

# On Observation as a Coordination Paradigm: an Ontology and a Formal Framework

Mirko Viroli, Gianluca Moro, Andrea Omicini  
DEIS - Università degli Studi di Bologna  
{[mviroli](mailto:mviroli@deis.unibo.it),[gmoro](mailto:gmoro@deis.unibo.it),[aomicini](mailto:aomicini@deis.unibo.it)}@deis.unibo.it



# Outline

- We introduce the Coordination Paradigm named Observation:
  - ⇒ modeling an observer willing to receive information about a source
  - An ontology capturing all most common schemata
  - A formal framework, which can be used:
    - as a specification tool
    - for comparing systems
- We describe some properties of the architectures:
  - JavaSpaces, Linda, TuCSoN, ...
  - ⇒ exploiting the framework to model recurrent behaviours

# Observing a source system

- We focus on the “observation actions”  
⇒ an observer system gets information from a knowledge source
- Different cases of observation:
  - Request-Reply (a.k.a. Client Server, ...)
  - Publish-Subscribe (a.k.a. Event Notification, ...)
  - Trigger entities (a.k.a. Internal Reactions, ...)
  - ....
- Goals of the paper:
  - to make up for the lack of a common ontology
  - to provide a unique setting for widely adopted concepts
  - to allow for a clear understanding of several observation features

# Hypothesis

- How we model interactions which are not about observation?
  - E.g. operations just changing the source
    - ⇒ we suppose the state of a knowledge source may change with unpredictable dynamics
  - we don't want to model the "Control paradigm"
- However: we do model internal control activities:
  - ⇒ they are often used to change the behaviour of a source
  - ⇒ they can be seen as a sort of "internal observers"
- We abstract away from:
  - Observations involving multiple sources
  - Issues about transactions

# Patterns of observation

- Some examples of recurrent observation schemata:
  - *Static observation*:
    - the *observer* is willing to receive some information as soon as possible  $\Rightarrow$  this calls for a *reactive* manifestation
  - *Dynamic observation*:
    - the observer is willing to be informed when relevant events occur in the source  $\Rightarrow$  this calls for a *proactive* manifestation
  - *Direct manifestation*:
    - the information is provided to the observer with a message
  - *Indirect observation*:
    - an *internal observer* is configured in the source, possibly altering its behavior so as it is perceived by (external) observers

# An ontology for observation

- Four entities involved in this coordination pattern:
  - Source systems:
    - show *observable items* (for static observation)
    - show *observable events* (for dynamic observation)
  - Observer systems:
    - receive messages from the source containing:
      - the value of some observable item
      - the specification of the observable events occurred
  - *Coordinators*: configure observation actions in a source
    - by sending a request message, or at start-up
    - often: coordinator and observer coincide
  - Internal observers: local processes altering the source behaviour
    - similar to observers, but their boundary is within the source

# A key concept: the configuration

- The *Configuration*:
  - is configured by the coordinator into the source
  - the source keeps a *configuration set*
  - each configuration may be responsible for one or more observation actions
- Configurations are evaluated by the source:
  - what happens?
    - a message sent to an observer
    - a change in the source
    - the configuration is re-written
  - when to evaluate a configuration?
    - as soon as it is registered in the source
    - each time an observable event occurs
    - the first time an observable event occurs
    - until a timeout expires ....

# Applying the ontology

- Method invocation in O-O systems:
  - Reactive and direct manifestation
  - ⇒ A configuration immediately evaluated and sending a message
- Notification in JavaSpaces:
  - Proactive and direct manifestation
  - ⇒ A conf. evaluated when an event occurs, sending a message
- Eval in Linda:
  - Reactive internal observer
  - ⇒ A conf. immediately evaluated, changing the source
- Trigger in Active DataBases:
  - Proactive internal observer
  - ⇒ A conf. evaluated when an event occurs, changing the source

# Need for a formal framework

- The ontology is a qualitative tool:
    - provides a common terminology
    - allows to compare aspects seemingly diverse
  - We are interested in formal descriptions:
    - providing a formal model for observation
    - emphasizing the key issues of the ontology
    - providing a specification tool for new and existing systems
    - precisely characterizing frequently occurring patterns
- ⇒ We introduce a formal model based on Transition Systems

