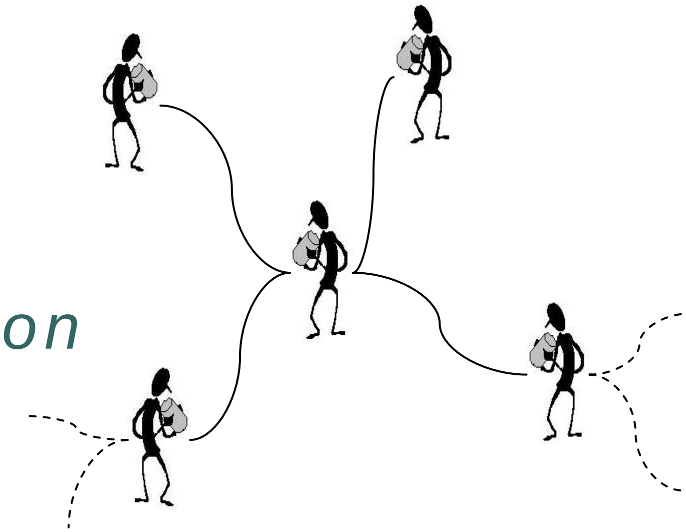
i-dialogue

Modeling Agent Conversation by Streams and Lazy Evaluation



Clement Jonquet & Stefano A. Cerri

LIRMM — Laboratoire d'Informatique de Robotique et de Microélectronique de Montpellier

Université Montpellier II

CNRS — Centre National de la Recherche Scientifique

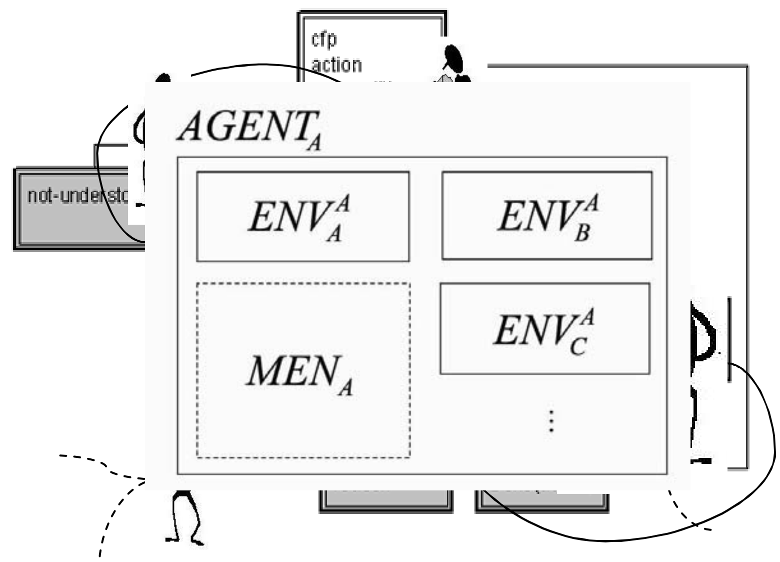
(International Lisp Conference 2005 – Stanford University – June 19-22, 2005)

Context – Interaction modeling

- In DAI and MAS communities: interacting entities
  - interaction + autonomy + intelligence = agents
- To enhance agent's autonomy
  - Communicate without knowing something about the other
  - Managing the entire conversation dynamically
- *I-dialogue* = abstraction of interaction inspired:
  - The *dialogue* abstraction [*O'Donnell, 1985*]
  - The STROBE agent model [*Cerri, 1999; Jonquet, 2004*]

