Architectural Design Rewriting as Architectural Description Language

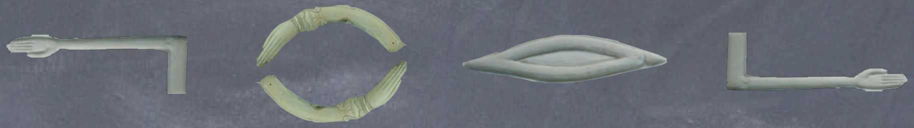
R. Bruni
A. LLuch-Lafuente
U. Montanari

E. Tuosto

Plan

- Architecture & SOC (our view)
- ADR
  - main features
  - ADR as ADL (through simple examples)

Models of SA



- [Perry & Wolf's, 92]
  - elements
  - form
  - rationale

- [Tracz, 93]: 4 'C's
  - components
  - connectors
  - configurations
  - constraints

Software architectures specify the design of system at a high level of abstraction (not the implementation level):

- the structure of components
- how they are interconnected
- (valid) architectural configurations (aka topologies), i.e.
  - present components
  - interconnections
  - their current state

Models of SA

$$\begin{tikzcd}
\text{[Perry \& Wolf's, 92]} & \text{[Tracz, 93]: 4 `C's} \\
\text{elements} & \text{components} \arrow[l] \\
 & \text{connectors} \arrow[ul] \\
\text{form} & \text{configurations} \arrow[l] \\
\text{rationale} & \text{constraints} \arrow[l]
\end{tikzcd}$$

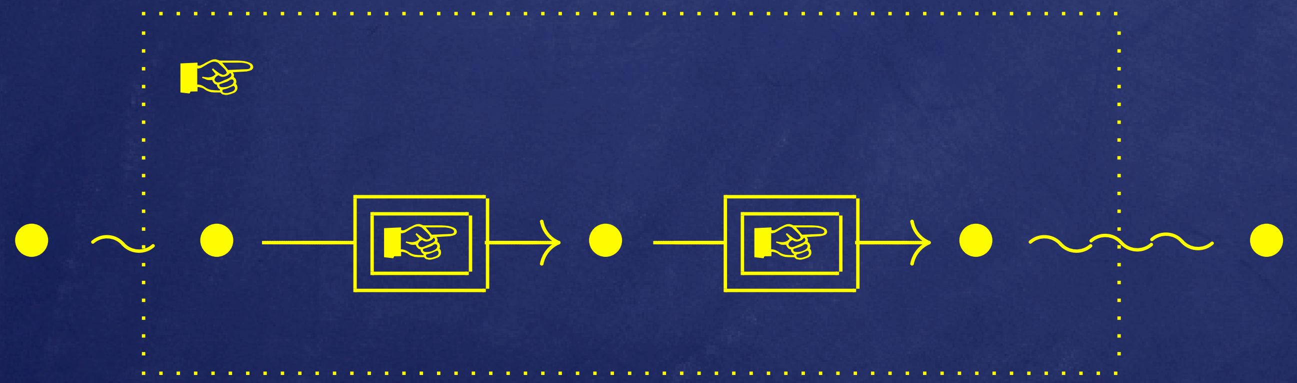
Software architectures specify the design of system at a high level of abstraction (not the implementation level):

- the structure of components
- how they are interconnected
- (valid) architectural configurations (aka topologies), i.e.
  - present components
  - interconnections
  - their current state

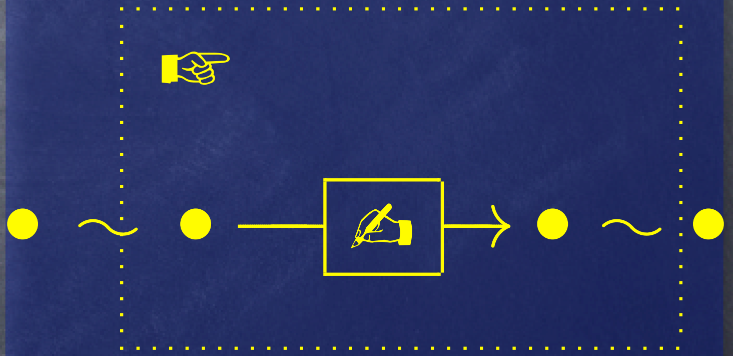
ADR's Key features

- Hierarchical/graphical design & algebraic presentation
  - Architectures as typed designs
  - Composed through design productions (operators)

