# Service Oriented Architectural Design

**Web Engineering Day**
**Athens, Sept. 3 2007**

Emilio Tuosto
Computer Science Department
University of Leicester

with

R. Bruni, A. Lluch Lafuente, and U. Montanari
Dipartimento di Informatica
Universita' di Pisa

## Motivations

SEnSOria aims to develop an approach
for engineering SOCs

- Key issues of service-based architectures:
  - design
  - reconfiguration
- Styles for reusing existing design patterns
- Run-time changes (e.g., dynamic binding)
  - require reconfigurations of architectures
  - complement their static reconfigurations
  - driven by architectural information specified during design
- Often, architectural styles must be preserved or consistently changed

# ADR principles

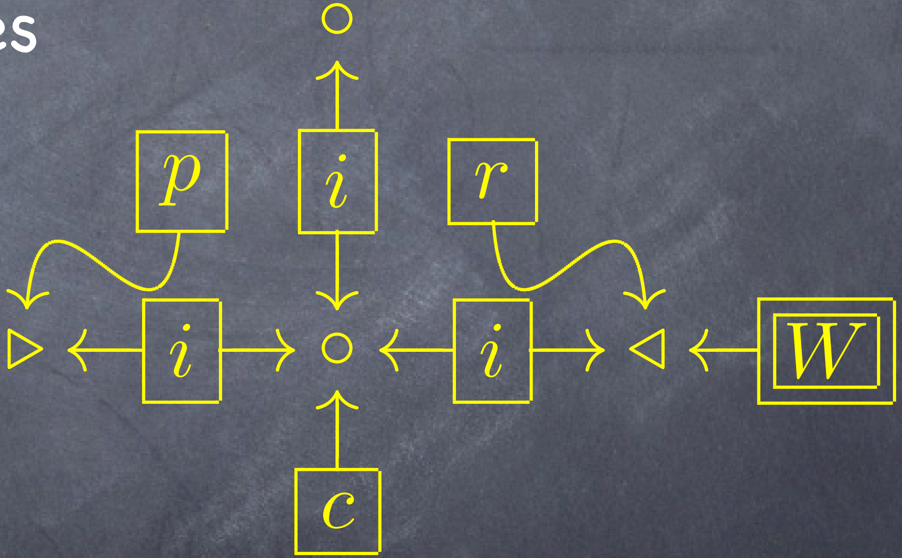
- Architectures are modelled as suitable graphs
- Hierarchical architectural designs
  - style preserving rules (not original)
  - algebraic presentation (original)
- Reconfigurations defined over style proofs instead of actual architectures
  - exploits the algebraic presentation
  - straightforward definition of hierarchical and inductive reconfigurations (ordinary term rewriting and SOS)
  - only valid contexts considered (not all concrete designs)
  - matching is simpler during reconfigurations (design driven)

## Overview

- Architectural Design Rewriting (ADR)
- Development/reconfiguration of software architectures
- Taking into accounts styles for “well-formed” reconfigurations
- Applying ADR to SRML so that SRML is respected by construction (i.e., style preserving rewritings)
- Concluding remarks

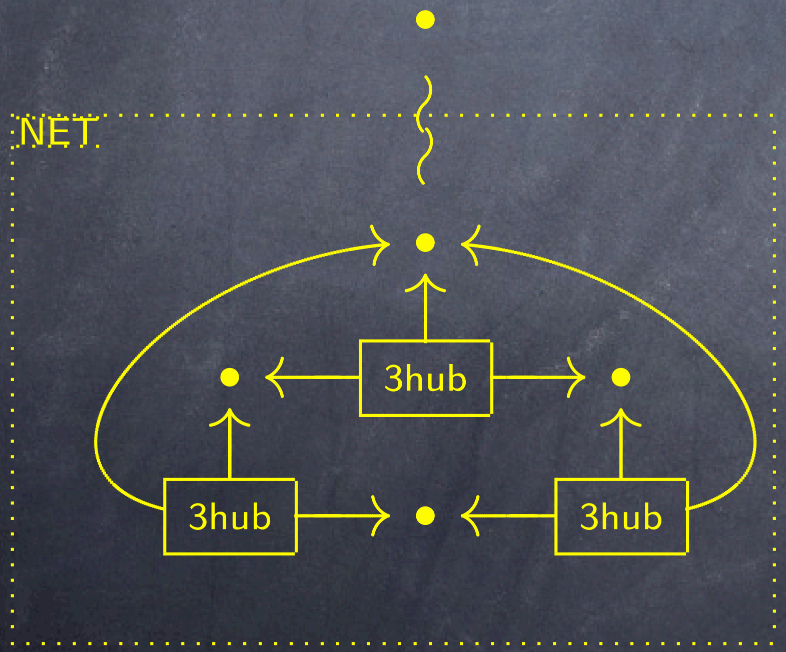
# ADR ingredients

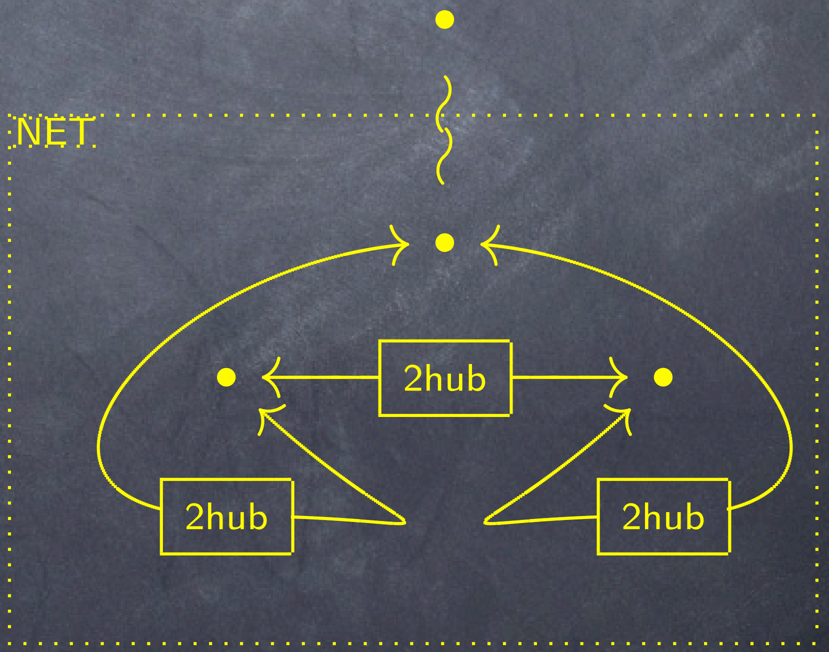
- Hypergraphs
  - edges model components: can be terminal and non-terminal edges
  - nodes model connecting ports
- Type-(hyper)graphs
- Productions
  - rules like L ::= R
  - specify how non-terminals should be replaced