# Formalization of a source

- A source system is mainly characterized by:
  - The set of positions:  $P$
  - The set of configurations it can handle:  $C$
- when it's useful to abstract from details of  $P$ , one can define:
  - The set of observable items  $V$  ( and  $R$  mapping  $P$  in  $V$ )
  - The set of observable events  $E$  ( and  $G$  mapping  $P \times P$  in  $E$ )
- The observation behavior of a source is characterized by:
  - An interpretation function  $J$ , which given a source transition ( $P \times P$ ):
    - $\Rightarrow$  Evaluates a configuration  $c$  returning a quartet:
      - A flag (guard) stating whether  $c$  applies
      - How  $c$  should be rewritten after evaluation
      - What changes should be made to the source
      - What messages have to be produced

# Formalization of observation

- The source status is described by a *global source position*:
  - the current transition of the position:  $\langle p, p' \rangle$
  - the configuration set:  $\underline{c}$
  - the pending messages:  $\underline{m}$
- The dynamics is described by means of a LTS:
  - source transition S (change on the source)
  - registration transition R (a configuration registered)
  - evaluation transition E (evaluation of the configuration set)
  - output transition O (sending of messages produced)
  - continuation output transition Q  
(sending messages followed by a new round of evaluation)

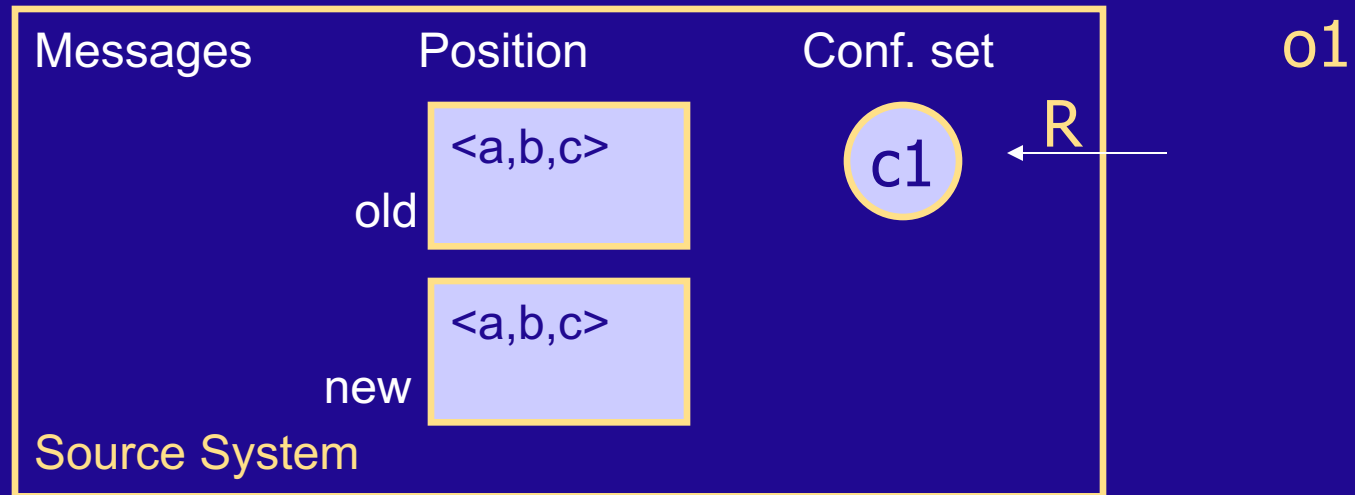
# Transition system: an example

- Linda-like tuple space:

- P: multiset of tuples ( $\underline{t}$ ), initially having only  $\langle a, b, c \rangle$
- Suppose the configuration “rd( $\pi, o$ )”:
- $J(\text{rd}(\pi, o))(\underline{t}, \underline{t}') = \langle g: \exists t \in \underline{t}': \text{mt}(t, \pi), r: \perp, u: \underline{t}', m: o(t) \rangle$