$\mathtt{pipe} : ☞ \times ☞ \longrightarrow ☞$

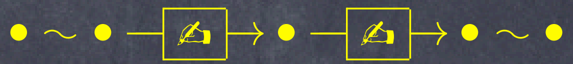


$\mathtt{atom} : \longrightarrow ☞$

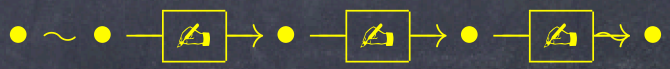


Examples

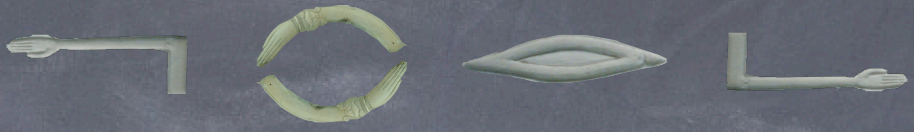
$\mathtt{pipe}(\mathtt{atom}, \mathtt{atom})$



$\mathtt{pipe}(\mathtt{pipe}(\mathtt{atom}, \mathtt{atom}), \mathtt{atom})$

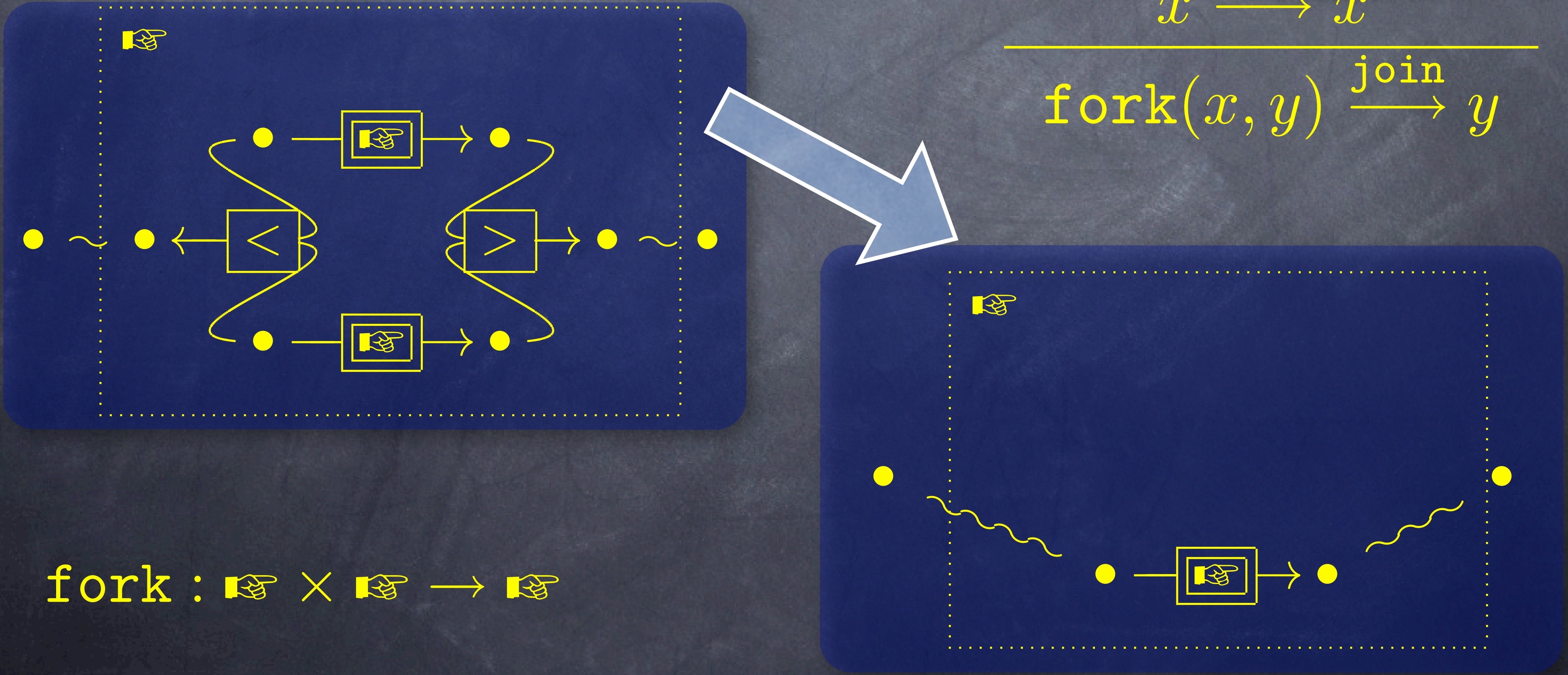


ADR's Key features



- Rule-based approach & inductively-defined reconfigurations
  - SOS
  - conditional term rewriting
- Constraints and architectural styles via **types**

$$\frac{x \xrightarrow{\texttt{stop}} x'}{\texttt{fork}(x,y) \xrightarrow{\texttt{join}} y}$$



$\texttt{fork} : ☞ \times ☞ \to ☞$

ADR “expressivity”

- Typed designs (graphs + interfaces)
  - styles as design terms
  - architectures as designs (i.e., graphs interpreting of design terms)