# ADR by example

- A local networking architecture
- 2 styles where each network hub has degree of connectivity 2 or 3
- Connections between hubs are also driven by the style

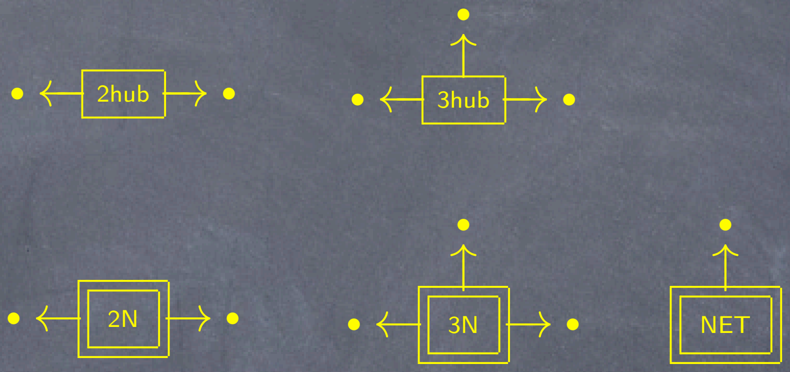




## Designs and productions

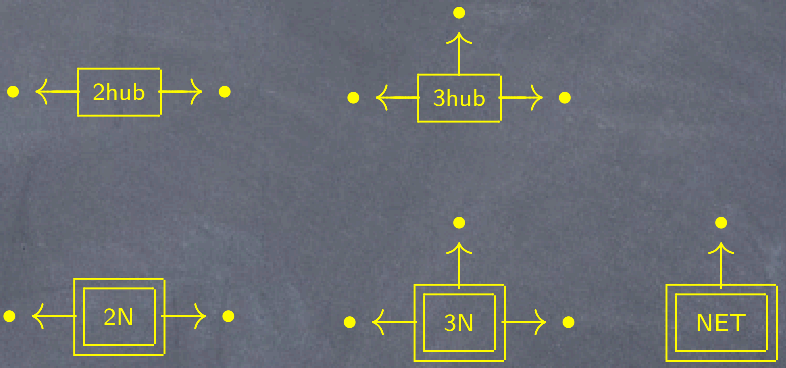
## Designs and productions

- Edges for the network example



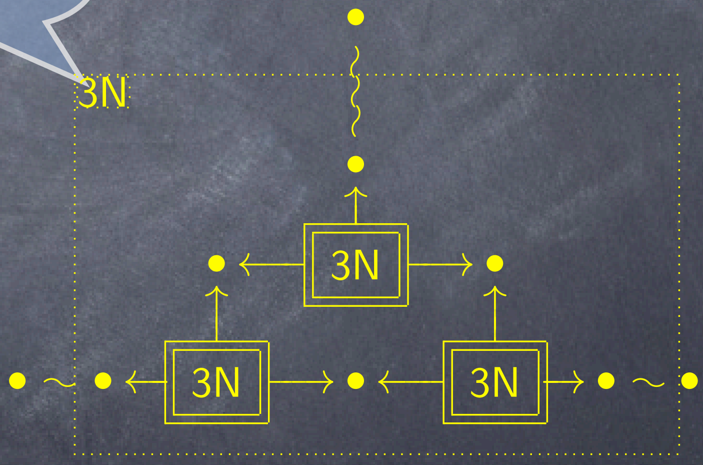
# Designs and productions

- Edges for the network example



- A **design** consists of
  - a lhs L which is a graph made of a single non-terminal edge
  - a rhs R graph possibly containing non-terminal edges
  - a map from the nodes of L to the nodes of R
- A **production** is a design where the occurrences of non-terminal are distinguished

represents the abstract class of the component



$3N ::= \texttt{link3}(3N, 3N, 3N)$

$\texttt{link3} : 3N \times 3N \times 3N \to 3N$

type of the production

## ADR methaphor

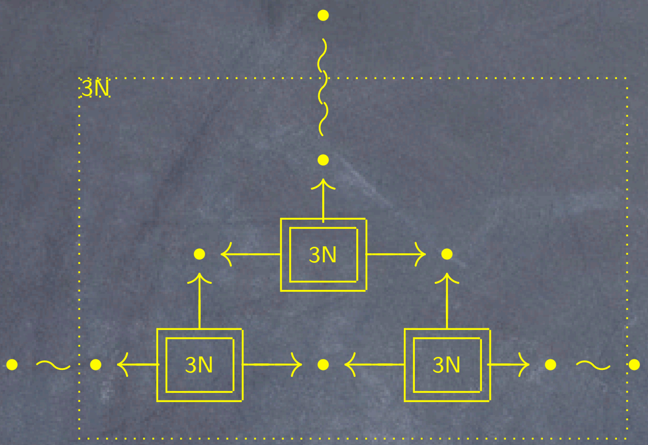
- A term of a grammar is an instance of a design
- Terms with variables are partial designs
- Replacing variables corresponds to refinement
- Replacing subterms with variables corresponds to abstraction
- Replacements are driven by term rewriting rules, namely reconfiguration rules t -> t’
    - style is preserved if t and t’ have the same abstract class
    - otherwise styles change...in a consistent way

