ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Modular Operational Semantics for CO-OPN$_{1.5}$ Objects

Didier Buchs

and

Jarle Hulaas

Swiss Federal Institute of Technology
Lausanne
Department of Computer Science

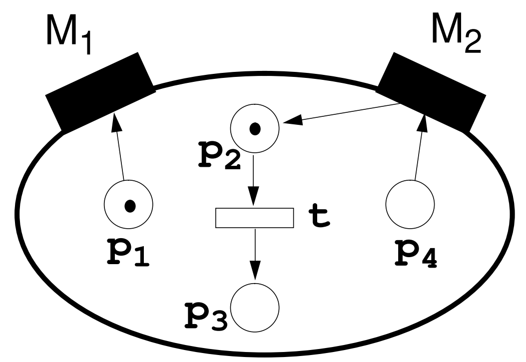
June 1996

Presentation Overview

- General form of the implementation
- Atomicity of method calls
- Resolution of synchronizations
- Strong synchronism
- Management of simultaneity
- CO-OPN prototypes in the real world

Basics of CO-OPN Objects

- Local algebraic Petri net
- Communication and synchronization = method calls + synchronization primitives
- Serve method calls only when stable (no active objects)



General Form of Implementation

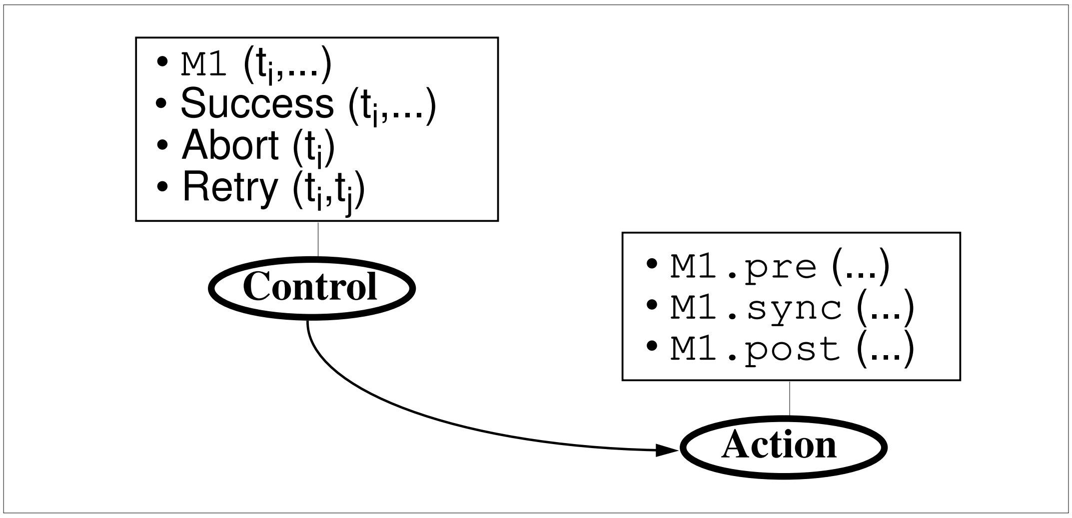
Targets: complex distributed systems on WANs

- One process per object
- Threads for managing simultaneous requests
- Communication by message-passing
- No logical variables

Mapping:

- CO-OPN Object $\rightarrow$ ***Control*** class + ***Action*** class
- Method $\rightarrow$ RPC Message to ***Control*** + function calls

Mapping of the CO-OPN Object



Atomicity of Method Calls

1. Method call succeeds: install new global state
   " " fails: restore state before call

   → **atomic actions (or transactions)**

2. Method depends loosely on success/failure in sub-objects

   → **nested transactions**

3. Semantics leave few chances for *unspecified* parallelism: requires independency in synchronization and stabilization

   → **pessimistic scheduling: 2 Phase Locking (2PL)**

Resolution-Oriented Methods

1. CO-OPN method $\cong$ predicate
   Modularity + encapsulation

$\rightarrow$ CO-OPN $\cong$ process-based distributed Prolog

2. Nondeterminism at 2 levels:
   - choice of axiom to service a method or transition
   - choice of tokens in input places

$\rightarrow$ (distributed) backtracking

Distributed Resolution and Transactions

Unify both concepts !