1) The entity “o1” invokes the operation  $\text{rd}(\langle d, \_, \_ \rangle)$

- A registration transition of the configuration  $c1 = \text{rd}(\langle d, \_, \_ \rangle, o1)$

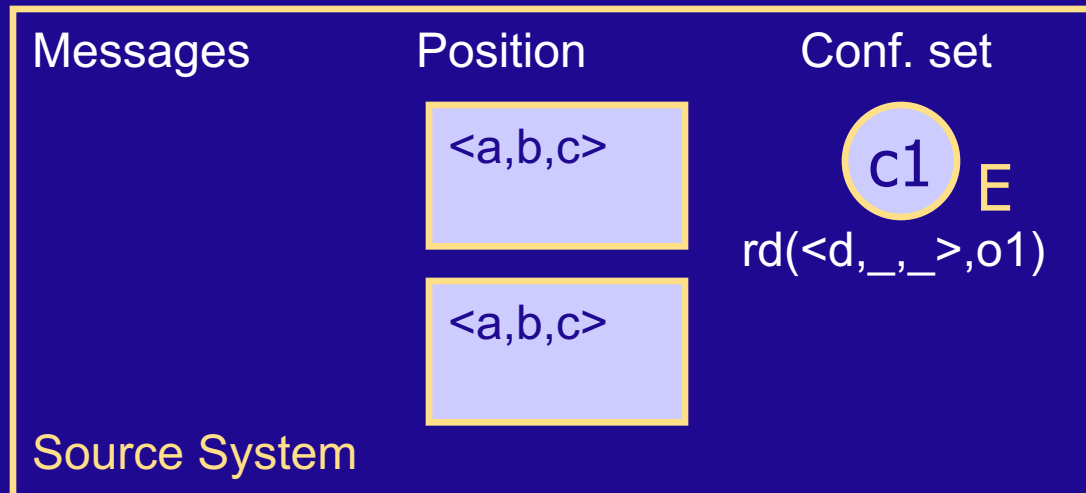


# Transition system: an example

- Linda-like tuple space:
  - P: multiset of tuples ( $\underline{t}$ ), initially having only  $\langle a, b, c \rangle$
  - Suppose the configuration “ $rd(\pi, o)$ ”:
    - $J(rd(\pi, o))(\underline{t}, \underline{t}') = \langle \exists t \in \underline{t}': mt(t, \pi), \perp, \underline{t}', o(t) \rangle$

2) The source evaluates its configuration set:  $\{c1\}$

- Evaluation transition: the guard of  $c1$  does not apply!

