france telecom

www.francetelecom.com/

(Unrestricted)

Research & Development

Overview of the JADE Semantics Add-on demo

Authors: Vincent PAUTRET, Vincent LOUIS

Version: 1.0

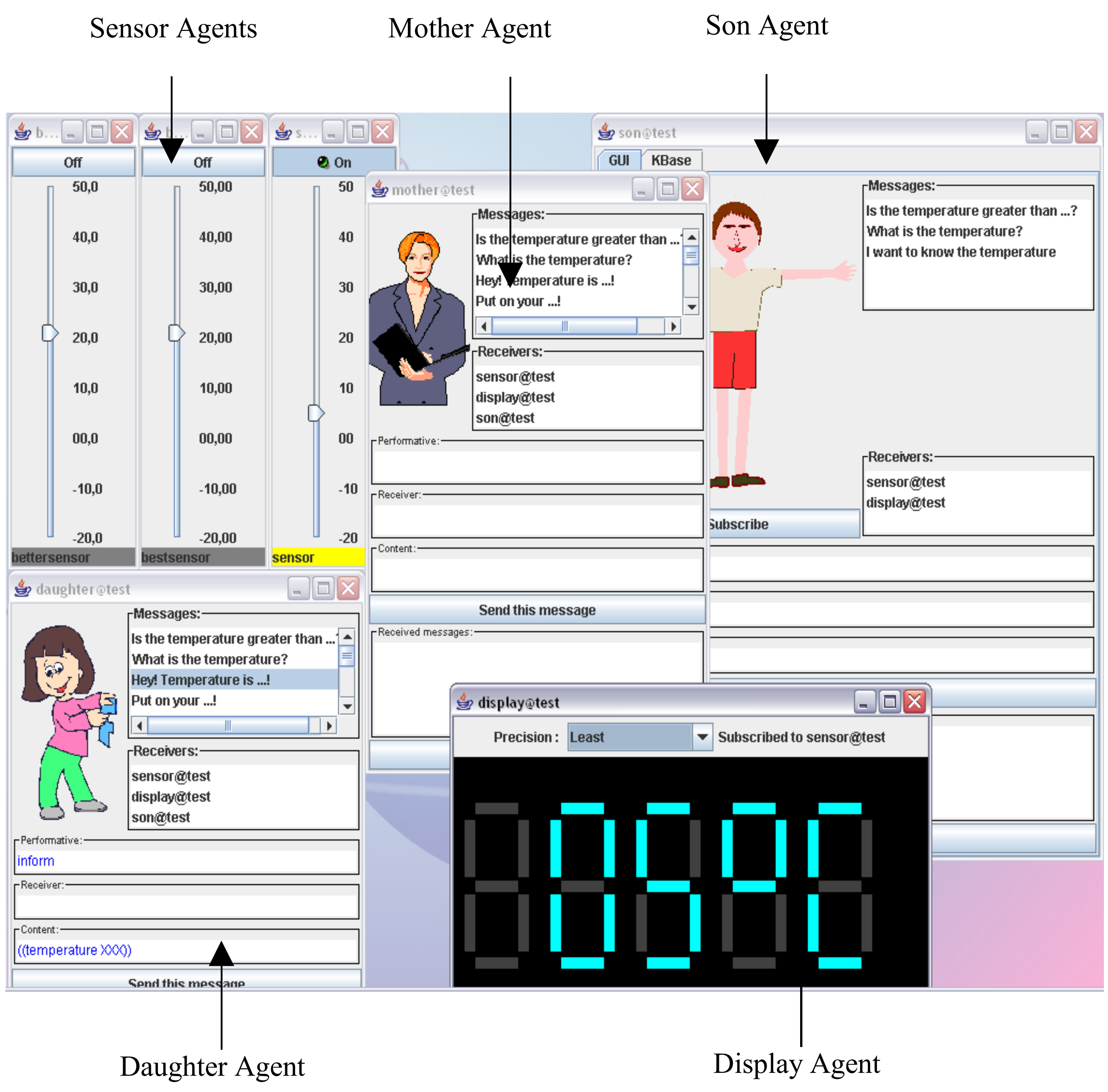
France Telecom
Research & Development

2, av. Pierre Marzin
22307 Lannion Cedex — France
Phone: +33 296 051 111

The new JADE Semantics Add-on is illustrated on a very simple “toy” application related to a temperature management domain.