- Hierarchical design (productions as operators of a multisorted algebra of designs)
  - refinement (top-down)
  - bottom-up (typing and well-formed composition)
- Reconfiguration as conditional term rewriting over design terms (rather than over designs)
  - style conformance can be guaranteed by construction

ADR as ADL

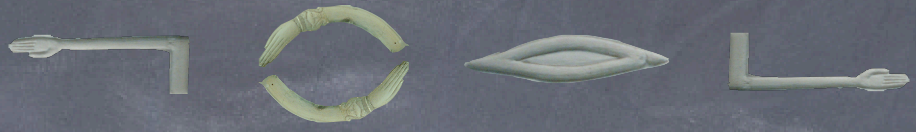
"An ADL must provide the means for their[1] explicit specification"
[Medvidovic & Taylor, 00]

[1]components (with interfaces), connectors and configurations

ADR meets most of the requirements of an ADL

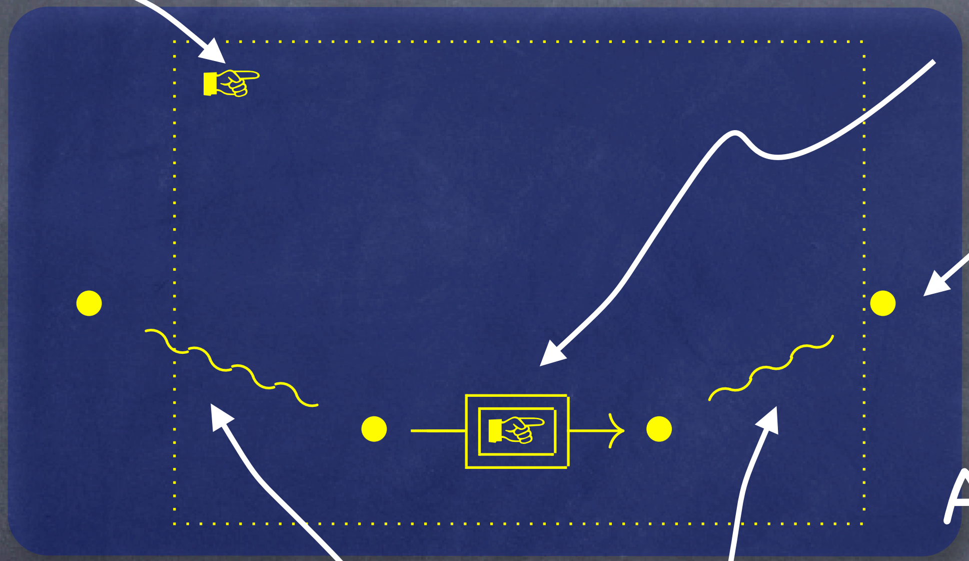
- Components/connectors
  - Typed elements with interfaces
  - Formal semantics
  - Constraints
  - Evolution
- Architectural configurations
  - Compositionality/ Understandability
  - Refinement
  - Traceability
  - Scalability/Dynamism