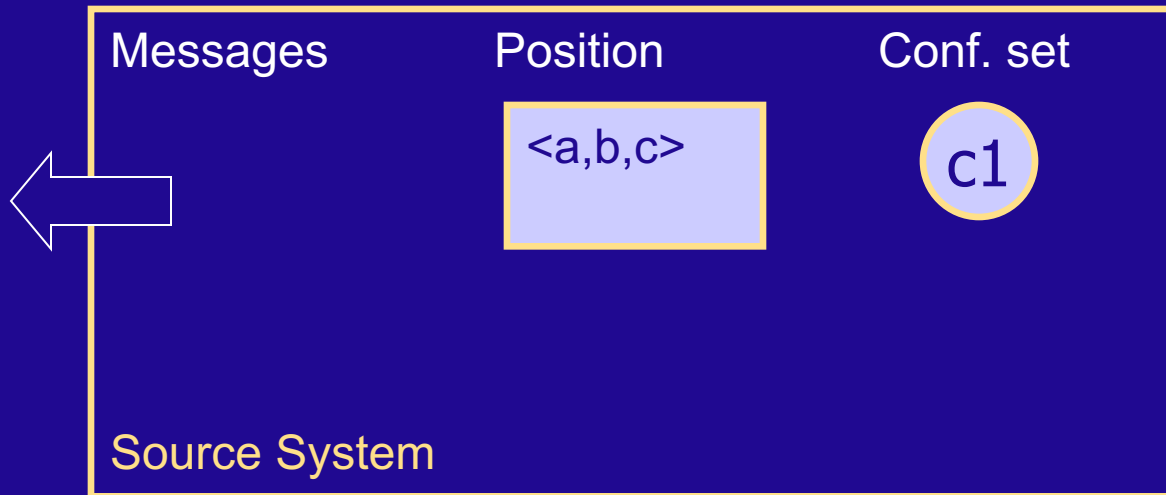


# Transition system: an example

- Linda-like tuple space:
  - P: multiset of tuples ( $\underline{t}$ ), initially having only  $\langle a, b, c \rangle$
  - Suppose the configuration “rd( $\pi, o$ )”:
    - $J(\text{rd}(\pi, o))(\underline{t}, \underline{t}') = \langle \exists t \in \underline{t}': \text{mt}(t, \pi), \perp, \underline{t}', o(t) \rangle$

## 3) The source returns on rest

- Output transition: no messages are sent



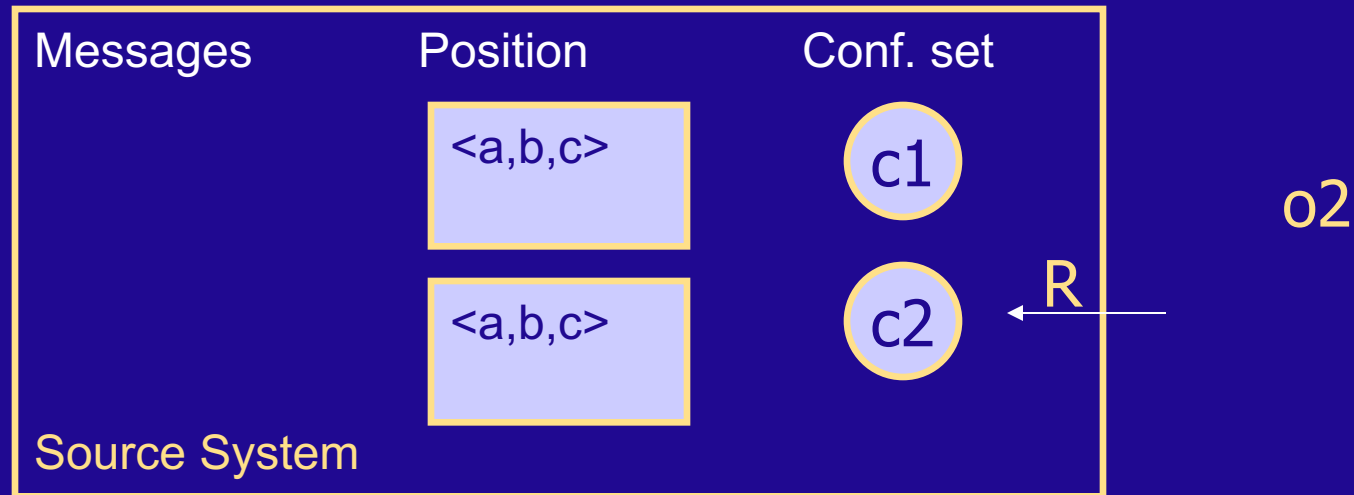
# Transition system: an example

- Linda-like tuple space:

- P: multiset of tuples ( $\underline{t}$ ), initially having only  $\langle a, b, c \rangle$
- Suppose the configuration “rd( $\pi, o$ )”:
  - $J(\text{rd}(\pi, o))(\underline{t}, \underline{t}') = \langle \exists t \in \underline{t}': \text{mt}(t, \pi), \perp, \underline{t}', o(t) \rangle$

4) The entity “o2” invokes the operation  $\text{rd}(\langle a, \_, \_ \rangle)$

- A registration transition of the configuration  $c2 = \text{rd}(\langle a, \_, \_ \rangle, o2)$