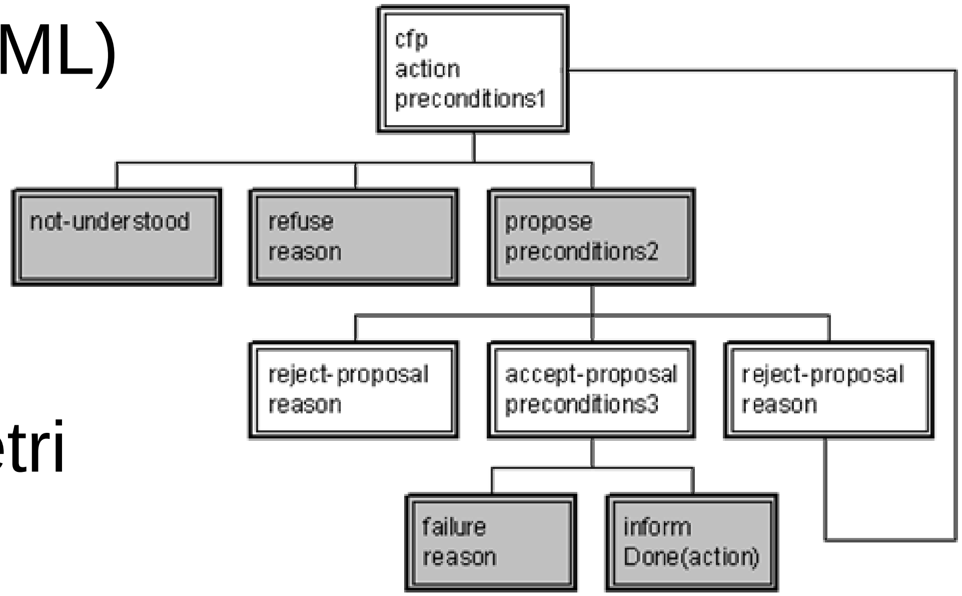
Speech overview

- Agent communication and conversation modeling
- The dialogue abstraction
- The i-dialogue abstraction
- The STROBE model
- Providing services applications
- Conclusion and perspectives



Agent communication

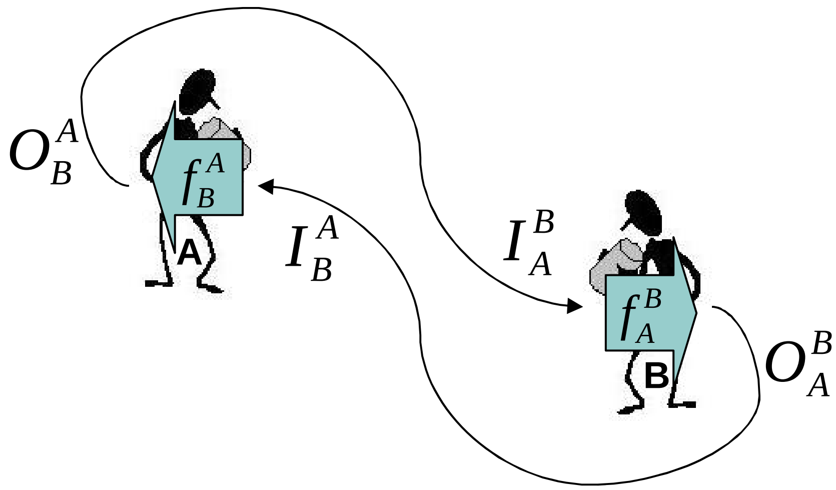
- ACLs (speech act oriented, FIPA, KQML)



- Communication protocols (FSM, Petri Nets)
  - ☺ Semantics
  - ☹ Reduce agent autonomy

The *dialogue* abstraction (1/2)

- Interactive session between 2 agents, which take turns sending messages to each other:



- Each agent computes a new state and a new output from its previous state and the last input it received from the other agent, using its transition function:

$$f_B^A : \left[\alpha_{j+k} \; I_B^A\right] \longrightarrow \left[\alpha_{j+k+1} \; O_B^A\right]$$

$$f_A^B : \left[\beta_k \; I_A^B\right] \longrightarrow \left[\beta_{k+1} \; O_A^B\right]$$

The *dialogue* abstraction (2/2)

- Applicative/Functional programming constructs:
  - Higher order functions
  - Streams [*Abelson and Sussman, 1996*] […]
  - Lazy evaluation [*Landin, 1965*] [*Friedman and Wise, 1976*] […]

- The dialogue function take 4 parameters and returns 3 values:

$$\text{Agent A:} \quad dialogue : \langle I_B^A \ \alpha_j \ f_B^A \ R_A \rangle \longrightarrow (I_B^A \ O_B^A \ val)$$

$$\text{Agent B:} \quad dialogue : \langle I_A^B \ \beta_0 \ f_A^B \ R_B \rangle \longrightarrow (I_A^B \ O_A^B \ val)$$

