Hans Schippers
Tim Molderez
Dirk Janssens

A Graph-based Operational Semantics for Context-oriented Programming

Ansymo
Antwerp Systems and software Modelling

Universiteit Antwerpen

Context

- Observation: Several types of languages aiming to improve separation of concerns: AOP, COP, FOP, role-based programming, …
- Support for these languages is often limited to the language level.

Context

- Goal: Create a VM supporting modularization of crosscutting concerns: delMDSOC
  (delegation-based Multi-Dimensional Separation of Concerns)
  - Hans Schippers et al., OOPSLA 2008
    Delegation-based Semantics for Modularizing Crosscutting Concerns
  - Michael Haupt and Hans Schippers, ECOOP 2007
    A Machine Model for Aspect-oriented Programming

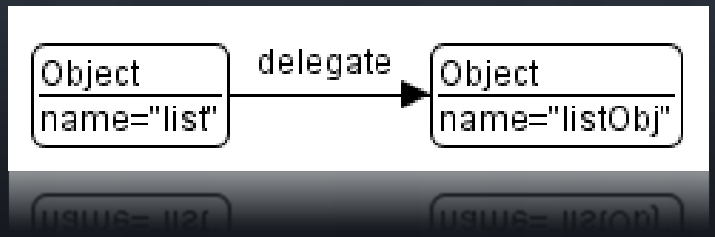
Introduction to delMDSOC

- Express all high-level concepts in terms of:
  - Prototype objects
  - Message passing
  - Delegation
- Each object has a proxy:

  Allows for dynamic (un-)deployment of code, in the form of aspects, layers, …

Dynamic deployment example

A list object and its proxy:



Deploy list caching:

Structural operational semantics

Production rule of a `with`-block:

$$\frac{\begin{array}{c} getMethods(P, L) = M_1 \ldots M_n \\ \forall i : M_i = T_i\ C_i.m_i(T_i'\ x)\{e_i\} \\ \forall k \in [1, n] : \iota_{lp,k} \notin dom(h) \\ h' = \quad h[\iota_{lp,1} \mapsto [\![m_1 : e_1]\!]][Del_{\iota_{lp,1}} \mapsto Del_{h(C_1)}] \\ [Del_{h(C_1)} \mapsto \iota_{lp,1}] \\ \ldots \\ [\iota_{lp,n} \mapsto [\![m_n : e_n]\!]][Del_{\iota_{lp,n}} \mapsto Del_{h(C_n)}] \\ [Del_{h(C_n)} \mapsto \iota_{lp,n}] \\ [Lyr(\iota_{lp,1}) \mapsto L] \ldots [Lyr(\iota_{lp,n}) \mapsto L] \\ P \vdash e, h', s \rightsquigarrow_\delta \iota, h'' \\ h''' = \quad h''[Del_{h(C_n)} \mapsto Del_{\iota_{lp,n}}] \ldots [Del_{h(C_1)} \mapsto Del_{\iota_{lp,1}}] \end{array}}{P \vdash withlayer(L)\{e\}, h, s \rightsquigarrow_\delta \iota, h'''}$$

SOS, some disadvantages

- Implicit representation of program state: hard to see the relations between objects
  - Makes it hard to reason about or experiment with the model
- Lack of tool support: no simulation
  - Simulation helps to understand semantics: Learn by example