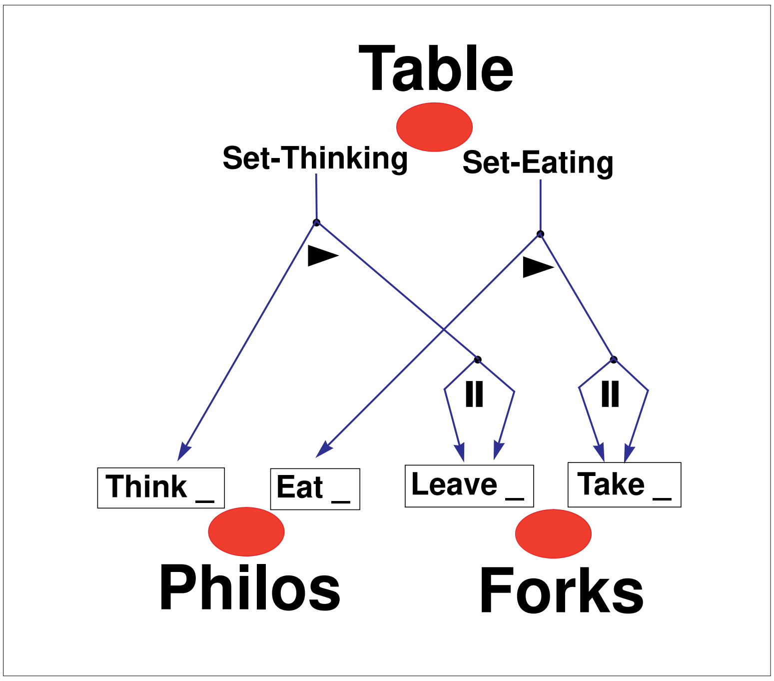
1. CO-OPN method invocation fails when missing ressources

$\rightarrow$ assimilate logical failure with hardware failure

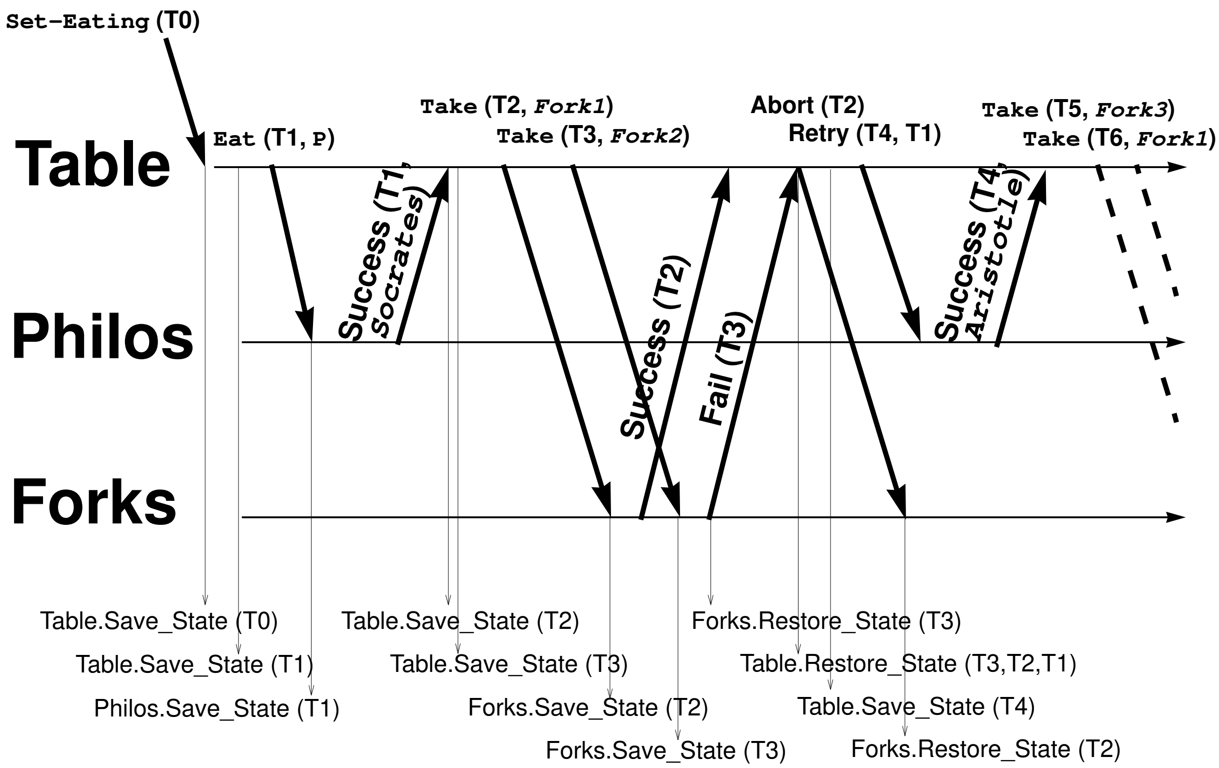
2. Nondeterminism + nested transactions:

$\rightarrow$ extension of 2PL protocol: `retry(Ti)`
$\rightarrow$ ability to restore *any* previous state of any object