The *dialogue* function

```
dialogue : ⟨inputs initial-state step-fcn result-fcn⟩ ≡
  letrec
    run ≡ λ ⟨inputs state⟩ .
      let
        (inputs' outputs' state' done') ≡
          stef-fcn : ⟨inputs state⟩
      in
        if done'
          then (inputs' outputs' result-fcn:state')
          else
            let
              (inputs" future-outputs" result") ≡
                run : ⟨inputs' state'⟩
            in
              (inputs"
              append-ll : ⟨outputs' future-outputs"⟩
              result")
  in
    run : ⟨inputs initial-state⟩
```

has 4 parameters

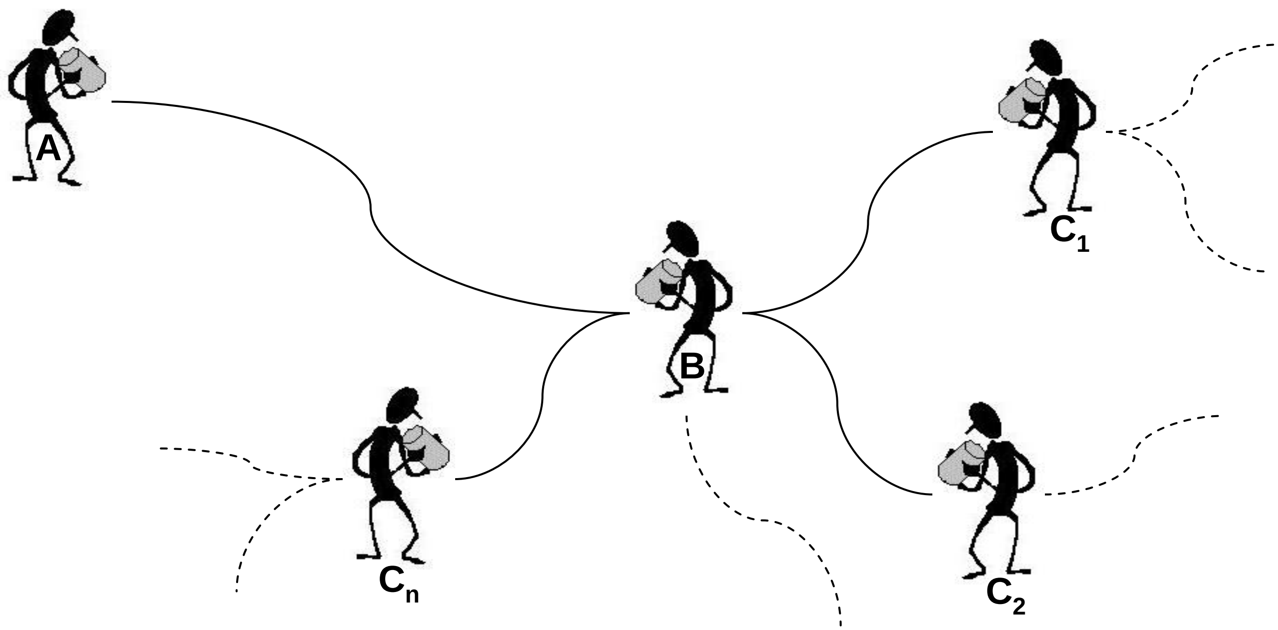
Transition function applied recursively on *inputs* and *state* and produces *outputs*, new *state*, *unused-inputs*, and a *boolean*