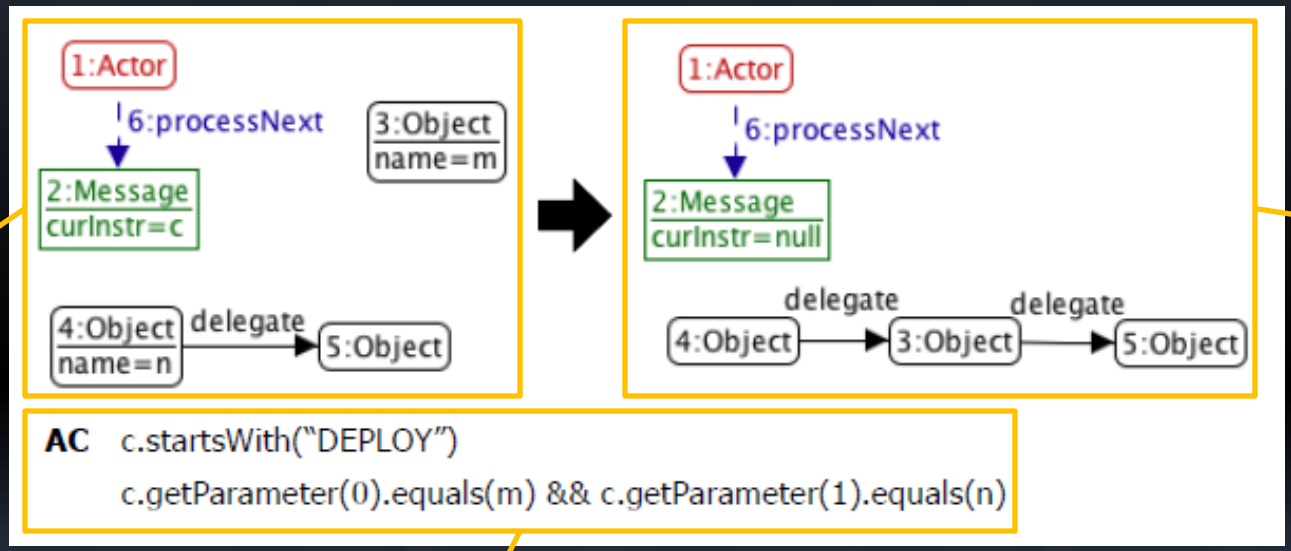
An alternative: Graph rewriting

- Explicit representation of program state
- Tool support:
  AGG was used to model this semantics

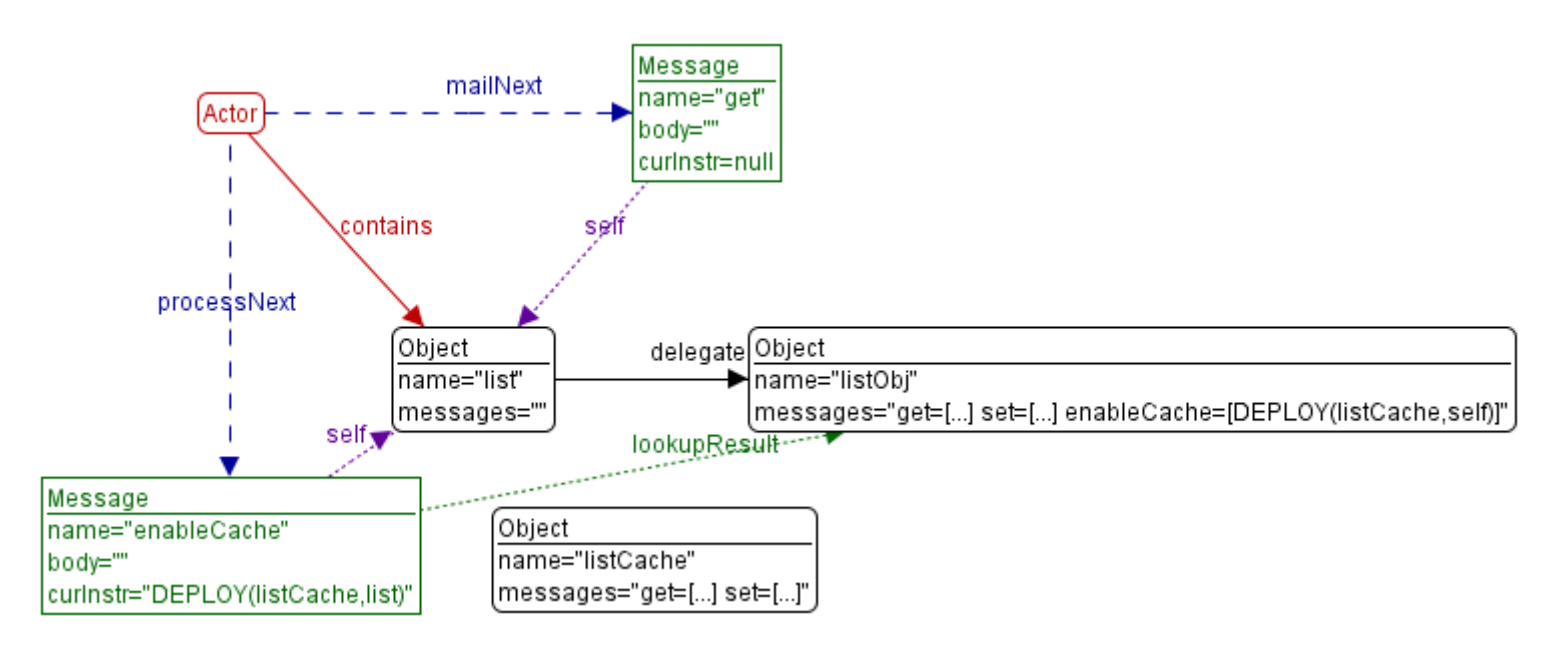
  - Grzegorz Rozenberg, 1997
    Handbook of Graph Grammars and Computing by Graph Transformation
  - Gabriele Taentzer, AGTIVE 2003
    AGG: A Graph Transformation Environment for Modeling
    and Validation of Software