# Design rewritings

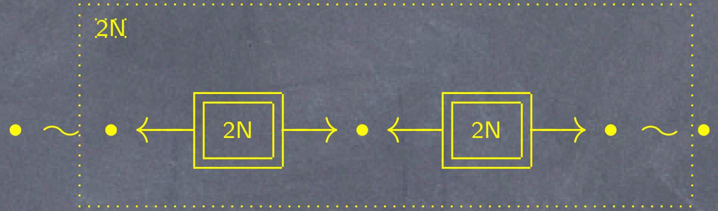
$$\texttt{link3to2}: \frac{x_1 \xrightarrow{\texttt{3to2}} x_1' \quad x_2 \xrightarrow{\texttt{3to2}} x_2' \quad x_3 \xrightarrow{\texttt{3to2}} x_3'}{\texttt{link3}(x_1, x_2, x_3) \xrightarrow{\texttt{3to2}} \texttt{link2}(\texttt{link2}(x_2', x_1'), x_3')}$$

## Design rewritings

$\texttt{link3} : 3N \times 3N \times 3N \longrightarrow 3N$



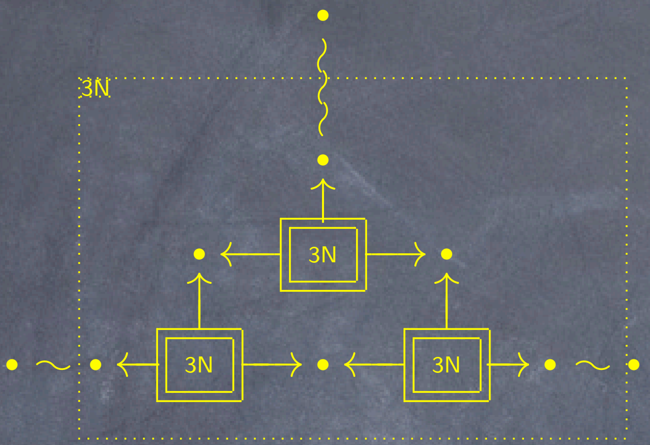
$\texttt{link2} : 2N \times 2N \longrightarrow 2N$



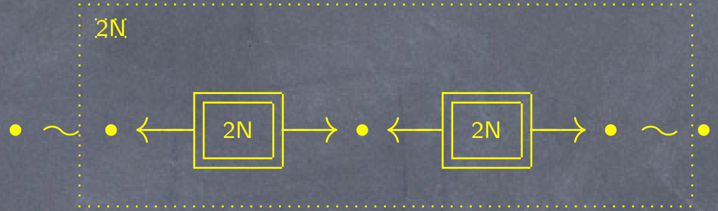
$$\texttt{link3to2} : \frac{x_1 \xrightarrow{\texttt{3to2}} x_1' \quad x_2 \xrightarrow{\texttt{3to2}} x_2' \quad x_3 \xrightarrow{\texttt{3to2}} x_3'}{\texttt{link3}(x_1, x_2, x_3) \xrightarrow{\texttt{3to2}} \texttt{link2}(\texttt{link2}(x_2', x_1'), x_3')}$$

# Design rewritings

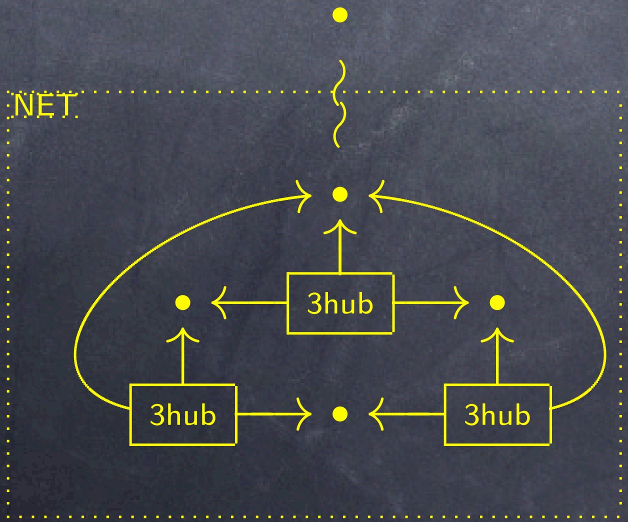
$\mathtt{link3} : 3N \times 3N \times 3N \rightarrow 3N$