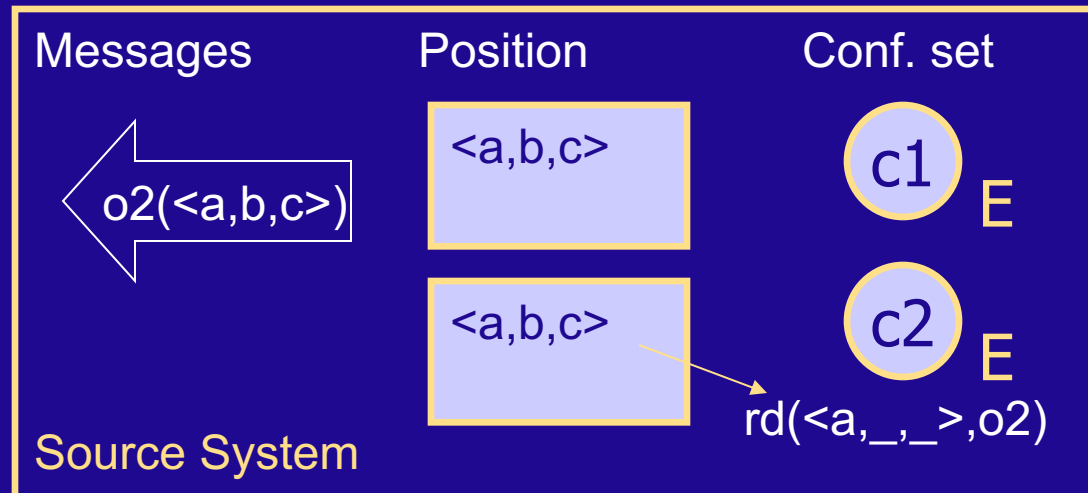
# Transition system: an example

- Linda-like tuple space:

- P: multiset of tuples ( $\underline{t}$ ), initially having only  $\langle a, b, c \rangle$
- Suppose the configuration “rd( $\pi, o$ )”:
- $J(\text{rd}(\pi, o))(\underline{t}, \underline{t}') = \langle \exists t \in \underline{t}': \text{mt}(t, \pi), \perp, \underline{t}', o(t) \rangle$

5) The source evaluates its configuration set:  $\{c1, c2\}$

- Evaluation: c2 applies, it is dropped and produces  $o2(\langle a, b, c \rangle)$

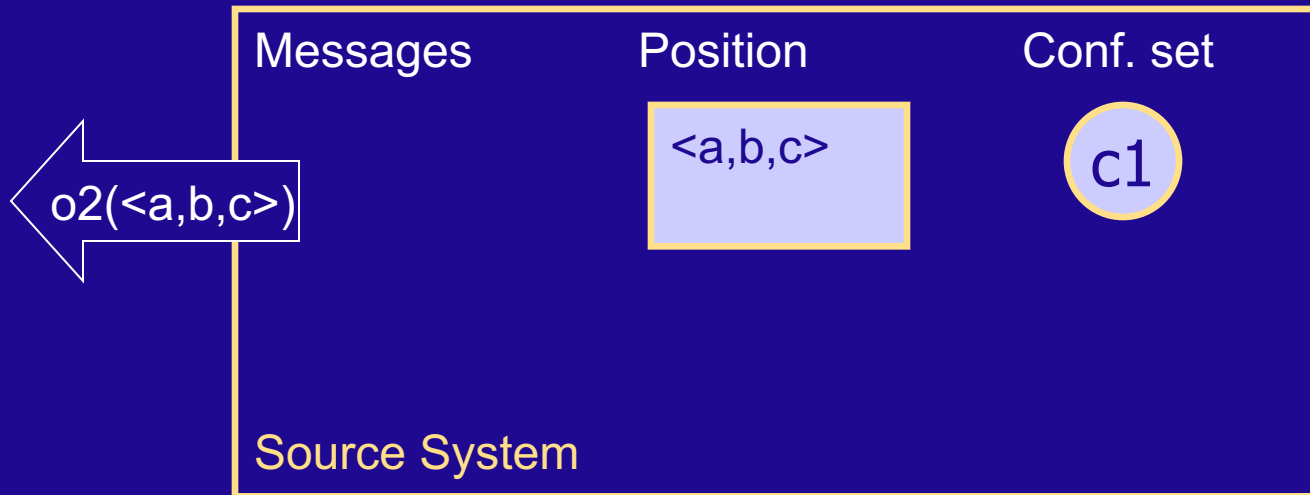


# Transition system: an example

- Linda-like tuple space:
  - P: multiset of tuples ( $\underline{t}$ ), initially having only  $\langle a, b, c \rangle$
  - Suppose the configuration “rd( $\pi, o$ )”:
    - $J(\text{rd}(\pi, o))(\underline{t}, \underline{t}') = \langle \exists t \in \underline{t}': \text{mt}(t, \pi), \perp, \underline{t}', o(t) \rangle$