Types&Interfaces



Type

Nodes & hyperedges can be typed



Interfaces

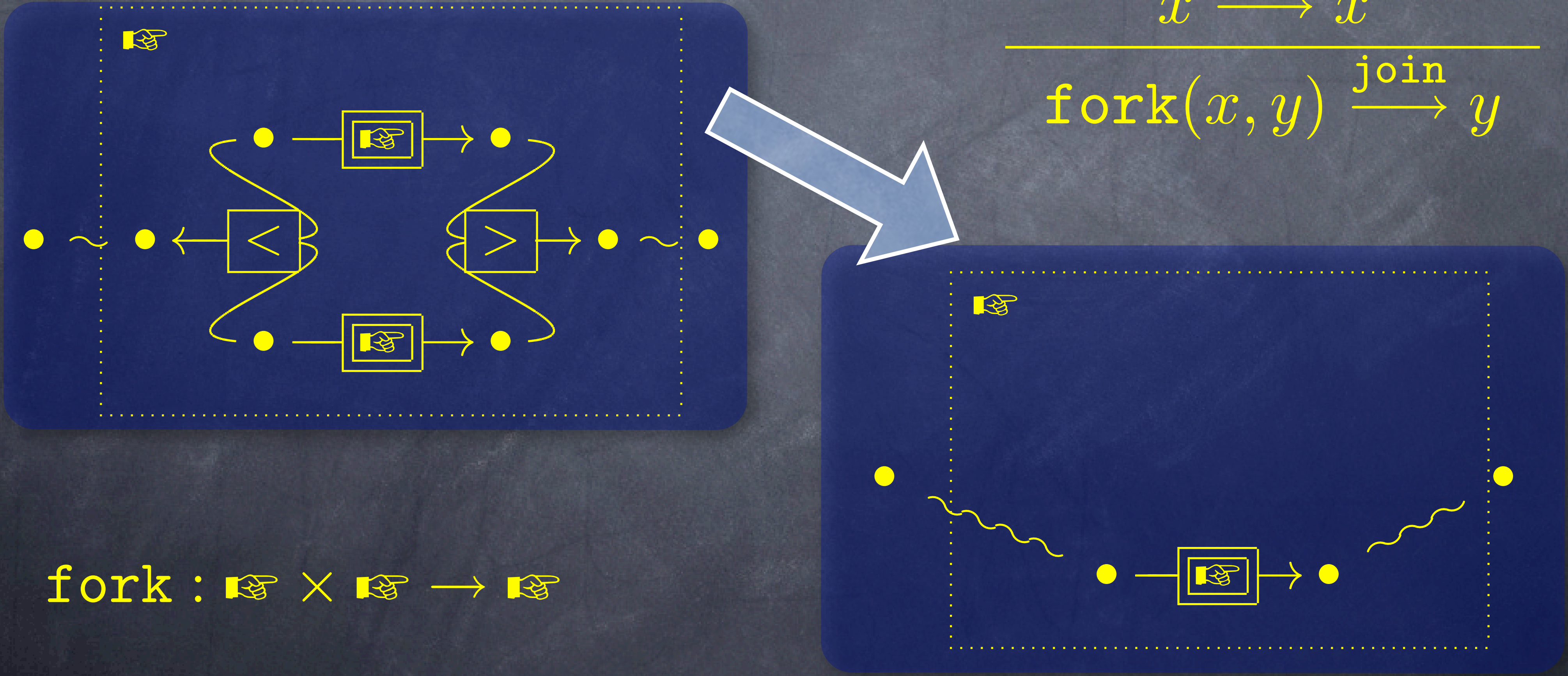
ADR promotes types for encoding constraints when possible, so that constraints preserving reconfigurations are given by construction