1 Description of the agents

1.1 Overall description



mother@test — Messages: Is the temperature greater than ...? / What is the temperature? / Hey! Temperature is ...! / Put on your ...! — Receivers: sensor@test / display@test / son@test

son@test — Messages: Is the temperature greater than ...? / What is the temperature? / I want to know the temperature — Receivers: sensor@test / display@test

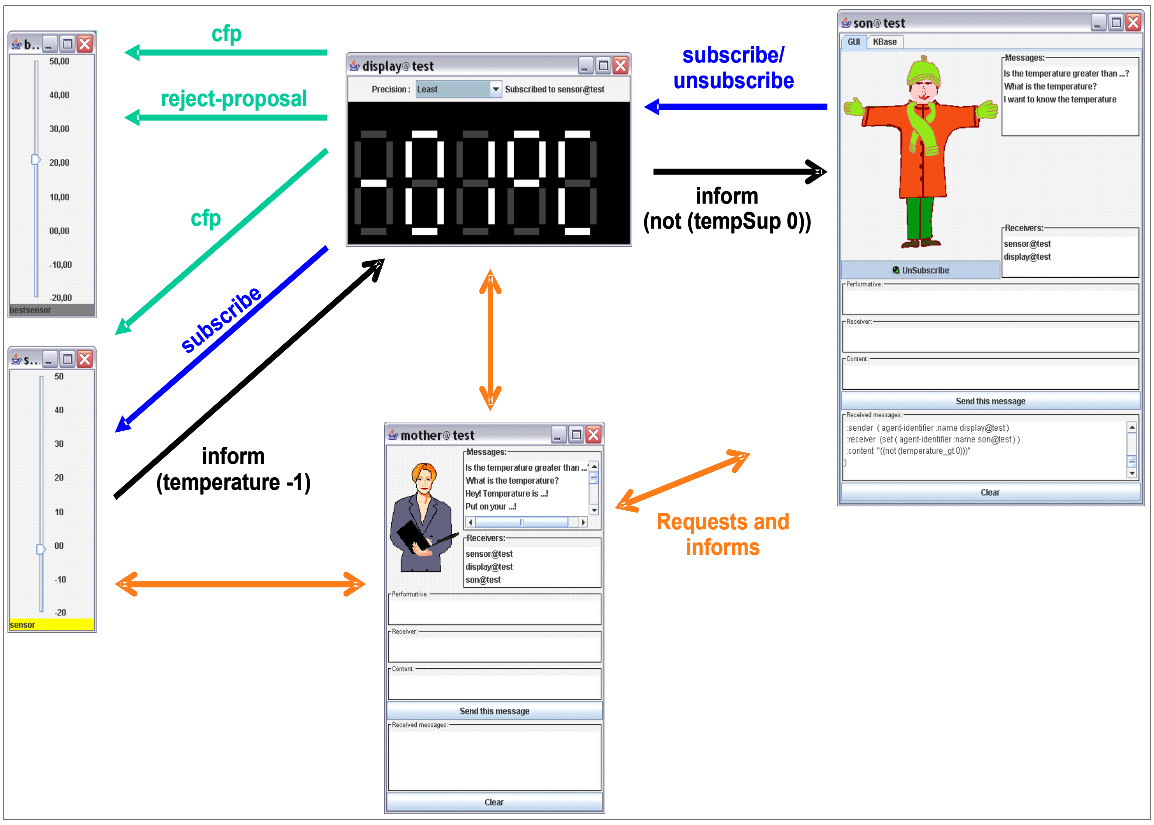
daughter@test — Messages: Is the temperature greater than ...? / What is the temperature? / Hey! Temperature is ...! / Put on your ...! — Receivers: sensor@test / display@test / son@test — Performative: inform — Content: ((temperature XXX))

display@test — Precision : Least — Subscribed to sensor@test

Figure 1 - General view

The demonstration is run by the 'runDemo.bat' or 'runDemo.sh' scripts, located in the 'demo' subdirectory (these files can be easily adapted to other system configurations). It shows seven Jade agents (see Figure 1), plus one additional one that has no graphic interface (DF Agent, see below). Five of them are developed using the JADE Semantics Add-on (called “semantic agents”), the two others are implemented as classical Jade agents:

- Sensors (3 semantic agents): each one represents a temperature sensor that other agents, such as the display, may query in order to get some information about the temperature. Each one gives the temperature with a different precision. The GUI of the sensors provides a slider that enables to vary the temperature. An on/off button enables to register/deregister each sensor in the the DF Agent (semantic agent) knowledge base;
- Display (semantic agent): it represents a thermometer able to give some information about temperature to other agents. Its GUI displays the temperature it currently knows. A menu enables to choose another sensor agent as information source (by sending a CFP to each sensor) with respect to the desired precision;
- Son Agent (semantic agent): it represents a young boy who puts on or takes off its clothes according to the temperature. Its GUI displays the ACL messages it receives, and enables to send ACL messages to the display or the sensor in order to interact with them. A picture shows the current dressing of the son (coat, pullover, trousers). A button enables to subscribe or unsubscribe to some temperature information from the display. A tab enables to see what the content of the Knowledge Base of the son agent is.
- Mother Agent (classical Jade agent): it represents the mother of the son. Its GUI enables to send ACL messages to the sensors, display and son agents and displays the messages received by the mother.
- Daughter Agent (classical Jade agent): it represents the daughter of the mother agent (and so the sister of the son agent). The same GUI as the one of the mother enables to send ACL messages and displays received messages.
- DF Agent (semantic agent): it represents a DF Agent (like a Jade classical one, but with semantic interaction enabled) that stores the active sensor agents. It does not have any GUI.



Subscribed to sensor@test

Messages:
Is the temperature greater than ...?
What is the temperature?
I want to know the temperature

Receivers:
sensor@test
display@test

Received messages:

```
:sender  ( agent-identifier :name display@test )
:receiver  (set ( agent-identifier :name son@test ) )
:content  "((not (temperature_gt 0)))"
)
```

Messages:
Is the temperature greater than ...?
What is the temperature?
Hey! Temperature is ...!
Put on your ...!

Receivers:
sensor@test
display@test
son@test

Figure 2 - Interaction between agents

1.2 Specific behaviours

Mother and **Daughter Agents** are classical Jade agents. They have no special behaviour. They only provide useful GUIs to send ACL messages to semantic agents (i.e. the sensors, the display or the son agent) and to see the replied messages. ACL messages to send can be typed from scratch or picked from a list of predefined messages (easier to type!). .
For example, with the mother GUI, you can ask the display for the temperature, you can inform the son of a temperature value, etc.

Sensors, Display, and Son Agents are semantic agents: they semantically interpret all the ACL messages they receive and react to their meaning. In addition to this generic behaviour, they have been developed with the following domain-specific features:

- they interpret the `temperature` predicate as a predicate with a unique value (e.g. `(temperature 10)` and `(temperature 15)` cannot be true at the same time),
- they interpret the `temperature_gt` predicate as: `(temerature_gt X)` is true if there is an `Y` such as `(temperature Y)` is true and `X` is greater than `Y`.

More specifically, the **son agent** interprets the terms `(PUT-ON :clothing X)` and `(TAKE-OFF :clothing X)` as the ontological actions of putting on and taking off the item of clothing `X`. Recognized values for `X` are the SL words `coat`, `trousers` and `pullover`. It also interprets the SL predicate `(wearing X Y)` as agent `X` is wearing the item of clothing `Y`.
The following specific behaviour[1] has been set for the son agent:

- it puts on his trousers as soon as temperature is less that 20 degrees, and take them off otherwise;
- it puts on his pullover as soon as temperature is less than 15 degrees, and takes it off otherwise;
- it puts on his coat as soon as temperature is less than 10 degrees, and takes it off otherwise.
- it puts on his cap as soon as temperature is less than 0 degree, and takes it off otherwise.