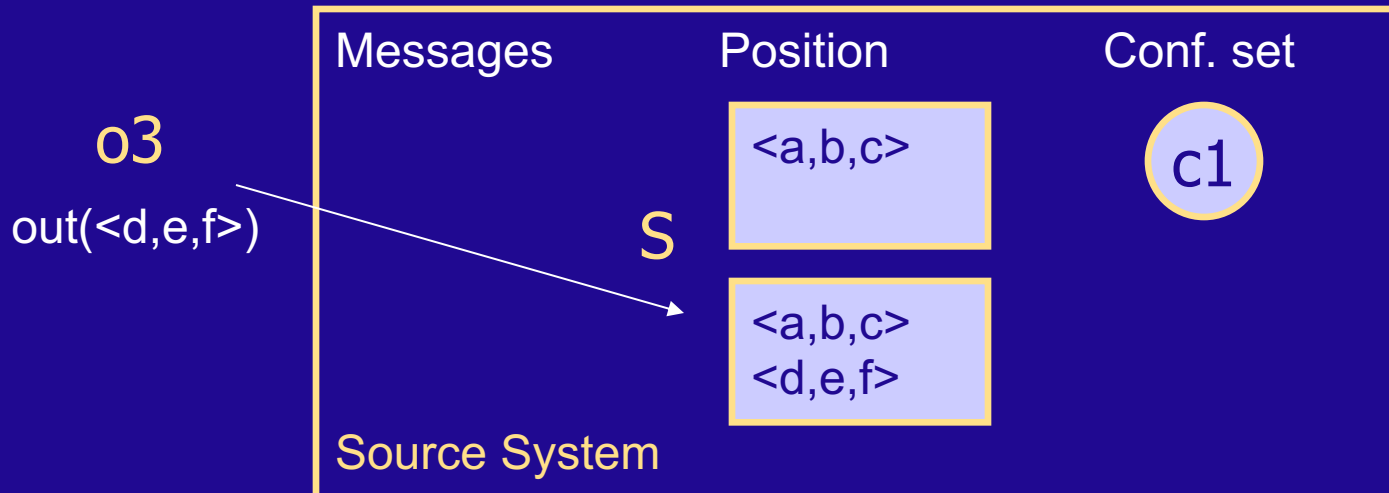
## 6) The source returns on rest

- Output transition: the message  $o2(\langle a, b, c \rangle)$  is shipped



# Transition system: an example

- Linda-like tuple space:
    - P: multiset of tuples ( $\underline{t}$ ), initially having only  $\langle a, b, c \rangle$
    - Suppose the configuration “rd( $\pi, o$ )”:
      - $J(\text{rd}(\pi, o))(\underline{t}, \underline{t}') = \langle \exists t \in \underline{t}': \text{mt}(t, \pi), \perp, \underline{t}', o(t) \rangle$
- 7) An entity “o3” invokes the operation  $\text{out}(\langle d, e, f \rangle)$
- A source transition  $\{\langle a, b, c \rangle\}, \{\langle a, b, c \rangle, \langle d, e, f \rangle\}$