A graph rewrite rule

Rule describing the `DEPLOY` instruction:

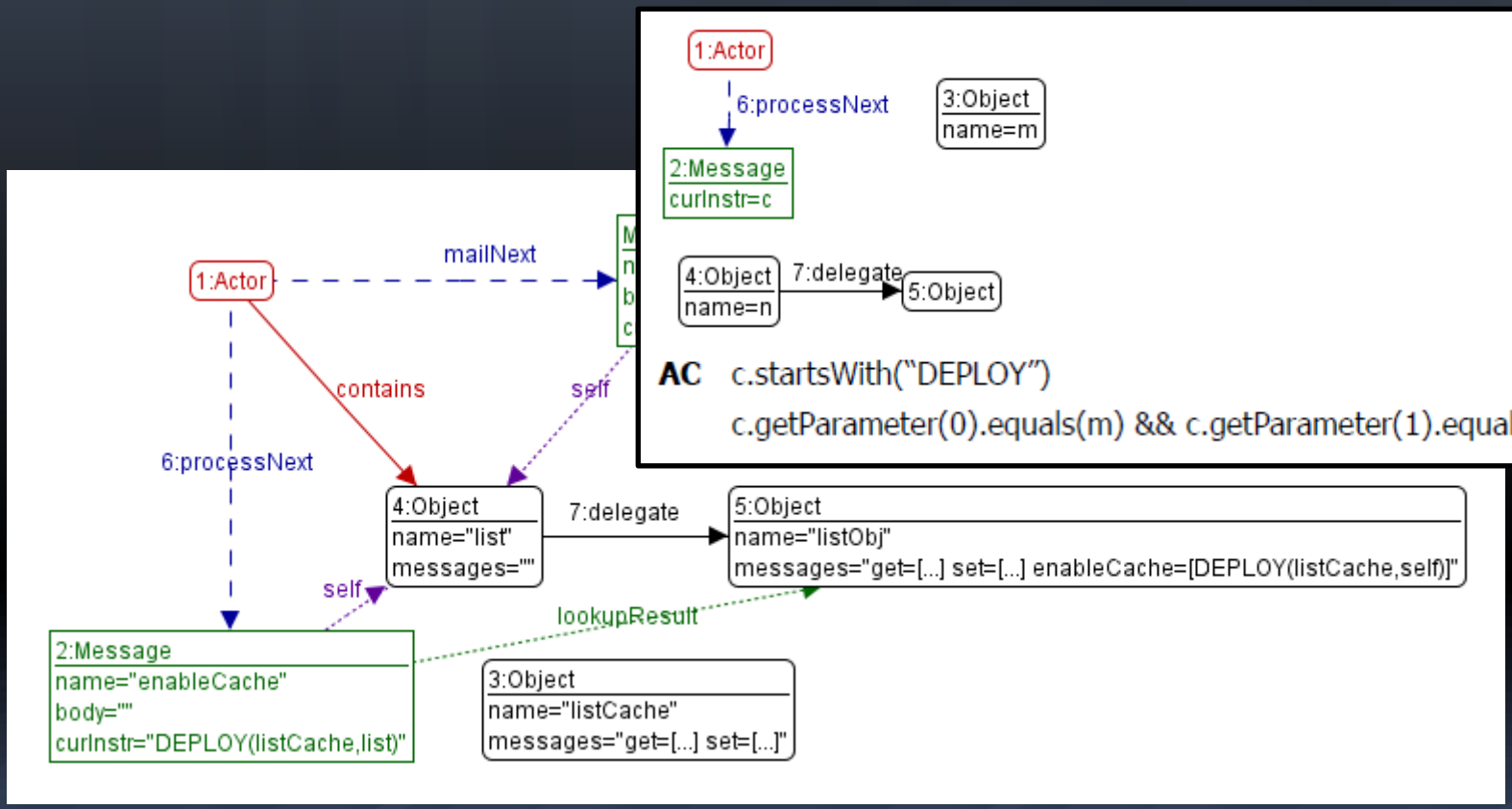


Rule application: before