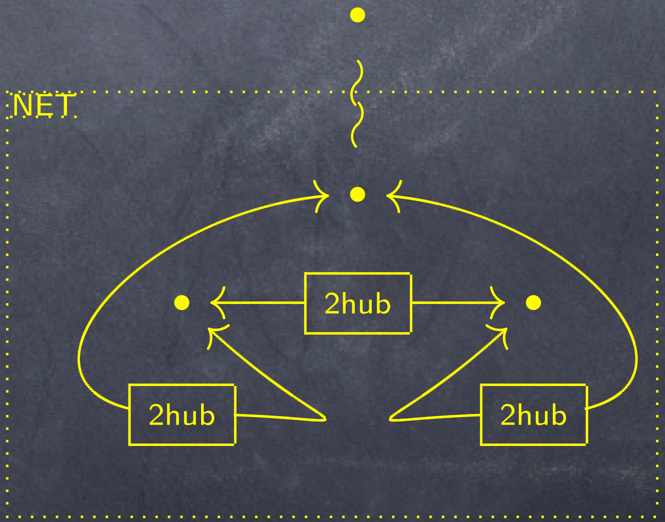


$\mathtt{link2} : 2N \times 2N \rightarrow 2N$

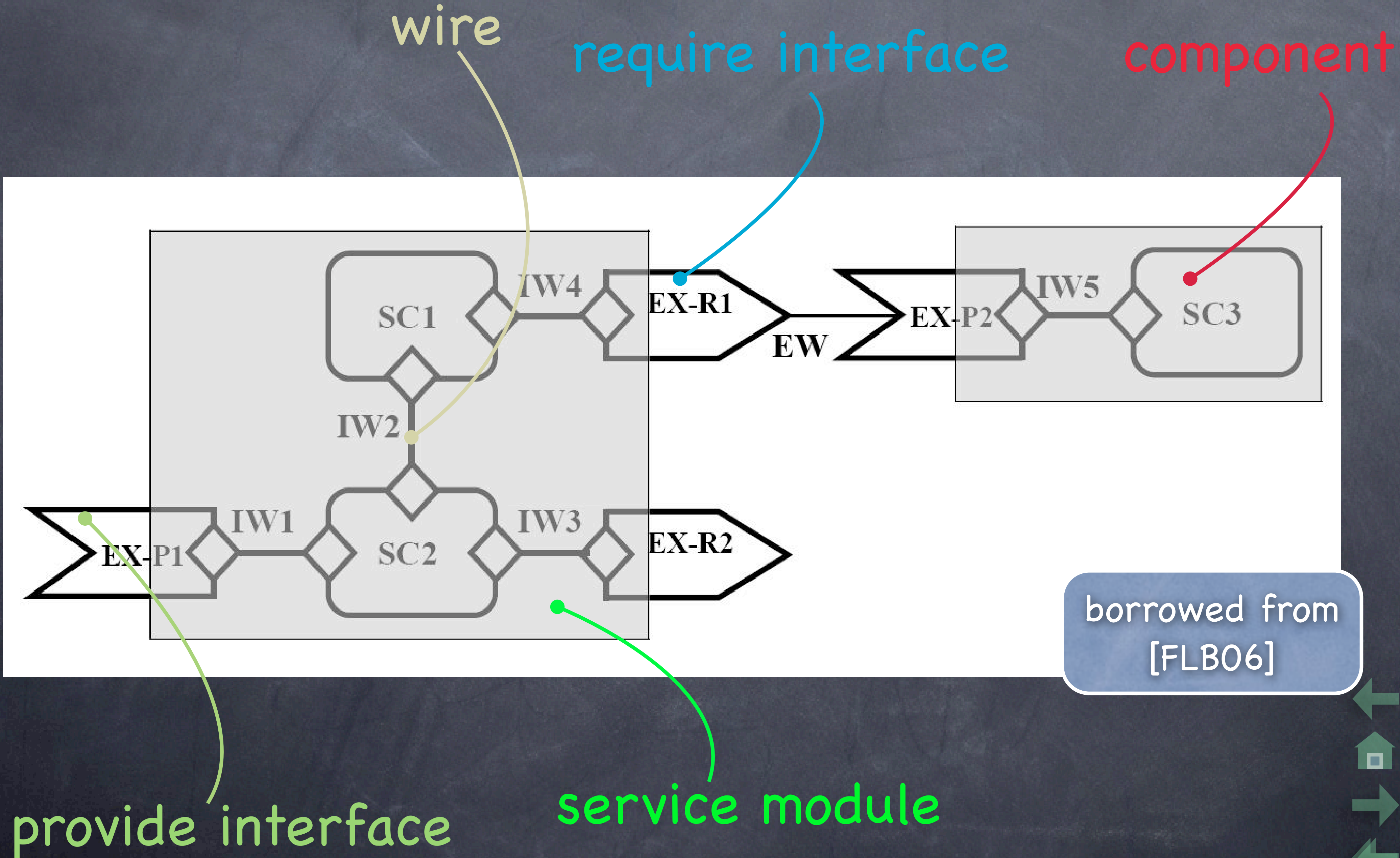


$$\mathtt{link3to2} : \frac{x_1 \xrightarrow{\mathtt{3to2}} x_1' \quad x_2 \xrightarrow{\mathtt{3to2}} x_2' \quad x_3 \xrightarrow{\mathtt{3to2}} x_3'}{\mathtt{link3}(x_1, x_2, x_3) \xrightarrow{\mathtt{3to2}} \mathtt{link2}(\mathtt{link2}(x_2', x_1'), x_3')}$$





# SRML architectural elements



borrowed from [FLB06]

## Terminals for SRML

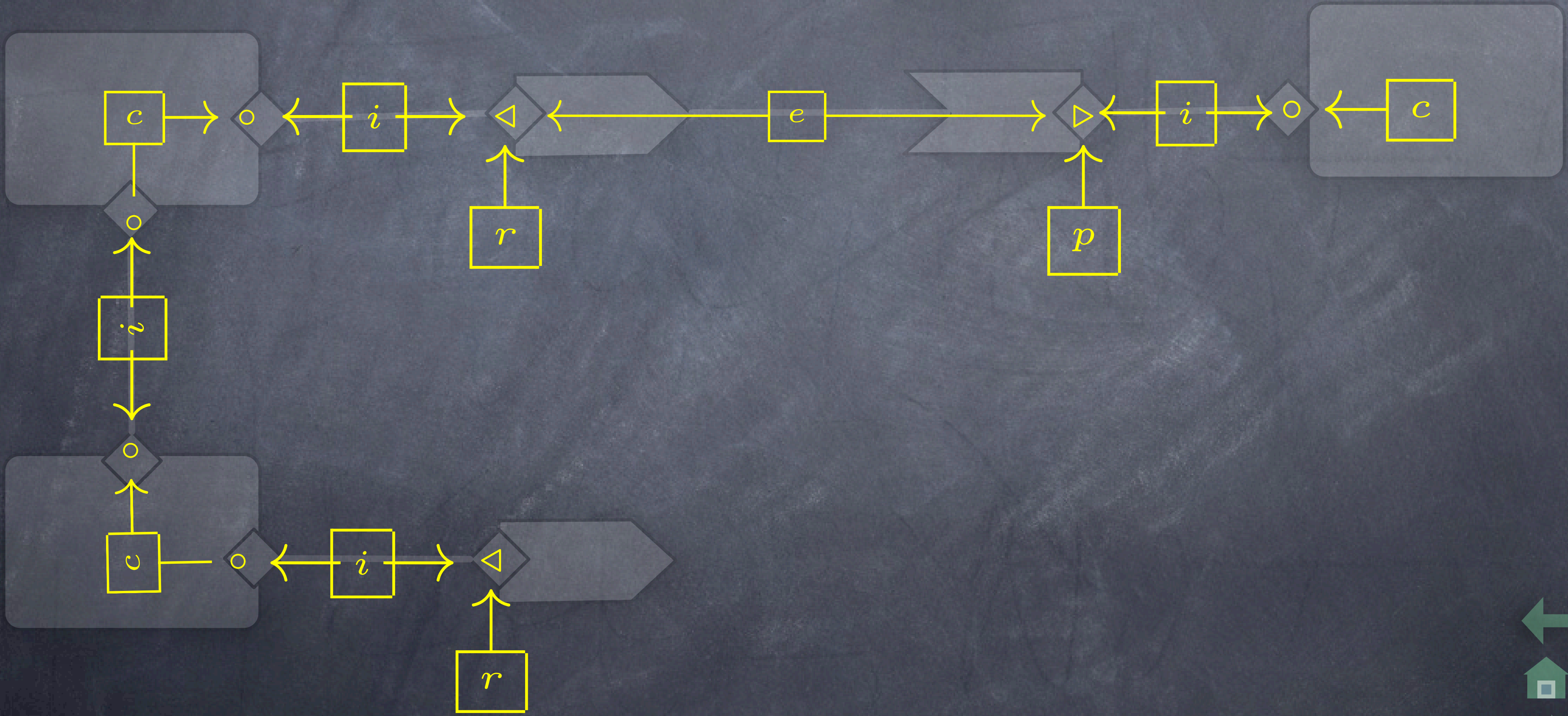
SRML components, wires and interfaces are modelled as terminal arcs