returns 3 elements

The *dialogue* abstraction limits

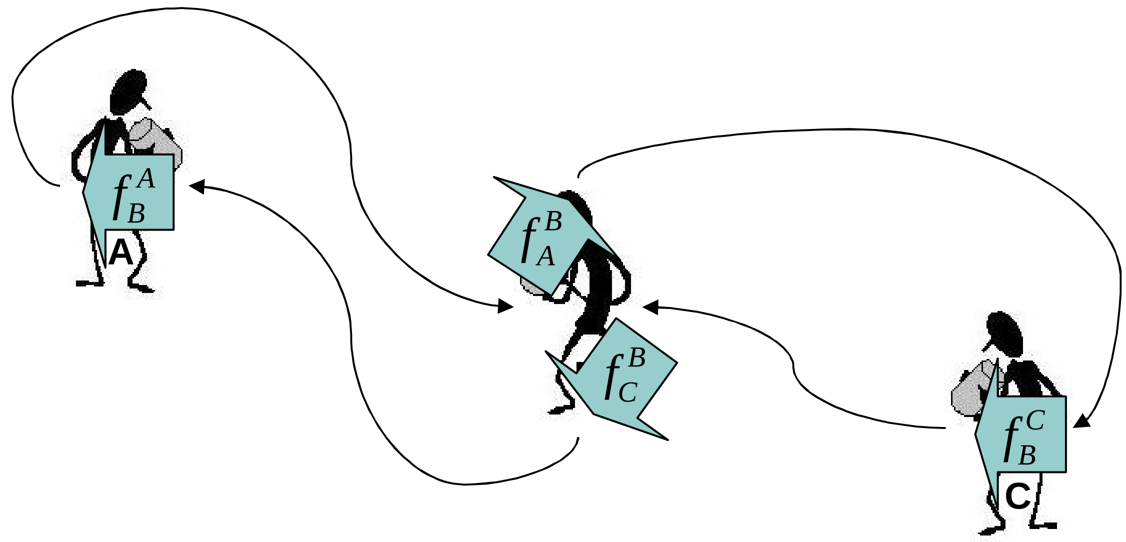
- Distributed systems: more than 2 agents.
- Several *dialogue* (serially or in parallel) do not model conversation among several agents
- Interpretation of one agent inputs produces not the outputs for this agent but another outputs intended to another agent.

The *i-dialogue* abstraction



- Modeling *intertwined-dialogue*
- Conversations between an agent and a group of agents

The 3 agents case



- Agent B should consumes 2 input streams and produces 2 output streams
- Transition functions of B, do not produce respectively an output stream for A and B but the opposite

The *trialogue* function

$$
\begin{array}{l}
\mathit{trialogue} : \langle \mathit{in}_A\ \ \mathit{in}_C\ \ \mathit{init\text{-}s}\ \ \mathit{s\text{-}fcn}_A\ \ \mathit{s\text{-}fcn}_C\ \ \mathit{r\text{-}fcn}\rangle \equiv \\
\quad \textbf{letrec} \\
\quad\quad \mathit{run} \equiv \lambda\ \langle \mathit{in}_A\ \ \mathit{in}_C\ \ \mathit{state}\rangle\ . \\
\quad\quad\quad \textbf{let} \\
\quad\quad\quad\quad (\mathit{in}'_A\ \ \mathit{out}'_C\ \ \mathit{state'}\ \ \mathit{done'}) \equiv \mathit{s\text{-}fcn}_A : \langle \mathit{in}_A\ \ \mathit{state}\rangle \\
\quad\quad\quad \textbf{in} \\
\quad\quad\quad\quad \textbf{if}\ \mathit{done'} \\
\quad\quad\quad\quad\quad \textbf{then}\ (\mathit{in}'_A\ \ \mathit{in}_C\ \ \mathit{null}\ \ o'_C\ \ \mathit{r\text{-}fcn{:}state'}) \\
\quad\quad\quad\quad\quad \textbf{else} \\
\quad\quad\quad\quad\quad\quad \textbf{let} \\
\quad\quad\quad\quad\quad\quad\quad (\mathit{in}'_C\ \ \mathit{out}'_A\ \ \mathit{state''}\ \ \mathit{done''}) \equiv \\
\quad\quad\quad\quad\quad\quad\quad\quad \mathit{s\text{-}fcn}_C : \langle \mathit{in}_C\ \ \mathit{state'}\rangle \\
\quad\quad\quad\quad\quad\quad \textbf{in} \\
\quad\quad\quad\quad\quad\quad\quad \textbf{if}\ \mathit{done''} \\
\quad\quad\quad\quad\quad\quad\quad\quad \textbf{then}\ (\mathit{in}'_A\ \ \mathit{in}'_C\ \ \mathit{out}'_A\ \ \mathit{out}'_C\ \ \mathit{r\text{-}fcn{:}state''}) \\
\quad\quad\quad\quad\quad\quad\quad\quad \textbf{else} \\
\quad\quad\quad\quad\quad\quad\quad\quad\quad \textbf{let} \\
\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad (\mathit{in}''_A\ \ \mathit{in}''_C\ \ \mathit{f\text{-}out}''_A\ \ \mathit{f\text{-}out}''_C\ \ \mathit{result''}) \equiv \\
\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad \mathit{run} : \langle \mathit{in}'_A\ \ \mathit{in}'_C\ \ \mathit{state''}\rangle \\
\quad\quad\quad\quad\quad\quad\quad\quad\quad \textbf{in} \\
\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad (\mathit{in}''_A\ \ \mathit{in}''_C \\
\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad \mathit{append\text{-}ll} : \langle \mathit{out}'_A\ \ \mathit{f\text{-}out}''_A\rangle \\
\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad \mathit{append\text{-}ll} : \langle \mathit{out}'_C\ \ \mathit{f\text{-}out}''_C\rangle \\
\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad \mathit{result''}) \\
\quad \textbf{in} \\
\quad\quad \mathit{run} : \langle \mathit{in}_A\ \ \mathit{in}_C\ \ \mathit{init\text{-}s}\rangle
\end{array}
$$