Semantic/Evolution

Algebraic graph transformation / SOS

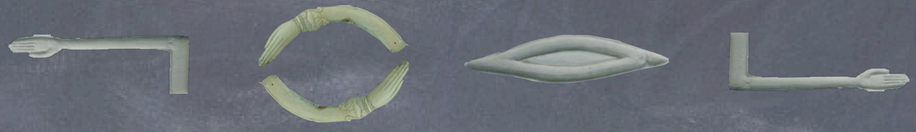
conditional term rewriting

$$\frac{x \xrightarrow{\texttt{stop}} x'}{\texttt{fork}(x,y) \xrightarrow{\texttt{join}} y}$$

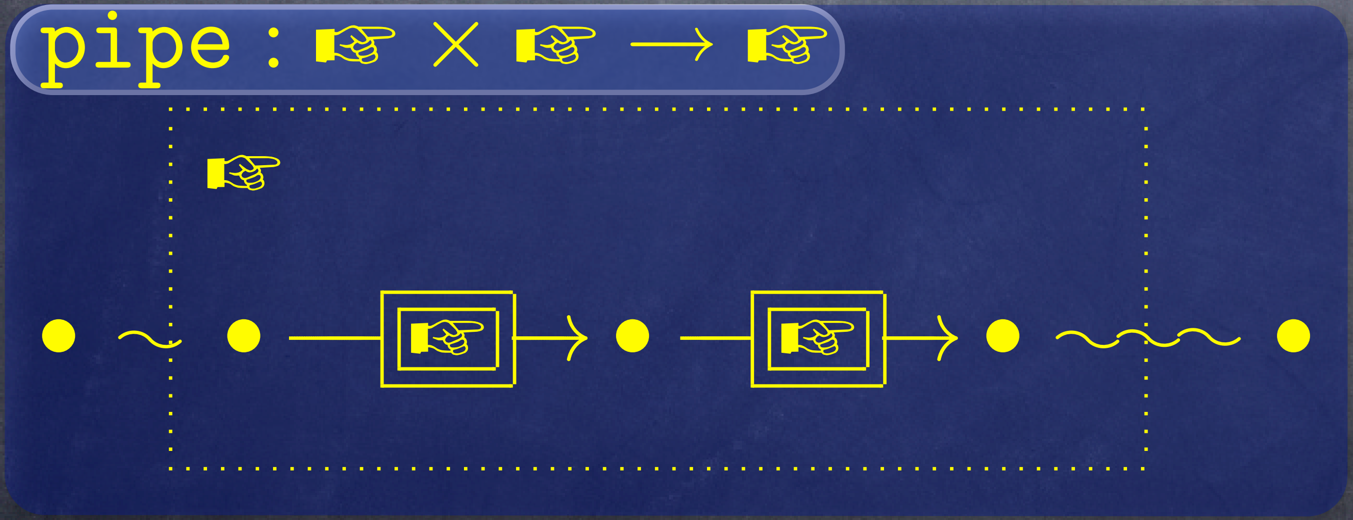


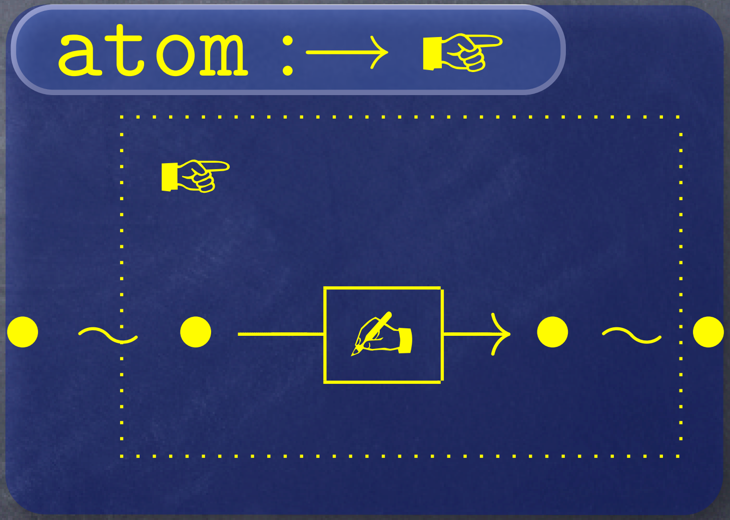
$\texttt{fork} :$ ☞ $\times$ ☞ $\longrightarrow$ ☞