# Terminals for SRML

SRML components, wires and interfaces are modelled as terminal arcs

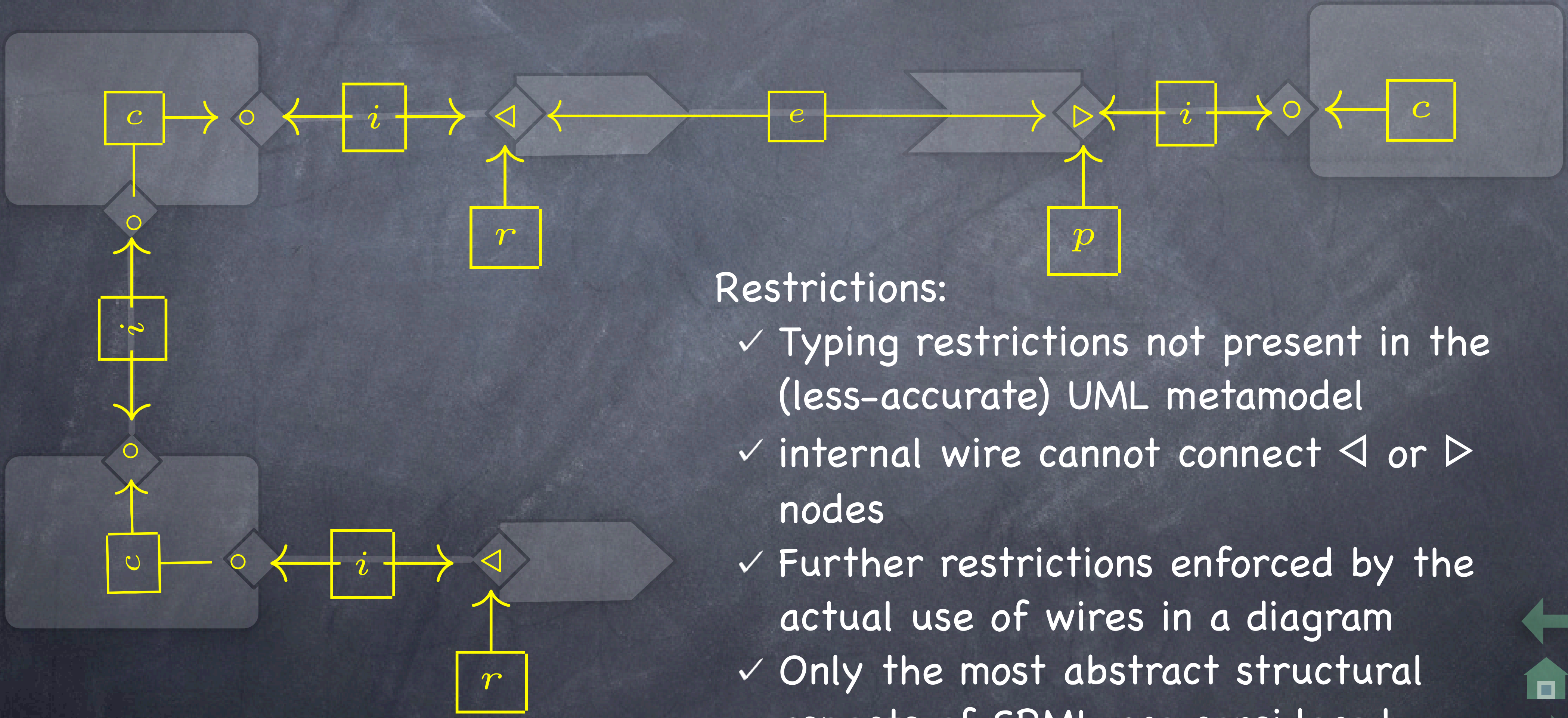
## Terminals for SRML

SRML components, wires and interfaces are modelled as terminal arcs



# Terminals for SRML

SRML components, wires and interfaces are modelled as terminal arcs



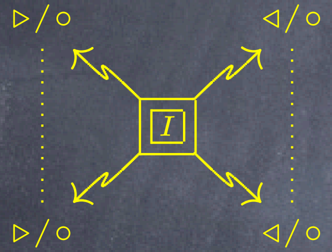
Restrictions:

- ✓ Typing restrictions not present in the (less-accurate) UML metamodel
- ✓ internal wire cannot connect $\triangleleft$ or $\triangleright$ nodes
- ✓ Further restrictions enforced by the actual use of wires in a diagram
- ✓ Only the most abstract structural aspects of SRML are considered