has 6 parameters

Different transition function applied in the given order on the different *inputs* and *state* and produce different *outputs*, new *states*, and different *unused-inputs*, and *booleans*

returns 5 elements

The *i-dialogue* function

- Generalization of the function trialogue:
  - List of inputs,
  - List of transition functions.
- Classic list recursion !
- The ordering of the elements of the lists corresponds to the semantics
- For agent B in the previous figure:

$$\textit{i-dialogue} : \langle \langle I_A^B I_{C_1}^B \ldots I_{C_n}^B \rangle \; \beta_0 \; \langle f_A^B f_{C_1}^B \ldots f_{C_n}^B \rangle \; R_B \rangle$$

$$\rightarrow (\langle I_A^B I_{C_1}^B \ldots I_{C_n}^B \rangle \; \langle O_A^B O_{C_1}^B \ldots O_{C_n}^B \rangle \; \mathit{val})$$

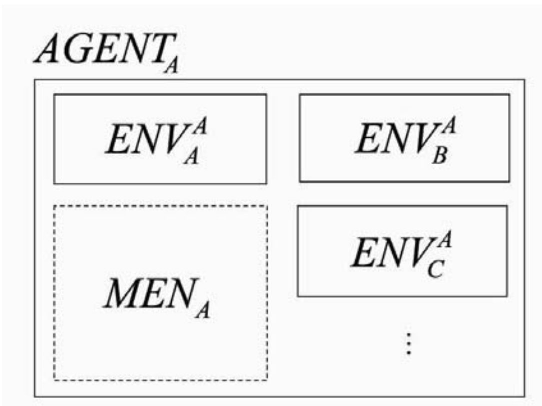
The STROBE model

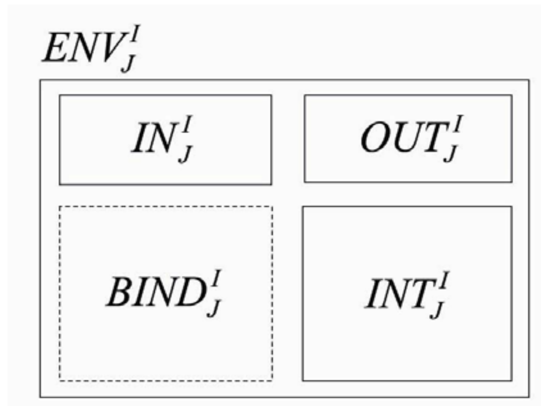
- Agent communication and representation model
- **STR**eams of messages exchanged by agents represented as **OB**jects and interpreted in multiples **E**nvironments
- Scheme specification/implementation

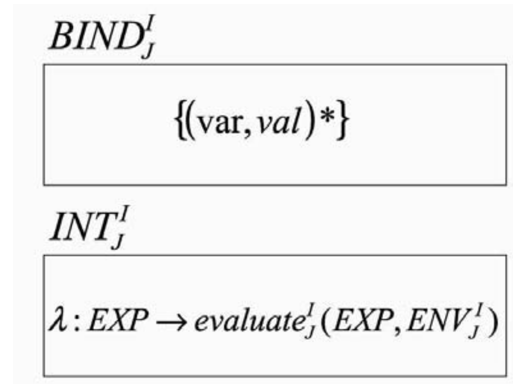
STROBE Agent architecture (1/2)