The son agent interprets the term `(WAIT :time X)` as the ontological action of waiting a while (the duration `X` is set in milliseconds).
The son agent has also been developed so that it always believes what it is told (i.e. it adopts beliefs of any agent), and it obeys only to queries from the mother agent (i.e. it adopts intentions of only the mother agent).

Each **sensor agent** can register/deregister itself in the DF Agent's knowledge base (using the button).

Finally, the **display agent**, by using the menu, can subscribe to a sensor. First, it sends a Query-Ref to the DF Agent to know the registered sensors. Then it sends a Call-for-Proposal to each sensor. Each sensor sends a proposal (through a Propose message). The display agent then chooses the appropriate sensor according to the required precision and sends a Subscribe to it (while sending Reject-Proposal messages to the other ones).

1.3 Agents settings and parameters

The GUI of the mother, daughter and son agents can be set by a text file (mother.txt, daughter.txt

1 We mean here a behaviour in the ordinary sense of the word, not a Jade behaviour.

and son.txt). This file defines some pre-typed messages and receivers that can be used in the GUI to build messages to send. The text file must have one line per receiver or per message to define.
The format of a receiver definition is:

`receiver# <jade_agent_name>` (e.g. mother@test)

The format of a message definition is:

`message# <label> # <performative> <content>`

The `<label>` field can be freely filled, it is used to represent the message in the list of messages displayed on the GUI.

The `<performative>` field must be filled with any FIPA performative (case insensitive), e.g. `inform`.

The `<content>` field must be filled with an SL content.

Additionally, for the mother and daughter agents only, the picture can be set by the following definition line:

`image#<image_file>`

The `<image_file>` must be filled with the name (without its path and without its extension) of a GIF file located in the "demo" directory.

Finally, here are the parameters expected for each agent in its jade command invocation:

- Sensor agent (demo.SensorAgent class): 2 mandatory parameters respectivly equal to the precision of the sensor, and the name of the DF Agent;
- Display agent (demo.DisplayAgent class): 1 mandatory parameter equal to the name of the DF Agent,
- Son agent (demo.ManAgent class): 3 mandatory parameters respectively equal to the GUI setup text file, the name of the display agent and the name of the mother agent, plus 1 optional parameter equal to 'showkb' (shows the "KBase" tab if mentionned)
- Mother and daughter agents (demo.DemoAgent class): 1 mandatory parameter equal to the GUI setup text file.

2 A short scenario

This section describes a typical scenario that illustrates most of the interesting properties of the semantic agents. It follows the interaction relationships shown on Figure 2.

At the beginning, each sensor (sensor, bettersensor, and bestsensor) registers to the DF Agent. When clicking on a precision in the list of the display's GUI, the display sends a CFP each registered sensor and then subscribes to the one whose precision matches the chosen criterion, so that this sensor will inform the display about the temperature value each time the temperature changes (by moving the slider on the GUI of the sensor). This behaviour of the sensor fully results from the generic interpretation of the `Subscribe` performative (there is to specific code in the sensor agent to manage it). It can be directly observed on the display when moving up or down the sensor slider. It can also be observed at the FIPA-ACL level by spying the sensor and the thermometer agents with the Jade Sniffer.

At this moment, the son agent has no knowledge about the temperature and his initial clothing is empty. This can be observed by the picture of the son GUI. It can also be observed at the internal knowledge base level by clicking on the "KBase" tab, which displays the current knowledge base content (expressed in FIPA-SL) of the agent. Each displayed fact `F` must be interpreted as "the agent

believes that F is true"[1].
In order to observe the behaviour of the son agent putting on or taking off clothing items (as described in the previous section), you can first move the sensor slider down to 17 degrees and then make the son request the display about the temperature (by choosing the appropriate receiver in the son GUI, choosing the "What is the temperature?" message and clicking on the "Send this message" button). The son agent then sends a Query-Ref message to the display and the display replies with an Inform message that you can read at the bottom of the son GUI. As a consequence, the son agent updates his belief about the temperature value, and then puts on his trousers, which in turn updates his beliefs about his clothing (see the son picture and the "KBase" tab).
The same behaviour would have been observed if the mother agent had sent an inform message about a temperature value to the son (by choosing the "Hey! Temperature is ...!" message in the mother GUI).