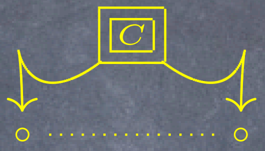
# Non-terminals for SRML

Non-terminals used as

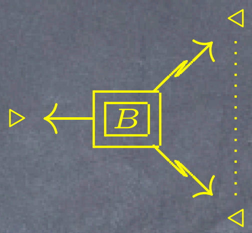
- the interface of a design (its type) and
- in the body of a design (as an abstract element)



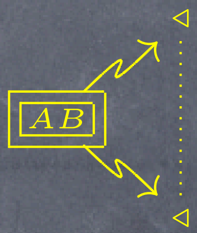
internal wires



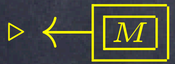
service components



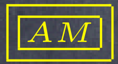
service module body



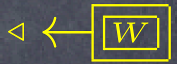
activity module body



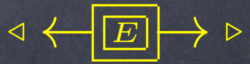
service module



activity module



wrapped service



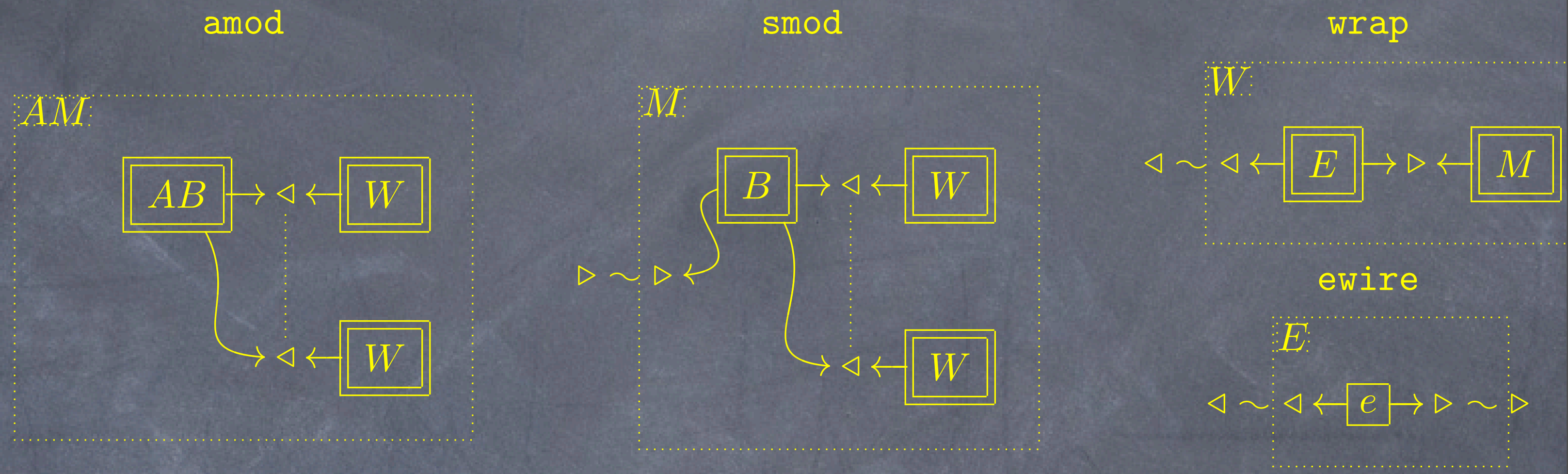
external wire