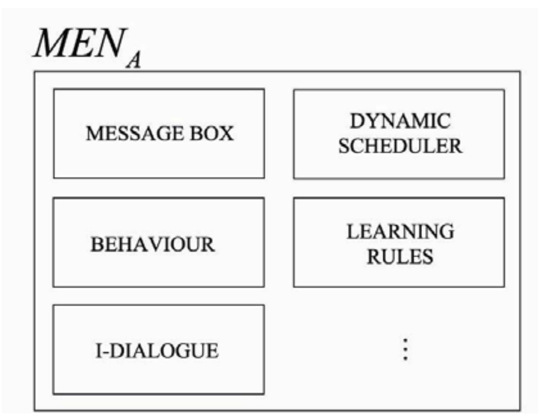
- ENV: Cognitive Environments (as knowledge base and context of evaluation of messages)
- INT: Cognitive Interpreters included in ENV
- Agents as interpreters: map the classical REP loop from FP to REPL
  - i.e: map the context of evaluation `(eval e r)` of Scheme expressions to interpretation of messages

STROBE Agent architecture (2/2)





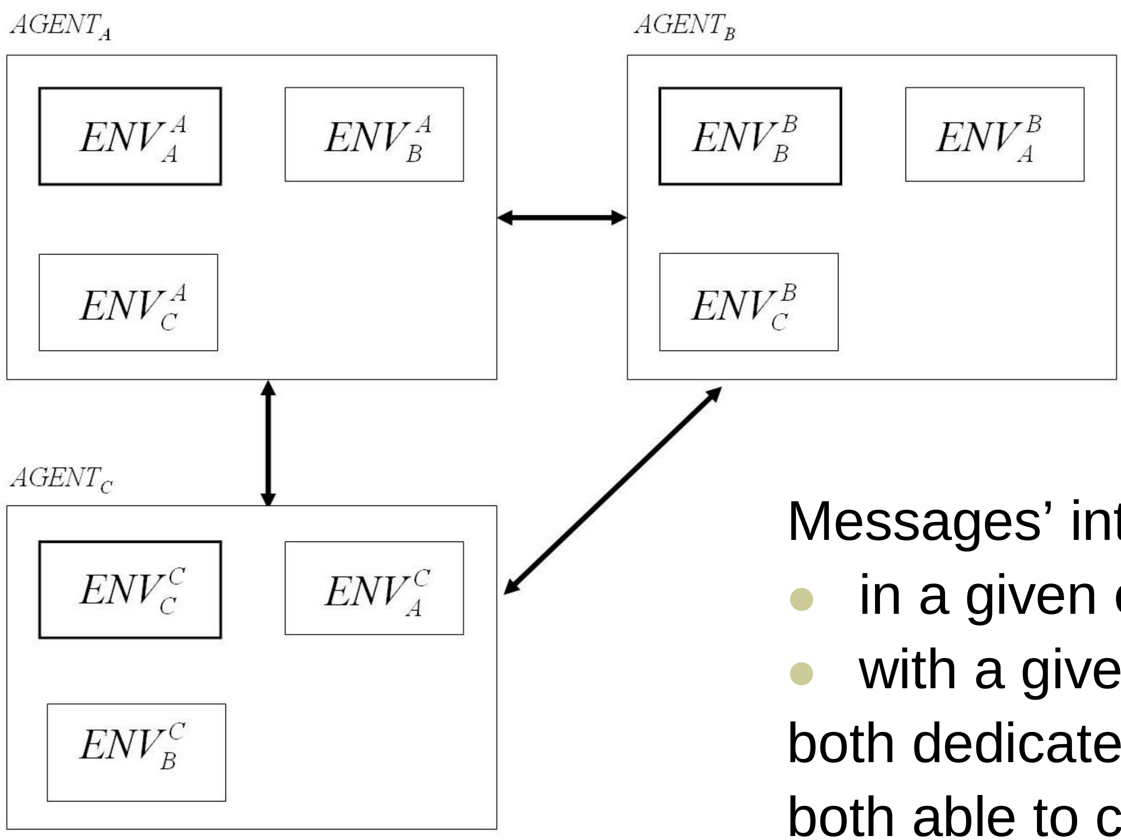




Mental states = agent own objectives, tendencies, behaviour, reasoning rules etc.

$$\text{With, } ITEM_Y^X = \begin{cases} \text{X local item dedicated to Y} & \text{if} \quad X \neq Y \\ \text{X global item} & \text{if} \quad X = Y \end{cases}$$

Message interpretation



Messages’ interpretation is done:

- in a given environment
- with a given interpreter

both dedicated to the interlocutor

both able to change

STROBE / i-dialogue integration

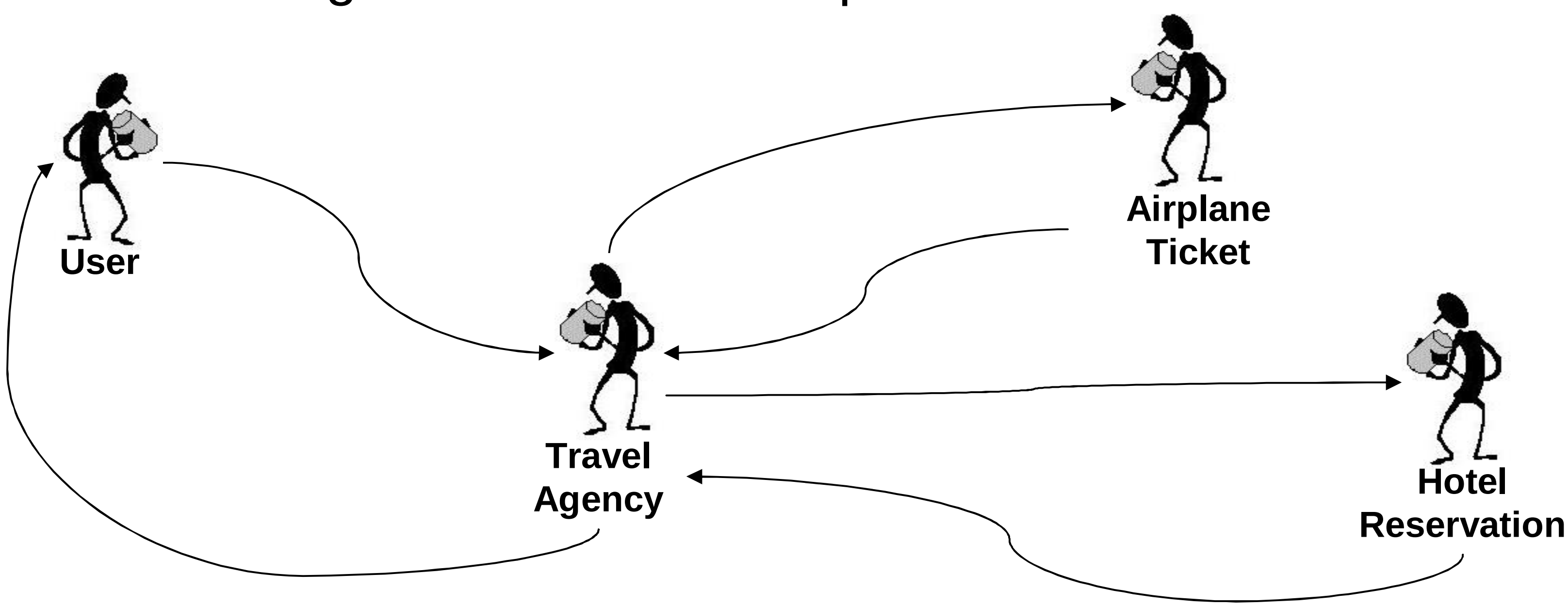
- Seeing the Cognitive Interpreters of STROBE as the transition functions (*step-fcns*) of i-dialogue.

➔ Changing *step-fcn*s dynamically while communicating (i.e. during message interpretation)

Providing service applications

- An agent executing an i-dialogue function provides a service realized by its *stef-fcns*

- i-dialogue models the composition of all the services



Dynamic Service Generation

- Opposed to classical product delivery
  - Buying *ready-to-wear clothes* having *clothes made by a tailor*
- Services constructed on the fly by the provider according to the conversation it has with the user.
  - ➔ Importance of the communication model
- STROBE developed as a toolkit for DSG
  - ➔ Highly dynamic service with on the fly modification of the *step-fcns*

Conclusions and perspectives

- 3 main contributions:
  - To spread the elegant *dialogue* abstraction to more complex situations implying several entities
  - To consider this abstraction for agent communication as it was suggested by STROBE
  - To open a new kind of consideration in service generation
- 2 main advantages:
  - Not reduce agent's autonomy
  - Allows to deal with the entire conversation
- 2 main perspectives:
  - Achieve the in progress integration with the STROBE model
  - Dynamic ordering of the *inputs* and *step-fcns* lists from i-dialogue