i-dialogue

Modeling Agent Conversation by Streams and Lazy Evaluation

Clement Jonquet & Stefano A. Cerri

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

- \textit{l-dialogue} = abstraction of interaction inspired:
  - The \textit{dialogue} abstraction \cite{O'Donnell, 1985}
  - The STROBE agent model \cite{Cerri, 1999; Jonquet, 2004}

Speech overview

- Agent communication and conversation modeling
- The dialogue abstraction
- The 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:

$$f^A_B : \left[ \alpha_{j+k} \quad I^A_B \right] \rightarrow \left[ \alpha_{j+k+1} \quad O^A_B \right]$$

$$f^B_A : \left[ \beta_k \quad I^B_A \right] \rightarrow \left[ \beta_{k+1} \quad O^B_A \right]$$

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 takes 4 parameters and returns 3 values:

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

  Agent B: $\text{dialogue : } \langle I_A^B, \beta_0, f_A^B, R_B \rangle \rightarrow (I_A^B, O_A^B, \text{val})$
The dialogue function

dialogue ≡ \langle\text{inputs initial-state step-fcn result-fcn}\rangle
let rec
run ≡ λ \langle\text{inputs state}\rangle .
let
\langle\text{inputs’ outputs’ state’ done’}\rangle ≡
step-fcn : \langle\text{inputs state}\rangle
in
if done’
then \langle\text{inputs’ outputs’ result-fcn:state’}\rangle
else
let
\langle\text{inputs” future-outputs” result”}\rangle ≡
run : \langle\text{inputs’ state’}\rangle
in
\langle\text{inputs” append-ll : outputs’ future-outputs”}\rangle
result”
in
run : \langle\text{inputs initial-state}\rangle

(has 4 parameters
Transition function applied recursively on \text{inputs} and \text{state} and produces \text{outputs}, new \text{state}, \text{unused-inputs}, and a \text{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 consume 2 input streams and produce 2 output streams.
- Transition functions of B, do not produce respectively an output stream for A and B but the opposite.
The triilogue function

\[
\text{trialogue } \equiv \text{letrec}
\]

\[\begin{align*}
\text{run } & \equiv \lambda \text{ (in}_A \text{ in}_C \text{ state) ;}
\text{let} \\
\text{in} & \text{ if done’ then (in}_A \text{ in}_C \text{ null } \text{ else}
\text{let} \\
\text{in’} & \text{ (in}_A \text{ out}_C \text{ state’ done’) } (\text{in}_A \text{ state})
\text{if done” then (in}_A \text{ in’}_C \text{ out}_A \text{ out}_C \text{ r-fcn:state”})
\text{let} \\
\text{in’’} & \text{ (in}_A \text{ in’}_C \text{ f-out}_A \text{ f-out}_C \text{ result”) } \text{run} : \text{ (in}_A \text{ in}_C \text{ state”)}
\text{in} \\
\text{in”} & \text{ (in}_A’ \text{ in’}_C’ \text{ append-lt : (out}_A’ \text{ f-out}_A’ \text{ append-lt : (out}_C’ \text{ f-out}_C’ \text{ result”)}
\text{run } : \text{ (in}_A \text{ in}_C \text{ init-s)}
\end{align*}\]

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:

\[ i\text{-dialogue} : \langle \langle \langle I_A^B I_{C_1}^B \ldots I_{C_n}^B \rangle \beta_0 \langle f_A^B f_{C_1}^B \ldots f_{C_n}^B \rangle R_B \rangle \]

\[ \rightarrow \langle \langle I_A^B I_{C_1}^B \ldots I_{C_n}^B \rangle \langle O_A^B O_{C_1}^B \ldots O_{C_n}^B \rangle \text{val} \rangle \]
The STROBE model

- Agent communication and representation model

  - **STREAMs of messages exchanged by agents represented as **OBJECTs and interpreted in multiples **ENVironments**

- 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 \((\text{eval } e \ r)\) of Scheme expressions to interpretation of messages
STROBE Agent architecture (2/2)

Agent = set of Cognitive Environments and mental states

Cognitive Environment = set of bindings + an interpreter + 2 input/output streams

A Cognitive Interpreter is an evaluate function

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

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

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-fcns dynamically while communicating (i.e. during message interpretation)
Providing service applications

- An agent executing an i-dialogue function provides a service realized by its $ste-fcns$

- 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-fcn*
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