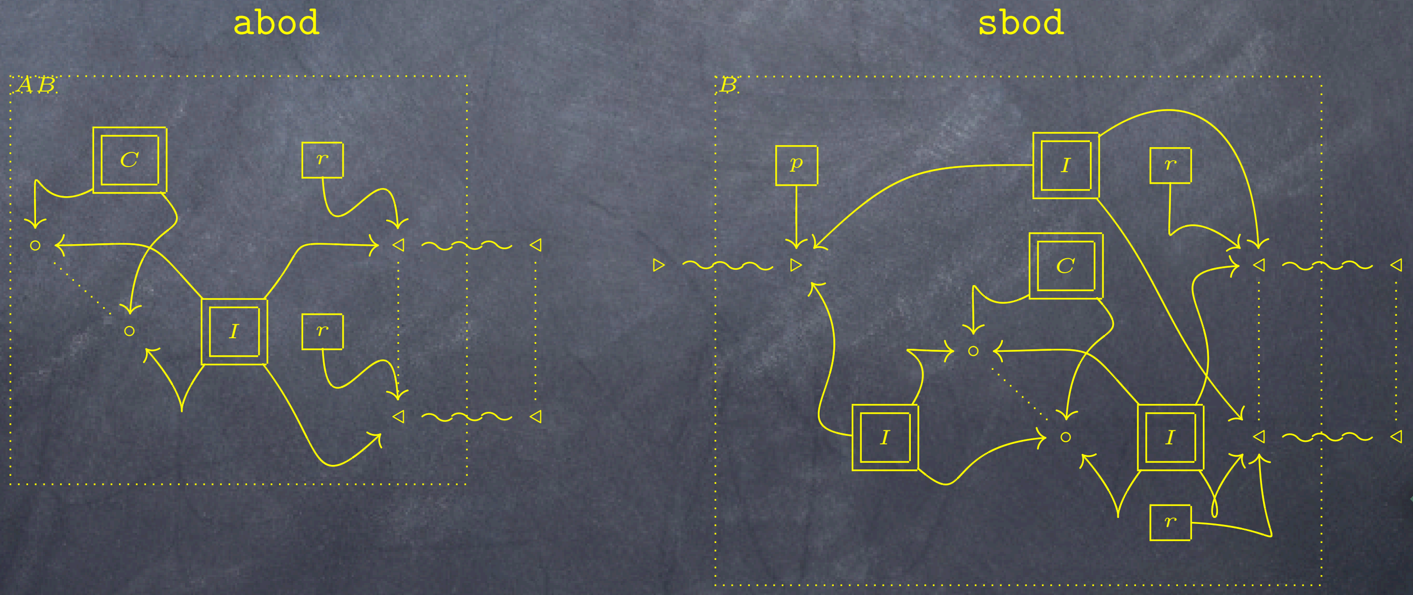
## ADR4SRML...top down

# ADR4SRML...top down

modules

bodies





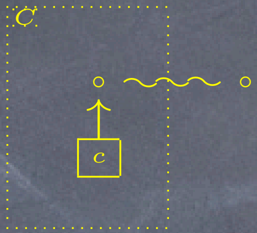
## ADR4SRML...top down

# ADR4SRML...top down

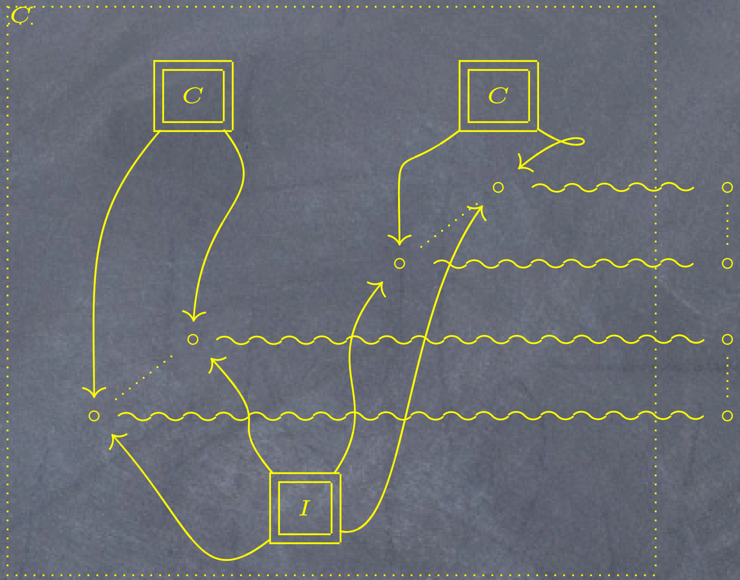
components

wires

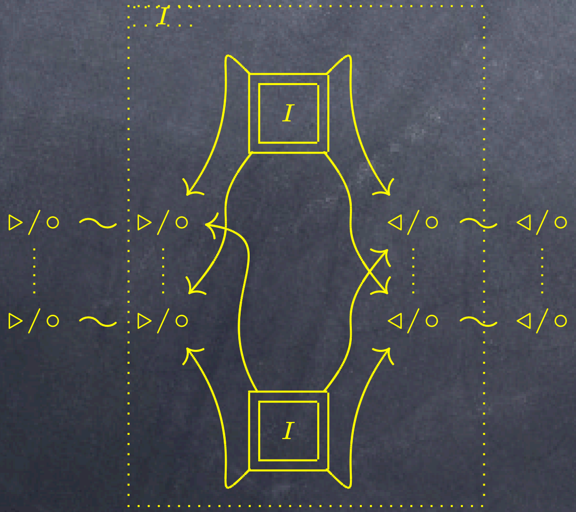
comp



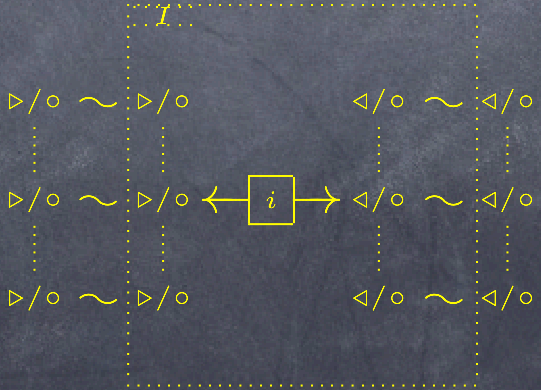
comps



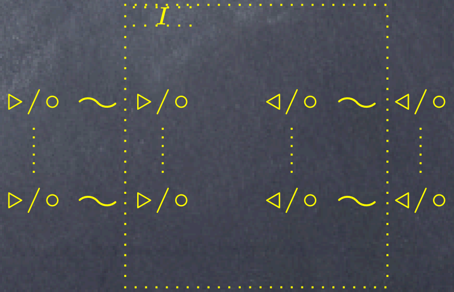
wires



iwire



nowire

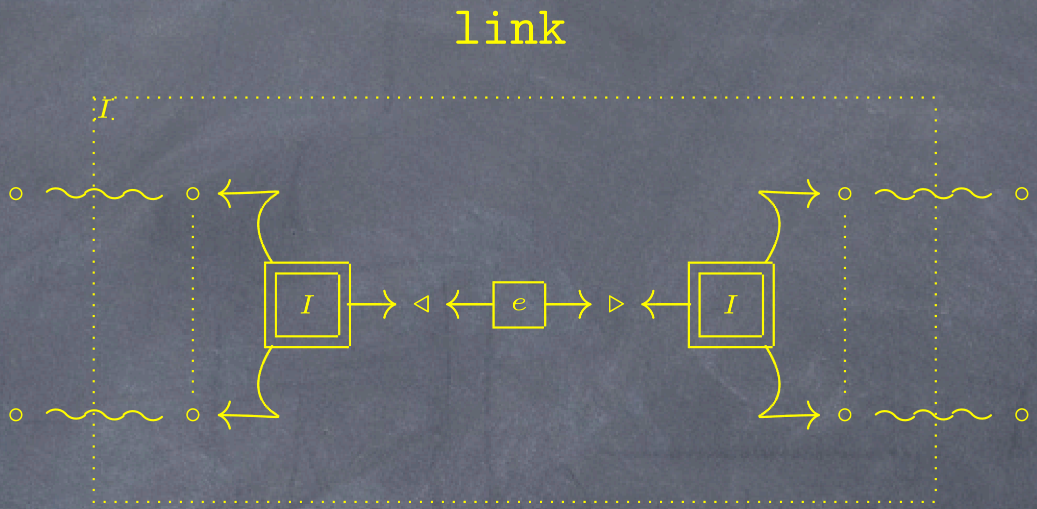


## ADR4SRML...

- Break-down SRML's composition operation
  - first wrap modules
  - then "internalise" wires
- An advantage is to get "consistency" by construction in SRML reconfigurations