In order to observe a more dynamic behaviour of the son on his clothing, you can click on the "Subscribe" button of the son GUI. This button makes the son agent send a list of Request-whenver messages to the display, so that the display will inform the son as soon as temperature is above or below 21, 16 and 11 degrees. The clothing of the son agent then automatically follows the temperature slider location, along the intervals described in the section 1.2.
Note that this subscription is different from the one of the display agent with respect to the sensor agent, because here, the display is not requested to give the exact temperature value to the son. This means that the semantic son agent is able to determine when to put on or to take off a clothing item by reasoning both on the temperature and the temperature_gt predicates. This reasoning feature can also be observed by making the mother agent (for example) request the temperature agent (this works as well with all other semantic agents, including the son agent) with various queries about the exact or relative value of the temperature (try the "Is the temperature greater than ...?" and the "What is the temperature?" messages in the list of the mother GUI).
When clicking again on the "Unsubscribe" button of the son GUI, the son agent sends a list of Inform messages to the thermometer informing that it does not require the previous subscriptions any more. Then the display does not send any more inform messages to the son when the temperature goes through any of the previous thresholds.

The son agent also naturally (as a result of its generic semantic interpretation mechanism) perform the actions other agents request it to do (generally by sending to it Request messages). For example, if the mother agent request the son to put on his coat (by choosing the "Put on your ...!" message in the mother GUI and replacing "XXX" with "coat" in the content field), the son agent executes this action (you can see the result on the son picture, as well as in his KBase content), whatever the temperature can be. You get the same result when the mother expresses an equivalent request with an Inform message ("I want you to put on your ...!" message in the mother GUI), the content of which stating the mother has the intention that the son performs the PUT-ON action.
A little more subtle is the mother requesting the son to query the thermometer about the temperature ("Look at the thermometer!" message in the mother GUI). There are two possible cases:

- If the son already knows the temperature (i.e. if a fact (temperature XXX) belongs to its KBase), then he cannot send a Query-Ref message to the display, because a feasibility precondition of this performative is that the sender must not know the value of the queried

1 If a fact F does not appear in the Kbase, this means that the agent neither believes it nor believes its contrary (not F). It is a state different from the state where the agent believes (not F) (in this case, (not F) belongs to the Kbase).

object. Consequently, when the mother requests him to send the `Query-Ref` message, he informs his mother that it is not possible.

- If the son does not know the temperature (i.e. no fact `(temperature XXX)` belongs to his KBase), then the feasibility precondition of the `Query-Ref` message is satisfied and he sends this message to the display. As a consequence, he updates his beliefs with a new temperature value and may also update his clothing depending on this value.

In order to test these two cases, you can use the "remove" button of the "KBase" tab on the son GUI to artificially modify the KBase content of the son agent.

Semantic agents are also able to perform actions with respect to a rational effect, without any explicit mention of a particular action. For example, if the mother agent sends an `Inform` message to the son agent stating she has the intention to know what is his dressing ("I want to know your dressing" message on the mother GUI), then the son agent will perform an `Inform` message towards the mother mentioning all the clothing items he currently wears. Indeed, the intention of the mother exactly fits with the rational effect of an `Inform-Ref` performative, the son agents then tries to perform it[1].
The same behaviour can be experimented by sending an `Inform` message the content of which is an intention of the mother that the son wears a given clothing item ("I'd like you wear ..." message on the mother GUI). In this case the intention content fits the effect of the ontological `PUT-ON` action, so that you can see the son agent putting on the requested clothing item.
Try the other examples of queries listed in the mother GUI, and observe the results! In particular, the "You want to wear your ..." message from the daughter to the son shows an example of inconsistent (although well formed) content. You can also freely edit the performative, receiver and content fields before sending messages (and so send customized messages). The only constraint is that the content must be a FIPA-SL content expression.