Compositionality

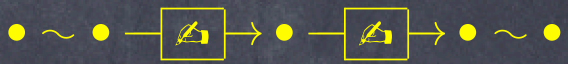


Compositionality achiedved using **design productions** that yield **hierarchical composition** (featuring also understandability)

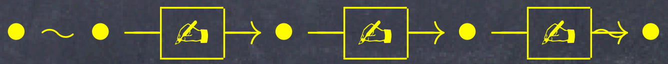




pipe(atom, atom)

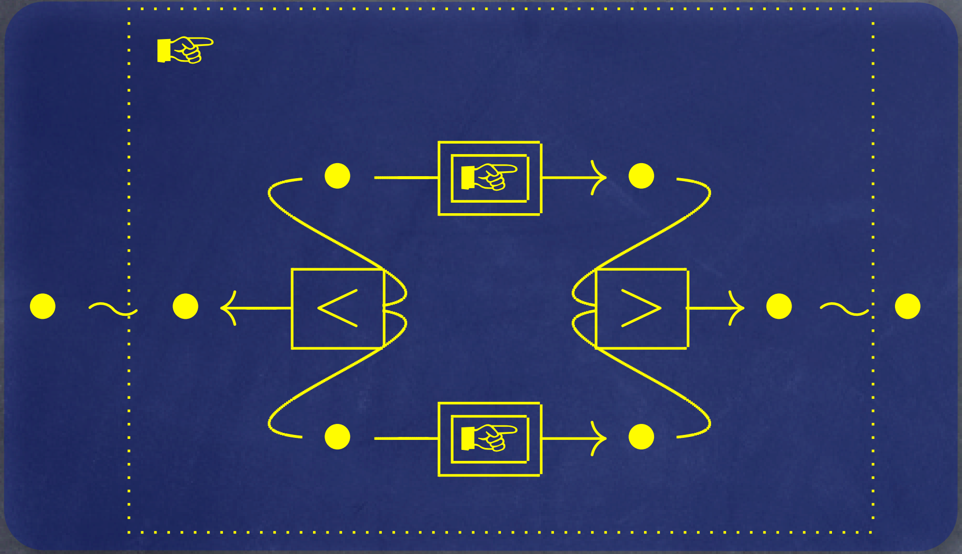


pipe(pipe(atom, atom), atom)



