

I.Vlahavas, P.Kefalas, N.Basileiadis, F.Kokkoras, I.Sakellariou
Texnhth Nohmosunh (B' Ekdosh),
Ekdoseis Gartaganh, Thessaloniki, 2005
<http://aibook.csd.auth.gr>

SCENARIO:

In order for an agent to be able to communicate via KQML messages, it needs:

- To check whether the incoming KQML messages are syntactically correct.
- To understand the message and do the appropriate action.
- To construct a new message as part of the action (if required).

For this laboratory session, we chose to implement the following format of messages:

```
(<performative>
  :sender <word>
  :receiver <word>
  :in-reply-to <word>
  :reply-with <word>
  :language <word>
  :content <expression>)
```

and the following subset of KQML performatives:

Performative	Meaning (S: sender, R: receiver)	Response Required
ask-if	<i>S wants to know if the :content is in R's Knowledge Base</i>	Yes
ask-one	<i>S wants one of R's instantiations of the :content that is true of R</i>	Yes
ask-all	<i>S wants all of R's instantiations of the :content that is true of R</i>	Yes
tell	<i>The sentence in :content is in S's Knowledge Base</i>	-
advertise	<i>S wants R to know that S can and will process a message like one in :content</i>	Yes
sorry	<i>S understands R's message but cannot provide a response</i>	-

Notes:

- The **sorry** performative does not have **:language** and **:content**.
- The fields of the message is in the strict order they appear.
- **<word>** is any sequence of characters (excluding blank)
- **<expression>** is either any string included in " " or another KQML message.

PREPARATION:

An agent is provided which has an interface as described above and also:

- Knows its own id (its name is **petros**).
- Knows only the languages Prolog and KQML.
- Can read a message from a file and respond by outputting the new message on the screen.

Note: You may assume that the message is read from the network and the new message is put in the network (but we would need a router plus some other things to do this).

You are provide with the following files:

- **parsekqml.pl**: the parser of the KQML messagen, written in simple DCG rules
- **readsentence.pl**: reads the KQML message character by character
- **respondkqml.pl** : takes the appropriate action according to message
- **lab3agent.pl**: the agent **petros**
- **msg[1-5].txt**: Five different incoming messages to be processed by the agent

Study the Prolog code provided, especially the parts concerned with:

- The grammar of the KQML messages (`parsekqml.pl`),
- The method of responding to KQML messages (`respondkqml.pl`),

Look at the five different messages (in `msg[1-5].txt` files). Before you try anything in Prolog, try to find out what the replies to those messages will be, and why.

Consult the file `lab3agent.pl` which will automatically load the rest of the files (that is, as long as you saved them with the same names as listed above).

Try the query:

```
?- incoming('msg1.txt').
```

in order to see the processing of the 1st message. Try other queries with the rest of the messages.

PRACTICAL WORK:

- Make the appropriate changes to accommodate two more performatives:

Performative	Meaning (S: sender, R: receiver)	Response Required
<code>stream-all</code>	<i>Multiple reponse version of <code>ask-all</code></i>	Yes
<code>eos</code>	<i>The end-of-stream marker to a mutiple-response</i>	-

Hint: For multiple responses you need a `do_requested_action/2` which returns a list of messages instead of a single message.

- Make the appropriate changes so that the respond message is written in a file. The message should be save under `<name>.txt` where `<name>` is the `<id>` (`:in-reply-to`) of the message.
- Assuming that each agent has its own directory where its messages are held, transform the above so that the message is written in the directory of the receiver agent, e.g. if the message receiver is agent `steffen` and the message id number is `petros-8` then the message should be saved as `steffen/petros-8.txt`

QUESTIONS:

- Think of how the performative `forward` can be implemented, i.e. *S wants R to forward the message in `:content`*. This performative needs two more keywords `:from` (the origin of the performative in `:content`) and `:to` (the final destination of the performative in `:content`).
- Think of how the performative `broadcast` can be implemented, i.e. *S wants R to send a message to all agents that R knows*.
- At the moment the keywords of the message are in strict order. Think of how you can rewrite the grammar in order to accept KQML messages where the keywords are stated in any order.
- At the moment, the `:content` filler can be either a string included in " " or a KQML message. However, the latter should sonly be valid if the performative is `advertise`, `forward`, `broadcast` etc. Think of how you can rewrite the grammar in order to comply with this requirement.
- At the moment, the `:content` filler can be KQML message, even thought the `:language` may be stated as `prolog`. Think of ways to make the two fillers consistent.