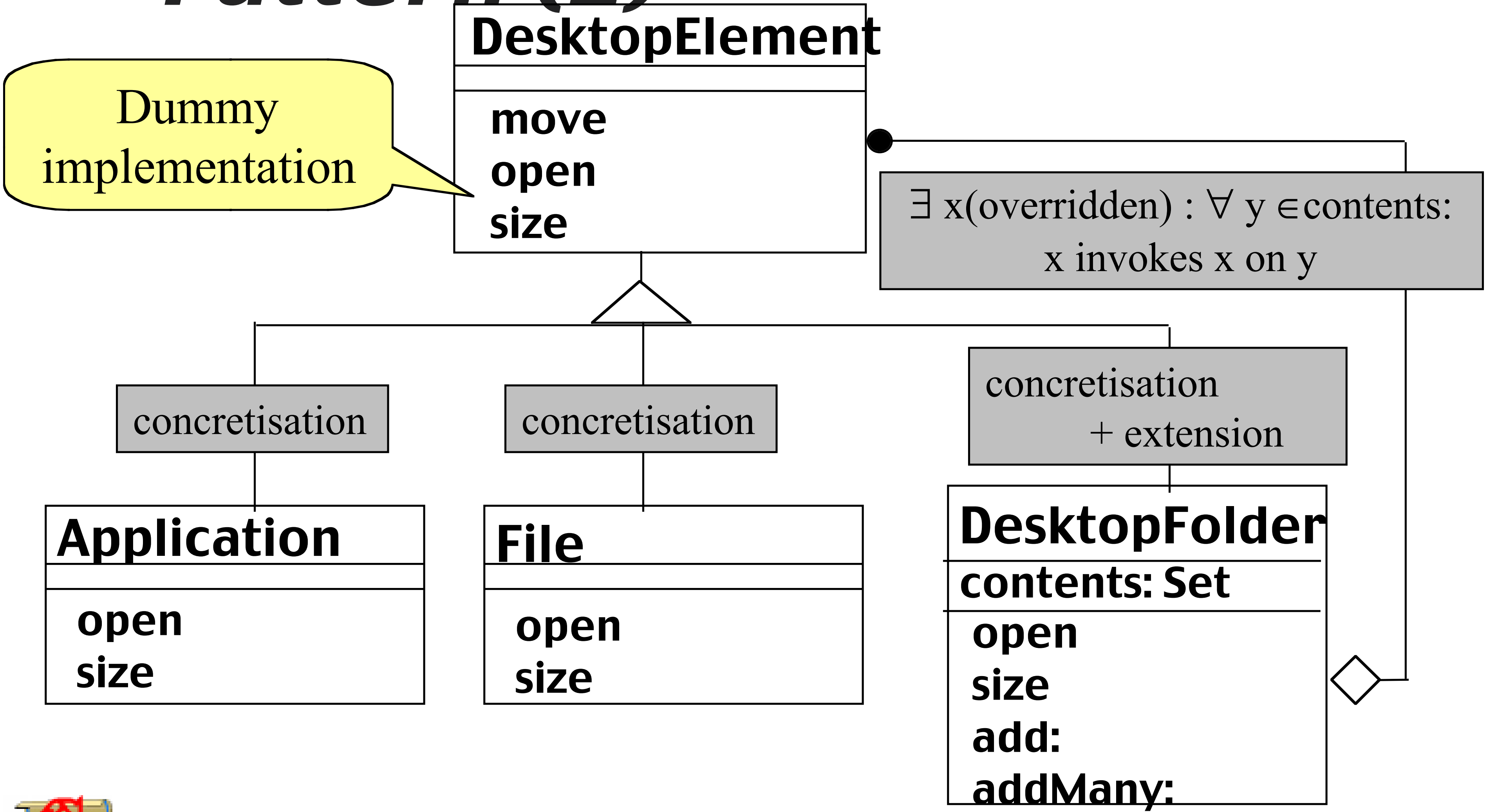
File

open
size

DesktopFolder

open
size
add:
addMany:

Example : Composite Pattern (2)



How to describe Design Information

- declarative (allows intuitive but formal descriptions)
- multi–way (describe relations)
- general (different kinds of design information can be incorporated)

Domains

- Classic PROLOG unifies over strings.
- Design patterns and Reuse Contracts are about classes, methods, instance variables, inheritance, ...
- Therefore, unification over strings is not enough
- Unification over user–definable domains is needed

An Open Unification Language

- To verify and extract design information we use :
  - a declarative language with unification
  - user–definable domains

Most current logic or constraint programming languages have fixed domains.

Our Language

- Prolog–like constructs
- Symbiosis with Smalltalk :
  - can be acessed from within Smalltalk code
  - Smalltalk code is used
    - in atoms
    - to define domains
    - to construct ‘virtual’ rules

Prolog like constructs

Syntax

```
FACT term.
RULE head IF body.
QUERY term.
```

Atoms use Smalltalk code

Examples

FACT abstract([DesktopFolder],[#size]).

RULE abstractClass(?C) IF abstract(?C, ?t).

QUERY abstract(?C,[#add:]).

Domains

Syntax

DOMAIN <*name*(*arg1*, … , *arg n*)> [*definition*].

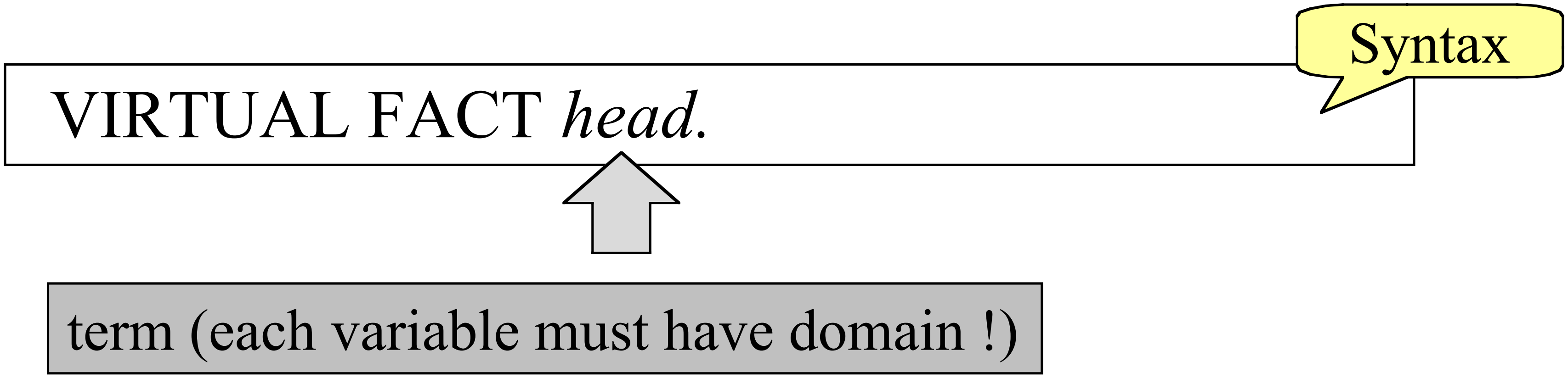
Domains are defined with smalltalk code

Examples

DOMAIN <Classes> [Smalltalk classNames].

DOMAIN <Method(class)> [class selectors].