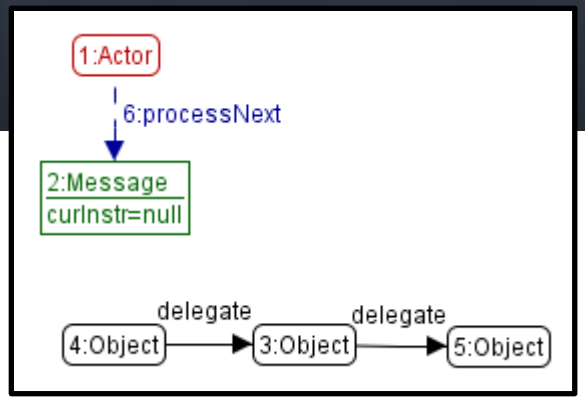
Rule application: before

Rule LHS:



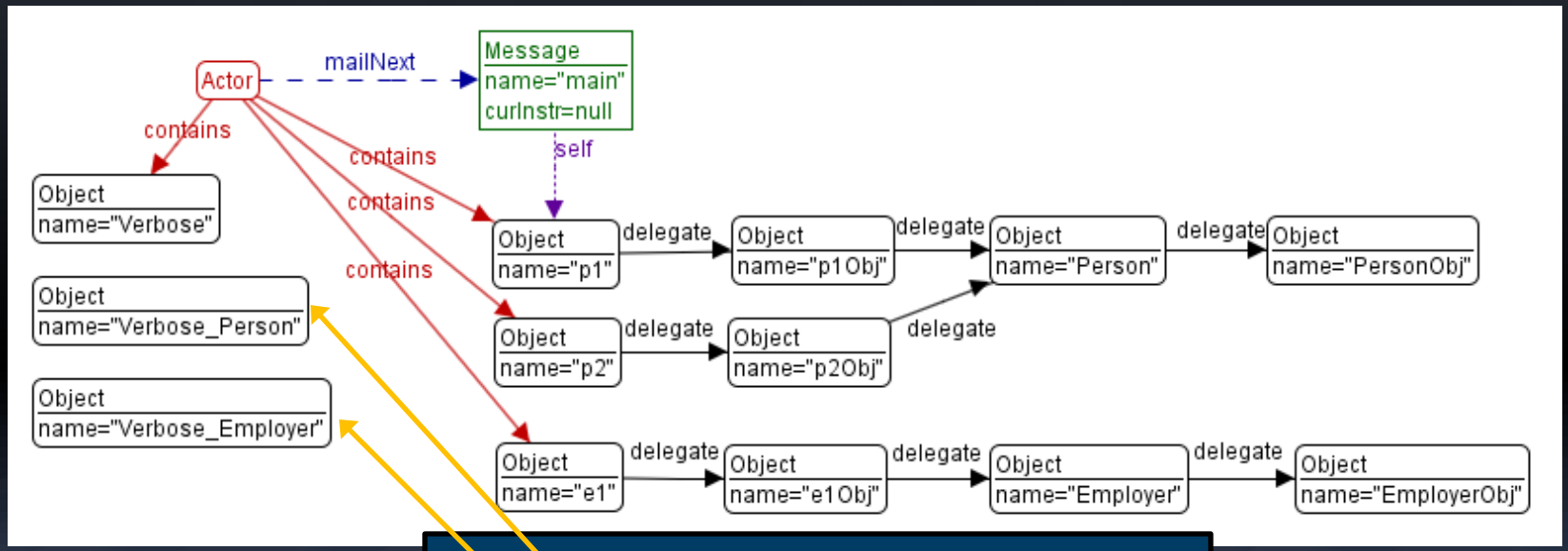
Rule application: after

Rule RHS:



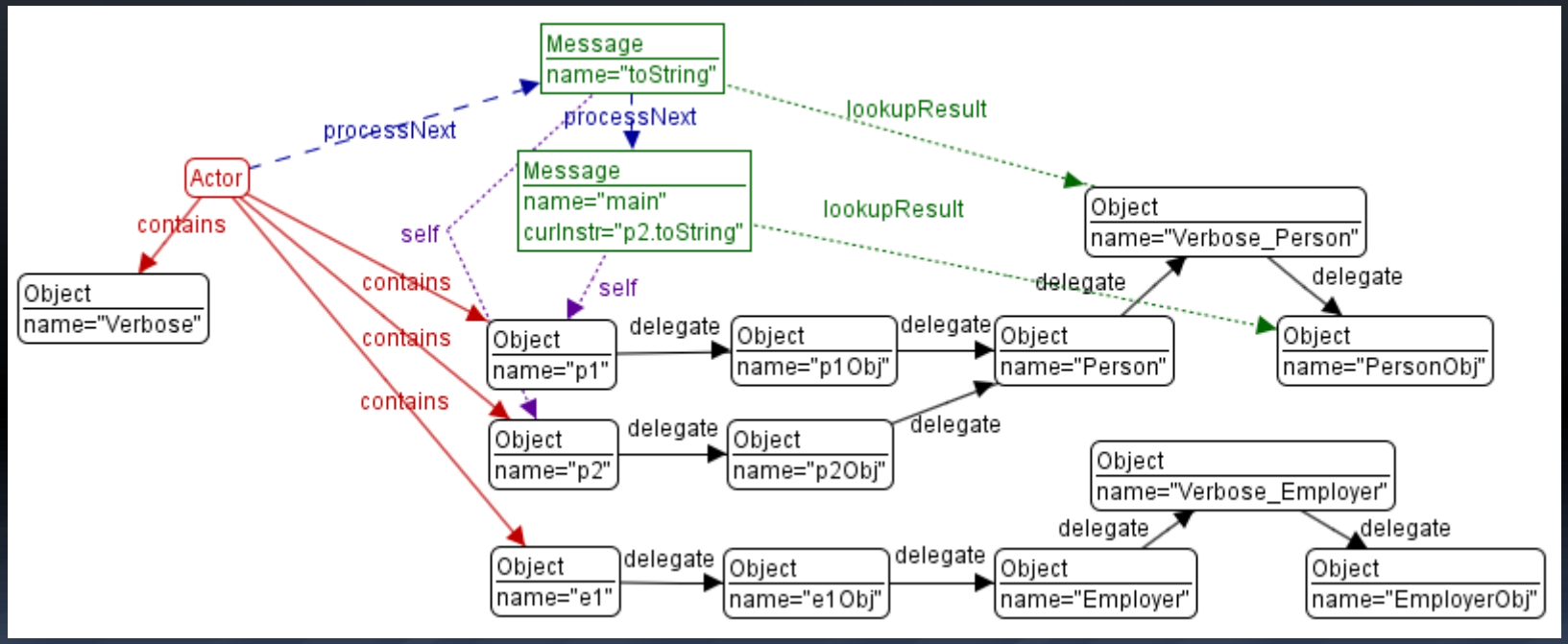
```
class Person {
        String name; Employer emp;
        String toString() {return name;}
        void main() {
                Employer e=new Employer("Cosmo Spacely",
                        "Sprocketlane 23, Orbit city");
                Person p=new Person("George Jetson", e);
                with(Verbose) {print(p.toString());}
        }
}
class Employer {
        String name; String addr;
        void toString() { return name; }
}
layer Verbose {
        void Person.toString() {return proceed()
                + "Employer:" + emp.toString();}
        void Employer.toString() {return proceed()
                + "Address:" + addr.toString();}
}
```

Initial program state




```
layer Verbose {
        void Person.toString() {...}
        void Employer.toString() {...}
}
```

After layer activation



Conclusion

- Graph rewriting as an alternative for specifying COP semantics:
  - Well-suited for experimentation, e.g.:
    - New language constructs
    - Specifying refactorings
  - Allows for learning by example
- Future work:
  - Define a graph-based semantic mapping from a COP language to delMDSOC