Example: Failing Philosopher



Failing Philosopher (cont'd)



Strong Synchronism

1. Methods and transitions are considered *instantaneous*

   $\rightarrow$ **CO-OPN is a strong synchronous language**

2. Strong synchronism is a global property (hierarchical view)

   $\rightarrow$ **(specified) parallelism = simultaneity**
   $\rightarrow$ **sequences are atomic**

Operational Consequences

1. Sequences are atomic
   $\rightarrow$ **Objects must be locked (exclusively) to anyone outside the sequence**

2. Interpretation of simultaneous invocations on same object:
   - Share its internal ressources
   - Split into independent objects for invocation+stabilization

3. Possible sources of simultaneity for an object:
   A) Incoming requests arriving at the same moment
   $\rightarrow$ **requires notion of discrete time (sampling)**
   B) Simultaneity as a modelling facility for parallelism
   $\rightarrow$ **symbolic constraint for shared objects**

Operational View of Synchronization Primitives

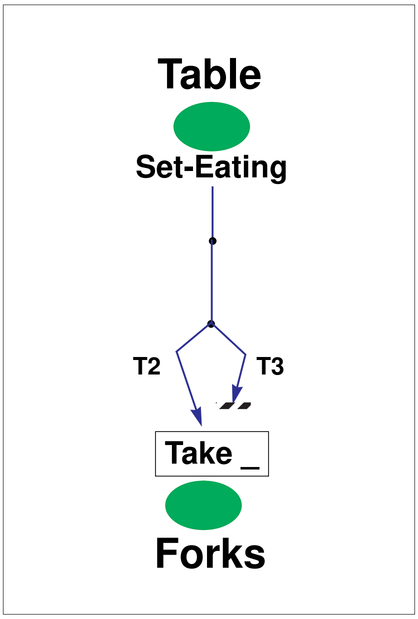
1. At the caller:
   - Sequence operator forces serial execution
   - Simultaneity “ ” parallel execution

2. At the callee:
   How does a shared object distinguish between
   - sequential invocations
   - simultaneous invocations separated by a long delay

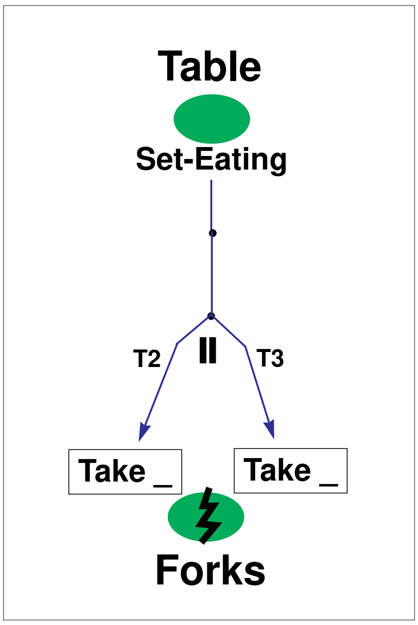
→ **the shared object must receive additional information (extension of 2PL protocol)**

Management of Simultaneity

Simple 2PL



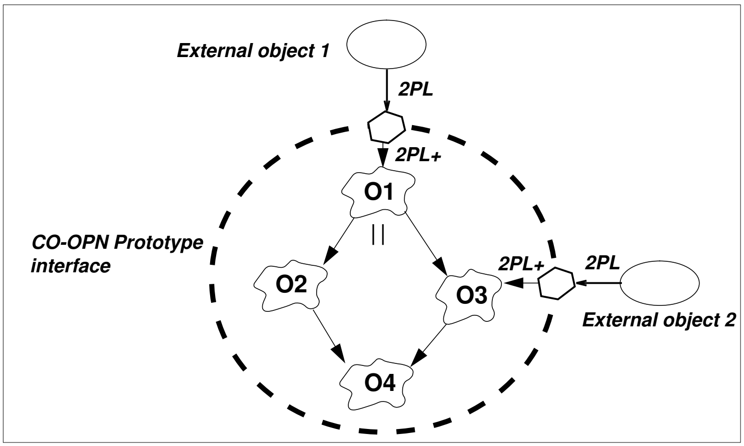
Extended 2PL



CO-OPN Prototypes in the Real World

- Prototypes should collaborate with external objects: *heterogeneous prototyping*

  → **generate interface objects for 2PL conversion (and transmission of virtual time)**



Conclusion

- Modular operational semantics with management of distributed simultaneity

- Extensions of 2PL protocol:
  - Transactions bring fault-tolerance to logical programs
  - Logical programs generalize nested transactions

- Under development:
  - Detailed algorithms
  - Implement prototyping tool