Virtual Facts



Syntax

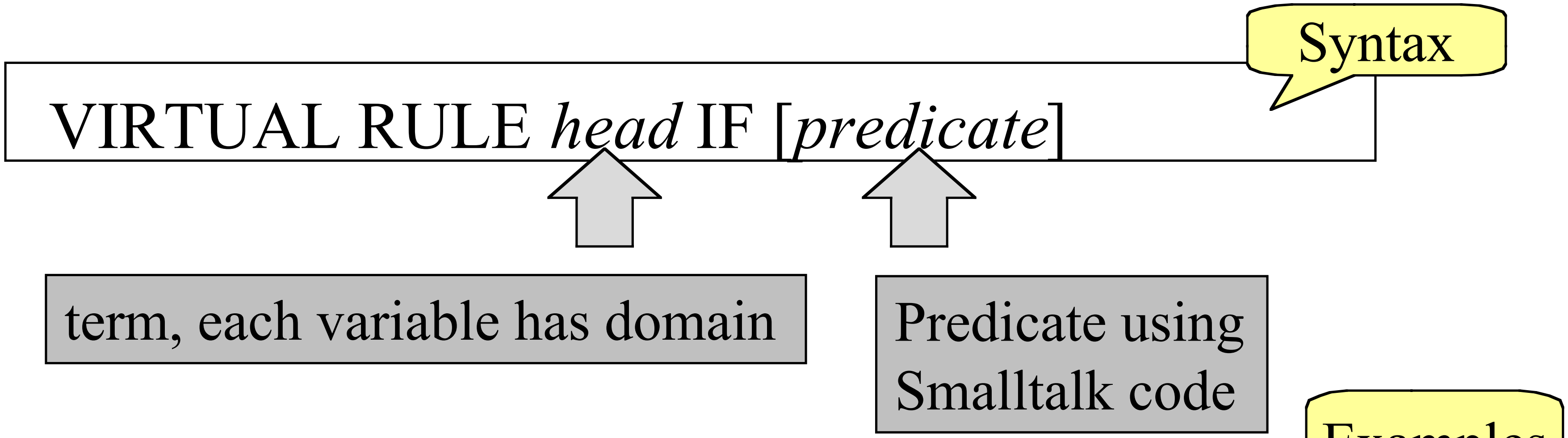
VIRTUAL FACT *head.*

term (each variable must have domain !)

Examples

```
VIRTUAL FACT
  Methods(?C<Classes>,?M<Method(?C)).
```

Virtual Rule



Syntax

VIRTUAL RULE *head* IF [*predicate*]

