Performative Patterns for Designing Verifiable ACLs

Mauro Gaspari
Dipartimento di Scienze dell'Informazione
University of Bologna

joint work with Nicola Dragoni

Language design issues
Agent Communication Languages

How to implement performatives starting from the specification of their semantics.
Outline

• The problem:
  – Are performatives “just” messages?
  – How to deal with failures of agents.
• Our solution: Performative patterns.
• Classification of ACL performatives.
• Discussion
  – Main features of our approach
  – ACL semantics and implementation
Yes --> to send performative as contents of asynchronous message passing primitives

[JADE PROGRAMMER'S GUIDE]:
“...An agent willing to send a message should create a new ACLMessage object, fill its attributes with appropriate values, and finally call the method Agent.send(). Likewise, an agent willing to receive a message should call receive() or blockingReceive() ...”
Observation

Following this approach all the performatives are transmitted in the same way.

More precisely: the **concurrent semantics** is the same for all of them. Informally: *the performative (ACLmessage object) is transmitted in a queue associated to the recipient agent*
**declarative semantics**

**insert(B, A, p)**

Informally: agent A asks agent B to insert a proposition p in its Knowledge Base

**KQML semantics (Labrou):**

......
Post(A): know(A, bel(B,p))
Post(B): bel(B, p)
The KQML insert performative is not “just” a message.

This depends on the “declarative” semantics of KQML.
Advantages of the KQML approach

Expressive power!

The insert performative is more powerful than a single message!

For example: after executing an insert successfully we can infer that both A and B believe p ($KS(p)$)

However this intended meaning cannot be implemented with a single performative following the JADE approach: the implementation becomes more complex
Is JADE correct?

yes!

*JADE is conform to the FIPA semantics*

FIPA assumes **asynchronous** communication actions

FIPA ACL's semantic model is based on the **rational effect** of each performative

*what the sender agent hopes to obtain by sending the message*

Thus there no assumptions on the state of the recipient agent that the sender agent should know
Limitations of JADE (FIPA)

It is not possible to infer properties such as $KS(p)$ in JADE after a single communication action: at least two communications are needed.

A:  
send(inform(B,A,p))
blockingReceive()

B:  
send(inform(A,B,bel(B,p)))

an extra speech act mechanism is needed: a blockingReceive()
What we have learned

there is a tradeoff between the expressive power of the ACL semantics and the complexity of the implementation

where complexity in this context does not mean computational complexity but concurrent complexity

However, although the JADE implementation is more complex it is possible to infer a property like $KS(p)$ in both the approaches
WEB based MAS: failures (crashes) of agents are possible

Suppose that agent B crashes,

is the previous JADE implementation of insert correct?
Dealing with agent crashes in JADE

First a timeout \( t \) is needed in the receive primitive:

\[
\text{send(inform(B,A,p))} \\
\text{blockingReceive(t)}
\]

Then the result of the receive should be considered:

\[
\text{send(inform(B,A,p))} \\
\text{result = blockingReceive(t)} \\
\text{if result = -1 then \{timeout - retry?\}} \\
\text{else \{KS(p) holds\}}
\]

However, \textbf{it is not clear how to implement the retry policy:}

1) \textit{should the agent send the inform performative again?}
2) \textit{or should the agent execute another blockingReceive waiting for some more time?}
Problems

The number of extra speech act primitives increases!

Is this high level human like communication?

The programmer needs to solve many low level programming issues, for example:
- establish the correct timeout
- establish the correct action when a retry is needed
Our claims

• There is a relationship between the declarative semantics of a performative and how a retry operation should be realized.
  • In most cases an agent needs to be aware of the fact that its partner has crashed.
  • Some (low level) crash handling operations can be hidden in the failure handling mechanisms (for example a criteria to establish when an agent is considered crashed).
  • Some (knowledge level) crash handling actions should be “attached” to performatives by the programmer (for example, finding another partner).

Definitely, performatives are not “just” messages!
Our approach

We also assume **asynchronous communication**.

We deal with **crash failures** of agents.

We extend the abstract syntax of performatives adding optional failure and success continuations to the communication act:

\[ \text{insert}(B,A,p)[\text{on\_ack} + \text{on\_fail}] \]

We have recognized that some performatives may share the same interaction model.

We have defined a set of **performative patterns** to characterize **common interaction models**.
performative(A,B,p)[success_continuation + failure_continuation]
Do action with ack

\[
\text{insert}(B,A,p)[\text{on\_ack}(P1) + \text{on\_fail}(P2)]
\]

\[
\text{perf\_name}(B,A,p)[\text{on\_ack}(P1) + \text{on\_fail}(P2)]
\]
Request and Answer

- **Performatives Templates:**
  - `request_name(B, A, K)[on_answer(P1) + on_fail(P2)]`
  - `answer_name(A, B, K)`
Request and answers

- **Performative Templates:**
  * request_name(B, A, K)[on answer(P1 ) + on fail(P2 )]
  * answer_name(A, B, K)
  * end_name(A, B, K)
Request to everybody

- **Performative Templates:**
  * request_name(A, K)[on_answer(P1) + on_fail(P2)]
  * answer_name(A, B, K)

- **Predicates:**
  * all-answers(K): returns true if all the replies about K have been received
## Classification of Standard Performatives

<table>
<thead>
<tr>
<th>Pattern</th>
<th>KQML</th>
<th>FIPA ACL</th>
<th>FT-ACL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assert</td>
<td>inform, agree, reject proposal</td>
<td>inform</td>
<td>inform</td>
</tr>
<tr>
<td></td>
<td>advertise, tell, untell, error</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>sorry, deny</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Assert with Ack</td>
<td>insert, uninsert, delete-one, delete-all, undelete, achieve, unachieve</td>
<td>inform[... + ...]</td>
<td></td>
</tr>
<tr>
<td>Do Action with Ack</td>
<td>insert</td>
<td></td>
<td>insert</td>
</tr>
<tr>
<td>Request and Answer</td>
<td>ask-if, ask-all, ask-one</td>
<td>request, query if, query ref</td>
<td>ask-one</td>
</tr>
<tr>
<td></td>
<td>{tell, untell, deny}</td>
<td>{inform, agree, cancel, failure}</td>
<td>{tell}</td>
</tr>
<tr>
<td></td>
<td>propose</td>
<td>propose</td>
<td></td>
</tr>
<tr>
<td>Request and Answers</td>
<td>stream-all, subscribe</td>
<td>subscribe</td>
<td></td>
</tr>
<tr>
<td></td>
<td>{tell, untell, deny, eos}</td>
<td>{inform, cancel}</td>
<td></td>
</tr>
<tr>
<td>Request to Everybody</td>
<td>ask-everybody</td>
<td></td>
<td>{tell}</td>
</tr>
</tbody>
</table>
Conclusions

- **Performatives are not “just” messages!**
- Our language FT-ACL:
  - Concurrency aspects and Knowledge-Level programming
  - It includes most of the performative patterns.
  - It deals with crash failures of agents.
  - It provides anonymous requests for knowledge.
- **Semantics:**
  - Weak declarative semantics
  - Strong concurrent semantics
Performative patterns can be used to implement performatives starting from the specification of a declarative semantics.

But not all the semantics can be implemented in a distributed system.

There is a tradeoff between the expressive power of an ACL (semantics) and the concurrent complexity (implementation).
References

**Knowledge-Level programming and concurrency:**

**Definition and Semantics of FT-ACL:**

**Implementation of FT-ACL:**

**Knowledge-Level Programming in FT-ACL:**