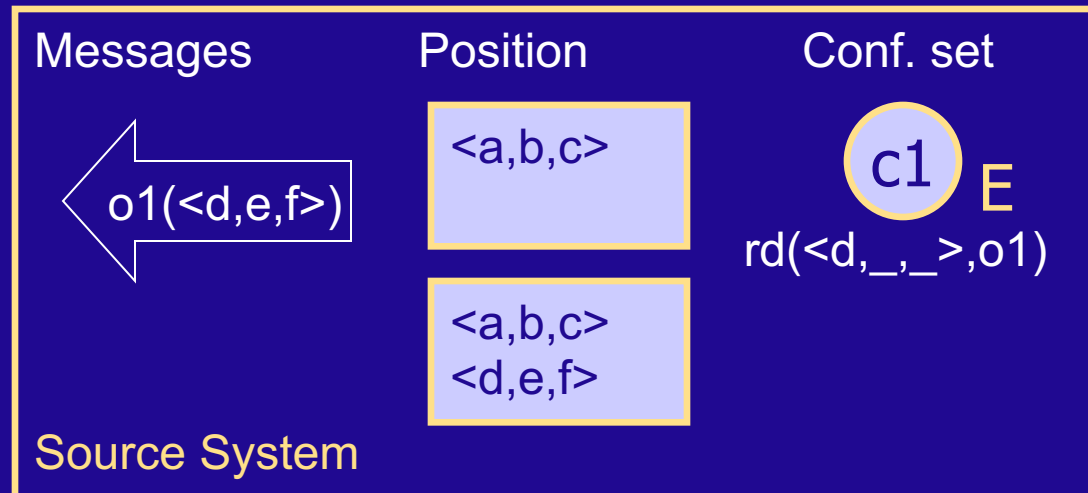
# Transition system: an example

- Linda-like tuple space:

- P: multiset of tuples ( $\underline{t}$ ), initially having only  $\langle a, b, c \rangle$
- Suppose the configuration “ $\text{rd}(\pi, o)$ ”:
  - $J(\text{rd}(\pi, o))(\underline{t}, \underline{t}') = \langle \exists t \in \underline{t}': \text{mt}(t, \pi), \perp, \underline{t}', o(t) \rangle$

8) The source evaluates its configuration set:  $c1 = \text{rd}(\langle d, \_, \_ \rangle, o1)$

- Evaluation:  $c1$  applies, it is dropped and produces  $o1(\langle d, e, f \rangle)$

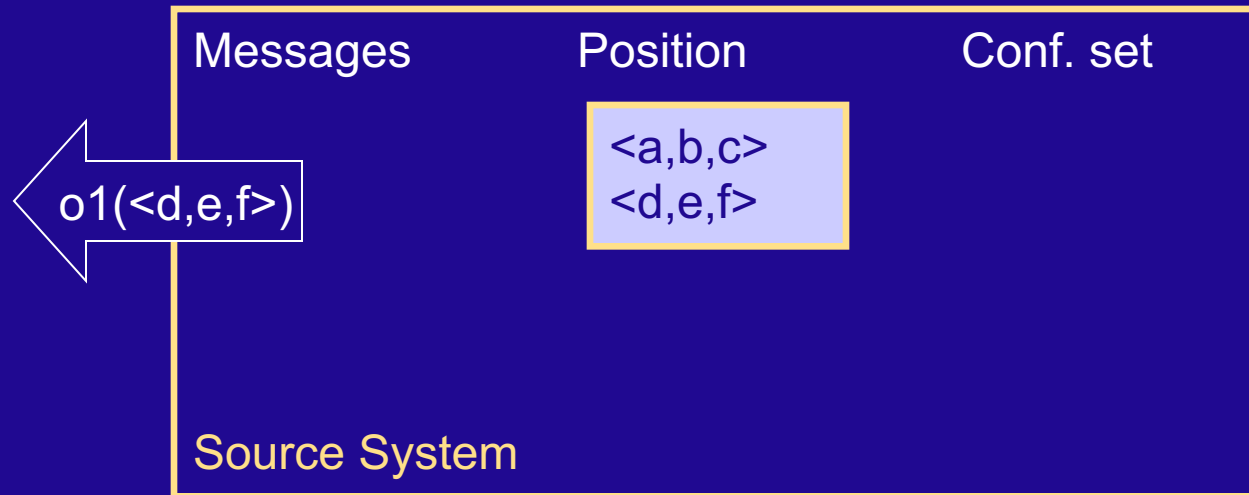


# Transition system: an example

- Linda-like tuple space:
  - P: multiset of tuples ( $\underline{t}$ ), initially having only  $\langle a, b, c \rangle$
  - Suppose the configuration “rd( $\pi, o$ )”:
    - $J(\text{rd}(\pi, o))(\underline{t}, \underline{t}') = \langle \exists t \in \underline{t}': \text{mt}(t, \pi), \perp, \underline{t}', o(t) \rangle$