term, each variable has domain

Predicate using Smalltalk code

Examples

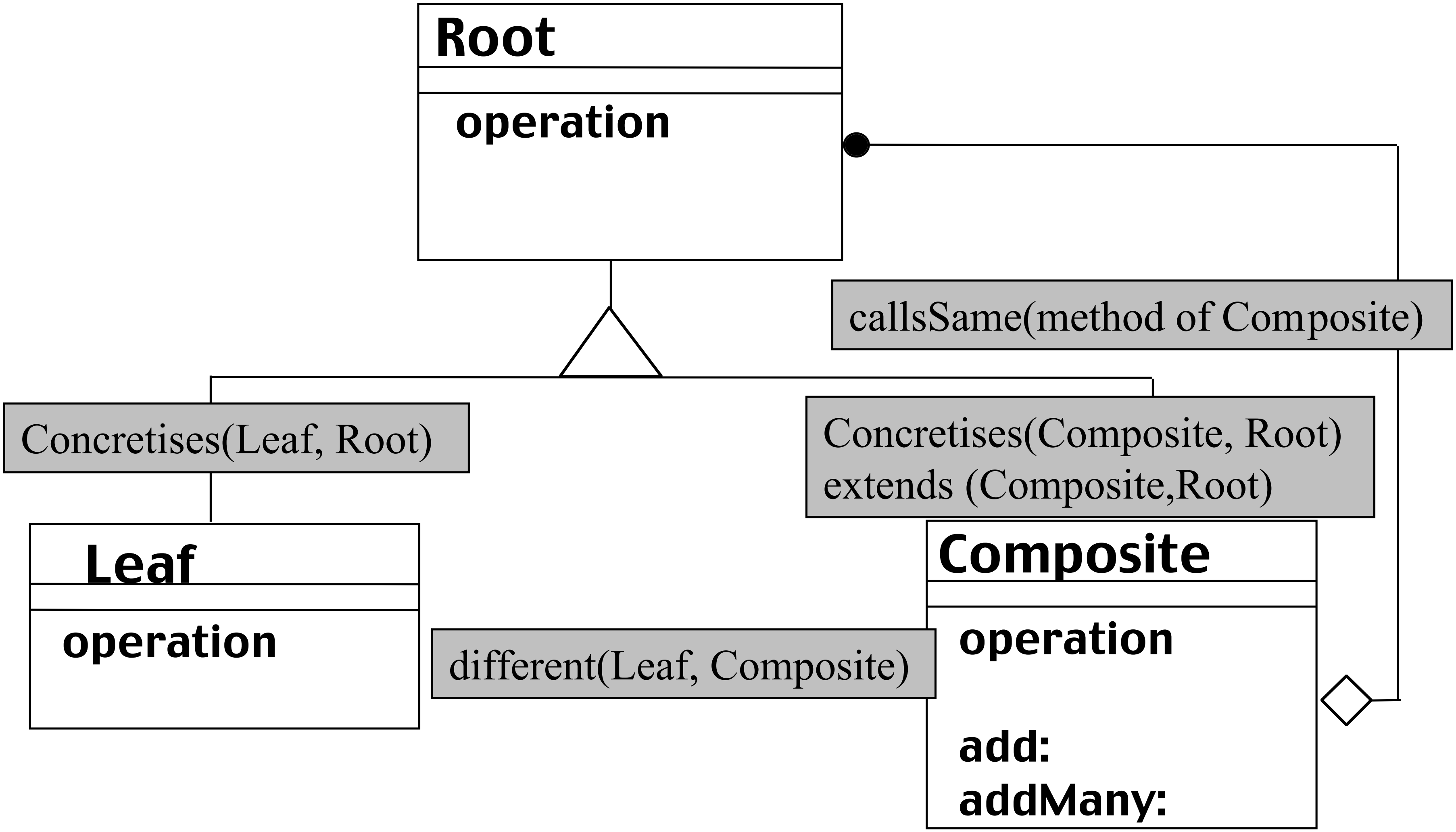
VIRTUAL RULE

super(?S<classes>,?C<classes>) IF [C super = S]

VIRTUAL RULE

abstractMethod(?C<classes>,?M<method(?C)>) IF [C abstractMethods include: M]

Composite Design Pattern



The composite design pattern : Rule

```
RULE composite(?root,
    ?leaf,?composite)
IF  concretises(?leaf,?root),
    concretises(?composite,?root),
    extends(?composite, ?root),
    different(?leaf,?composite),
    methods(?root, ?M),
    callsSame(?composite, ?M).
```

The composite design pattern : some auxiliary rules

- RULE hierarchy(?R,?S) IF super(?S,?R).

  RULE hierarchy(?R,?S) IF
  super(?T,?R),hierarchy(?T,?S).
- VIRTUAL FACT methods(?C<Classes>,?M<Methods(?C)>).
- RULE overrides(?C,?M) IF
  hierarchy(?S,?C),
  methods(?C,?M),methods(?S,?M).

The composite design pattern : extraction and verification

- Extraction

  QUERY composite(?Root, ?Leaf, ?Comp).

- Verification

  QUERY composite([DesktopItem],[File] ,[DesktopFolder]).

  QUERY ([DesktopItem],?L,?C)

Conclusion

- We use an open unification language to express design information
- It can be used to build tools to
  - extract/enforce and verify different kinds of design information, such as Reuse Contracts and Design Patterns
  - detect conflicts
  - do quality assessment

More Information...

e-mail : rwuyts@is1.vub.ac.be

Http://progwww.vub.ac.be/
prog/persons/rwuyts/research.html