Finally, an other interesting feature of semantic agents is that the belief transfer and the intention transfer steps of their semantic interpretation mechanism can be set up individually for each agent. For example, the son agent has been developed so that he always applies belief transfers (i.e. he comes to believe everything from everyone) and he applies intention transfer only for intentions of his mother (i.e. he comes to intend everything from his mother only and nothing from other agents). This can be observed by making the mother request the son to put on his coat ("Put on your ...!" message in the mother GUI) and by making the daughter request the same (the same message can be found in the daughter GUI). In the first case, the son agent obeys, whereas he ignores the daughter request in the second case. However, if the daughter sends to the son an inform message the content of which is an intention of the mother to do an action, then the son will adopt this intention as a belief (unconditional belief transfer) and then adopt the intention of the mother as an intention of its own (intention transfer from mother only), and so he will perform the requested action. This can be observed by choosing the "Mum would like you put on your ...!" message on the daughter GUI. This example shows the degree of expressivity provided by the FIPA-ACL language.
Similarly, the display agent has been developed so that he applies belief transfers only from the sensor if the content is about temperature, and in any case otherwise. Consequently, if the mother sends to him an `Inform` message with a temperature value ("Hey! Temperature is ...!" message in the mother GUI), the display ignores it and does not update his belief about the temperature (and so does not affect the temperature display). However, if the mother sends an `Inform` message the

1 Note that the actual performance of an `Inform-Ref` message is an instantiated `Inform` message, since the `Inform-Ref` performative denotes a "macro-action" (see the FIPA-ACL specifications).

content of which is `(foo)`, then the thermometer adopts this belief (this can be easily checked by sending to him a `Query-If` message the content of which is `(foo)`). In the same way, the sensor agent has been developed so that he never applies belief transfer dealing with temperature features from any agent.

Of course, it is still possible to send messages to the mother and the daughter agents, however, they will never provide any reaction since they are not semantic agents. Instead their GUI will display all received messages and enables to send any FIPA-ACL message to any other agent.

3 Summary of domain-specific features

For this demonstration, all semantic agents interpret two predicates:

- `(temperature X)`, where `X` is a numeric value, meaning the current temperature value is `X`,
- `(temperature_gt X)`, where `X` is a numeric value, meaning the current temperature value is (strictly) greater than `X`,

In the meanwhile, all semantic agents manage in their knowledge base some obvious constraints underlying these predicates (see the 'SingleNumValueDefinition.java' file). For example, if an agent believe both `(temperature X)` and `(temperature Y)`, then `X` is equal to `Y`; if an agent believe `(temperature_gt X)`, then he believes `(temperature_gt Y)` for any `Y` lower than `X`; etc.

Additionally, the son agent interprets the following predicates:

- `(wearing X Y)`, where `X` is a term denoting an agent (typically a functional term such as `(agent-identifier :name ...)`), and `Y` is a word constant denoting a clothing item (recognized values are `cap`, `coat`, `pullover` and `trousers`), meaning that the `X` agent currently wears the `Y` clothing item.

In the meanwhile, the son agent interprets the following domain-specific actions:

- `(PUT-ON :clothing X)`, where `X` is a word constant denoting a clothing item (the same as in the wearing predicate), meaning that the agent puts on a clothing item. This action is defined in a semantic way by a feasibility precondition equal to `(not (wearing A X))` and a postcondition equal to `(wearing A X)` (where `A` is the agent of the action).
- `(TAKE-OFF :clothing X)`, where `X` is a word constant denoting a clothing item (the same as in the wearing predicate), meaning that the agent takes off a clothing item. This action is defined in a semantic way by a feasibility precondition equal to `(wearing A X)` and a postcondition equal to `(not (wearing A X))` (where `A` is the agent of the action).
- `(WAIT :time X)`, where `X` is an integer constant denoting a number of milliseconds, meaning that the agent waits a specified time. This action is defined in a semantic way by a feasibility precondition equal to `true` and a postcondition equal to `true`.

The defined semantic features enable the son agent to soundly (and automatically, without any specific code) update his knowledge base when he performs the corresponding action. They also enable the son agent to select the corresponding action when he adopts an intention dealing with the wearing predicate (see for example the “I'd like you wear your ...” message on the mother GUI).