## 9) The source returns on rest

- Output transition: the message  $o1(\langle d, e, f \rangle)$  is shipped



# Direct manifestation: JavaSpaces

- JavaSpaces:
  - $P$  = multiset of tuples
  - $C = \{\text{read}, \text{readIfExists}, \text{take}, \text{takeIfExists}, \text{notify}\}$
- Operation:  $\text{readIfExists}(\pi) > o$ 
  - $J(\text{readIfExists}(\pi) > o)(p, p') = \langle \text{true}, \perp, p', o(\text{match}(p', \pi)) \rangle$
  - $\Rightarrow$  The pattern of a reactive configuration for a direct manifestation:  
$$J(c)(p, p') = \langle \text{true}, \perp, p', \text{msg} \rangle$$
- Operation:  $\text{notify}(\pi) > o$ 
  - $J(\text{notify}(\pi) > o)(p, p') = \langle p' = t \mid p : \text{mt}(t, \pi), \text{notify}(\pi) > o, p', o(t) \rangle$
  - $\Rightarrow$  is a proactive configuration for a direct manifestation !  
$$J(c)(p, p') = \langle \text{cond}, c, p', \text{msg} \rangle$$

# Internal observation: TuCSoN

- Reaction behaviour of TuCSoN (Omicini&Denti):
  - Tuple Space + Reaction Specification Tuples
  - The reactions alters the behaviour of the source
- In our framework:
  - $J^p$  and  $J^a$ , interpretation for pre-conditions and actions
  - Configuration:  $\text{react}(\text{pre}, \text{act})$
  - $J(\text{react}(\text{pre}, \text{act}))(p, p') =$   
 $\langle J^p(\text{pre}, p, p'), \text{react}(\text{pre}, \text{act}), J^a(\text{pre}, p, p'), \emptyset \rangle$
  - $\Rightarrow$  is a pro-active internal observer!
  - $J(c)(p, p') = \langle \text{cond}, c, p'', \emptyset \rangle$

# A trivial kind of observation: Eval

- Eval in Linda:  $\text{eval}(1+2, 3+4, \dots)$ 
    - it contains the specification of a tuple to be inserted, which should be processed by the tuple space
    - This process doing can be modeled as an internal observer
  - Our model:
    - $\Psi$ : models the processing of eval
    - Configuration:  $\text{eval}(l_1, \dots, l_n)$
    - $J(\text{eval}(l_1, \dots, l_n))(p, p') = \langle \text{true}, \perp, p' \mid \Psi(l_1, \dots, l_n), \emptyset \rangle$
    - $\Rightarrow$  is a reactive internal observer!
- $$J(c)(p, p') = \langle \text{true}, \perp, p'', \emptyset \rangle$$

# Observation in other areas

- Ontology and Formal Framework can be applied to many models and architectures in Computer Science
- Object-Orientation
  - Supposing JavaBeans... consider a bean as a source
    - getXXX  $\Rightarrow$  request for a reactive and direct manifestation
    - addXXX  $\Rightarrow$  request for a pro-active and direct manifestation
- Databases
  - Select query  $\Rightarrow$  request for a reactive and direct manifestation
  - Trigger  $\Rightarrow$  configuring a proactive internal observer
- Agents Architectures
  - In FIPA, many messages are about observation: query-if, request, request-when, request-whenever, subscribe, ...

# Lessons Learned

- Observation occurs in different ways in very heterogeneous models, paradigms, systems,...
- It is interesting to:
  - isolate observation actions from general interactions
  - factorize and abstract observation features of different models
- It is possible to define a common ontology for observation
  - aspects which are seemingly diverse can be factorized
- The formal framework:
  - captures details of the dynamics of a source
  - allows to identify the most common kinds of observation

# Current and Future Work

- We are studying a core version of the framework
  - throws away many details
  - evaluation of one configuration at a time
  - observation behavior by means of two functions
    - Selection: which configuration is to be evaluated
    - Evaluation: rewriting, update, messages
  - separation between:
    - behaviour of a configuration
    - interference between different configurations
- In future work:
  - Dealing with transactions
  - Modeling Control
  - Engineering issues