Examples

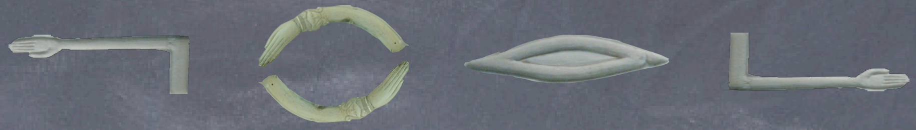
Refinement



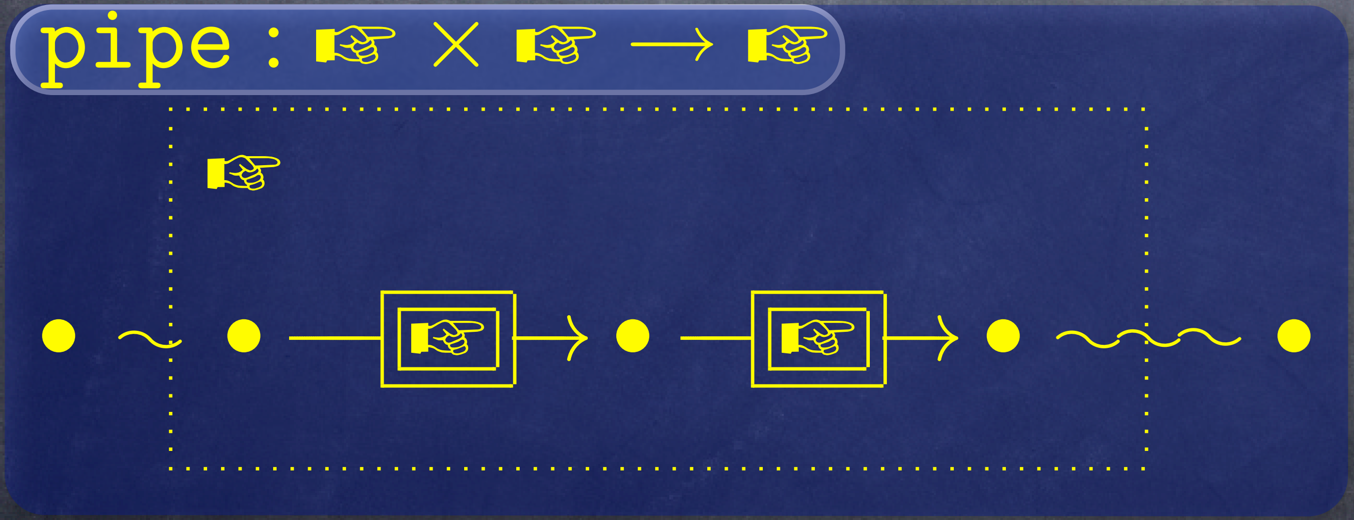
Design production can be read "top-down": a 'pipe' can be refined by forking two parallel 'pipes'

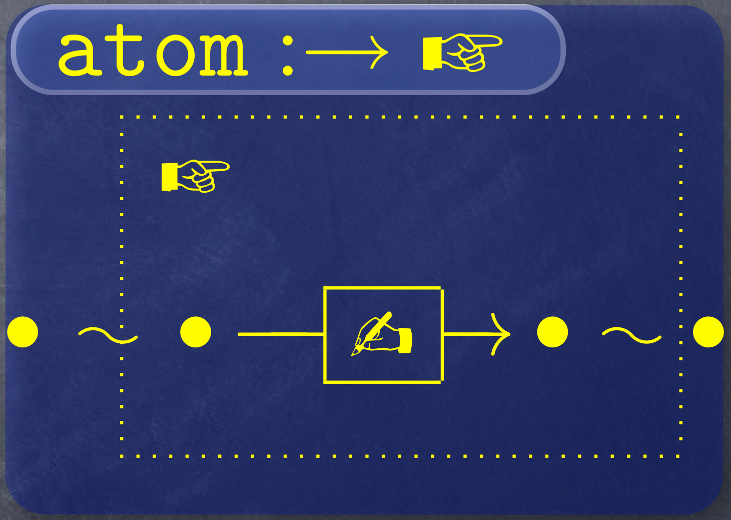
Remarkably, design production can be read "bottom-up" as well: the forking 'pipes' are valide provided that the two inner 'pipes' are

Traceability

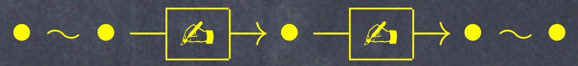


A design (i.e. an actual architecture) are traced trough a design terms namely a “witness” of the design construction

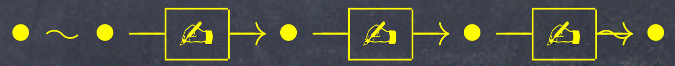




pipe(atom, atom)



pipe(pipe(atom, atom), atom)



Dynamism

Architectural changes are expressed in ADR by conditional rewrite rules in a SOS style in order to define complex behaviours and reconfigurations.

ADR yields a modular approach, so that, e.g., the addition of new components can be localised in the desired sub-architecture, without affecting the rest of the system.

References

- ADR site http://www.albertolluch.com/adr.html
- [Perry & Wolf's, 92]: "Foundations for the study of software architectures". SIGSOFT Software Eng. Notes, V. 17, No. 4, October 1992
- [Tracz, 93]: "LILEANNA: A parameterized programming language". Proc. 2nd Int. Workshop on Software Reuse and Eng. Center. July 1995
- [Medvidovic & Taylor, 00]: "A classification and comparison framework for software Architecture Description language". IEEE trans. on Soft. Eng., V. 26 N. 1, January 2000