# ADR4SRML...

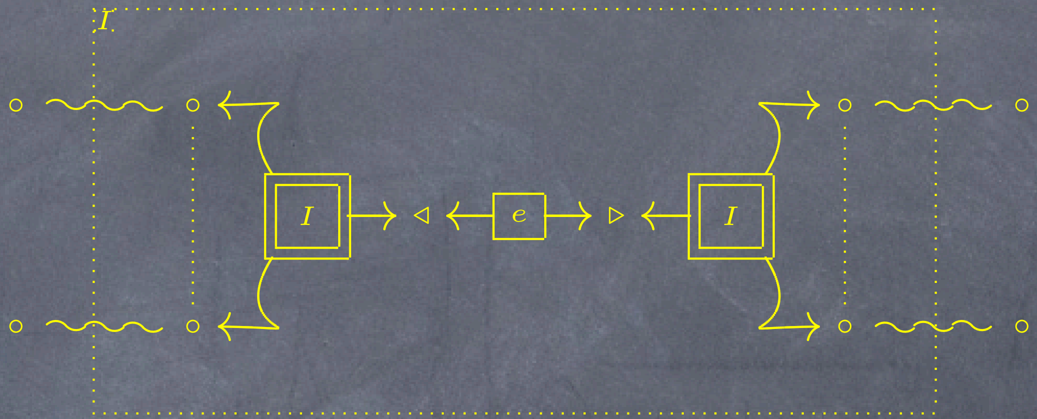
- Break-down SRML's composition operation
  - first wrap modules
  - then "internalise" wires
- An advantage is to get "consistency" by construction in SRML reconfigurations



# ADR4SRML...

- Break-down SRML's composition operation
  - first wrap modules
  - then "internalise" wires
- An advantage is to get "consistency" by construction in SRML reconfigurations

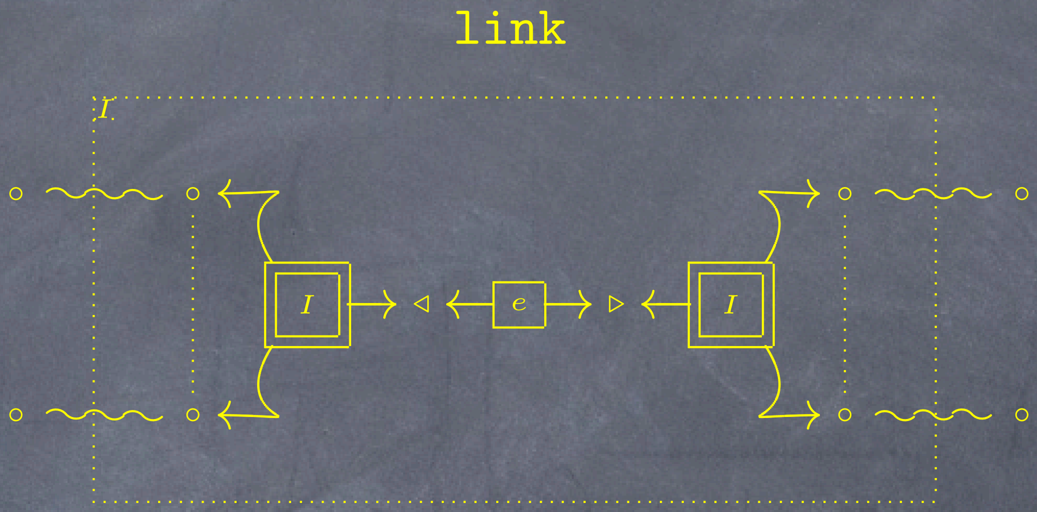
link



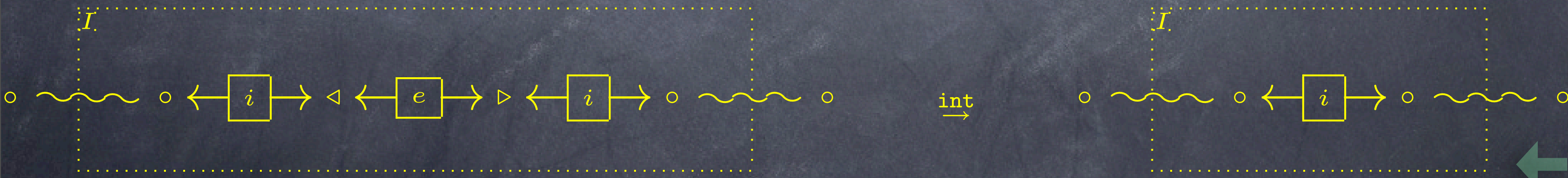
$\mathtt{link}(\mathtt{iwire}, \mathtt{iwire}) \xrightarrow{\mathtt{int}} \mathtt{iwire}.$

# ADR4SRML...

- Break-down SRML's composition operation
  - first wrap modules
  - then "internalise" wires
- An advantage is to get "consistency" by construction in SRML reconfigurations



$$\mathtt{link}(\mathtt{iwire}, \mathtt{iwire}) \xrightarrow{\mathtt{int}} \mathtt{iwire}.$$



## Conclusions

- We propose ADR as a framework for style-preserving reconfigurations of software architectures
- Based on algebra of typed-graphs with interfaces
- Hierarchical and inductive features for representing complex reconfigurations
- Formal model for SRML...reconfigurations of which are compliant with SRML meta-model by construction
- Future work: application of ADR to SOA

## Useful pointers

- A technical report is available at http://www.di.unipi.it/TR/ (TR-07-17)
- Emails
  - Roberto: bruni@di.unipi.it
  - Alberto: llafuente@di.unipi.it
  - Ugo: ugo@di.unipi.it
  - Emilio: et52@le.ac.uk