JADE Semantics Add-on

July 29th, 2005, Utrecht, AAMAS'05

Vincent Louis, Thierry Martinez

France Telecom Research & Development

Outline

- **Principles**
  - JADE Semantics Add-on & JADE Semantic Agent (JSA)?
  - Components of a JSA
  - The semantic interpretation behavior
  - What does a JSA look like ?
- **General mechanisms**
  - SL expressions handling
  - Handling Inform, Query and Request acts
  - Handling Query and Request protocols
- **Demonstration**
- **Coding a JADE Semantic Agent**
  - Setting up the knowledge base
  - Setting up the action table
  - Setting up the standard customization

Principles

Principles

- JADE Semantics Add-on & JADE Semantic Agent (JSA)?
- Components of a JSA
- The semantic interpretation behavior
- What does a JSA look like ?

General mechanisms

- SL expressions handling
- Handling Inform, Query and Request acts
- Handling Query and Request protocols

Demonstration

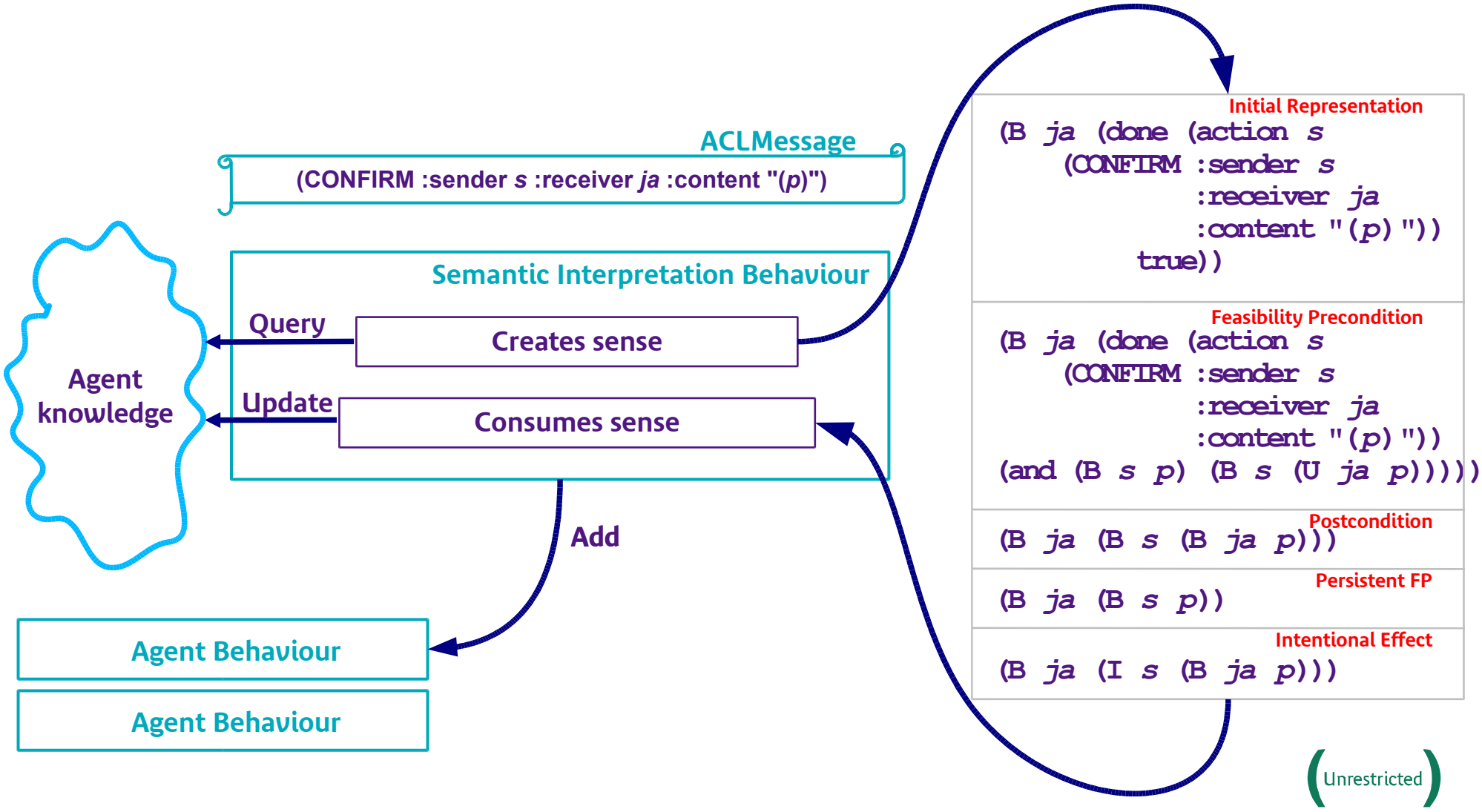
Coding a JADE Semantic Agent

- Setting up the knowledge base
- Setting up the action table
- Setting up the standard customization

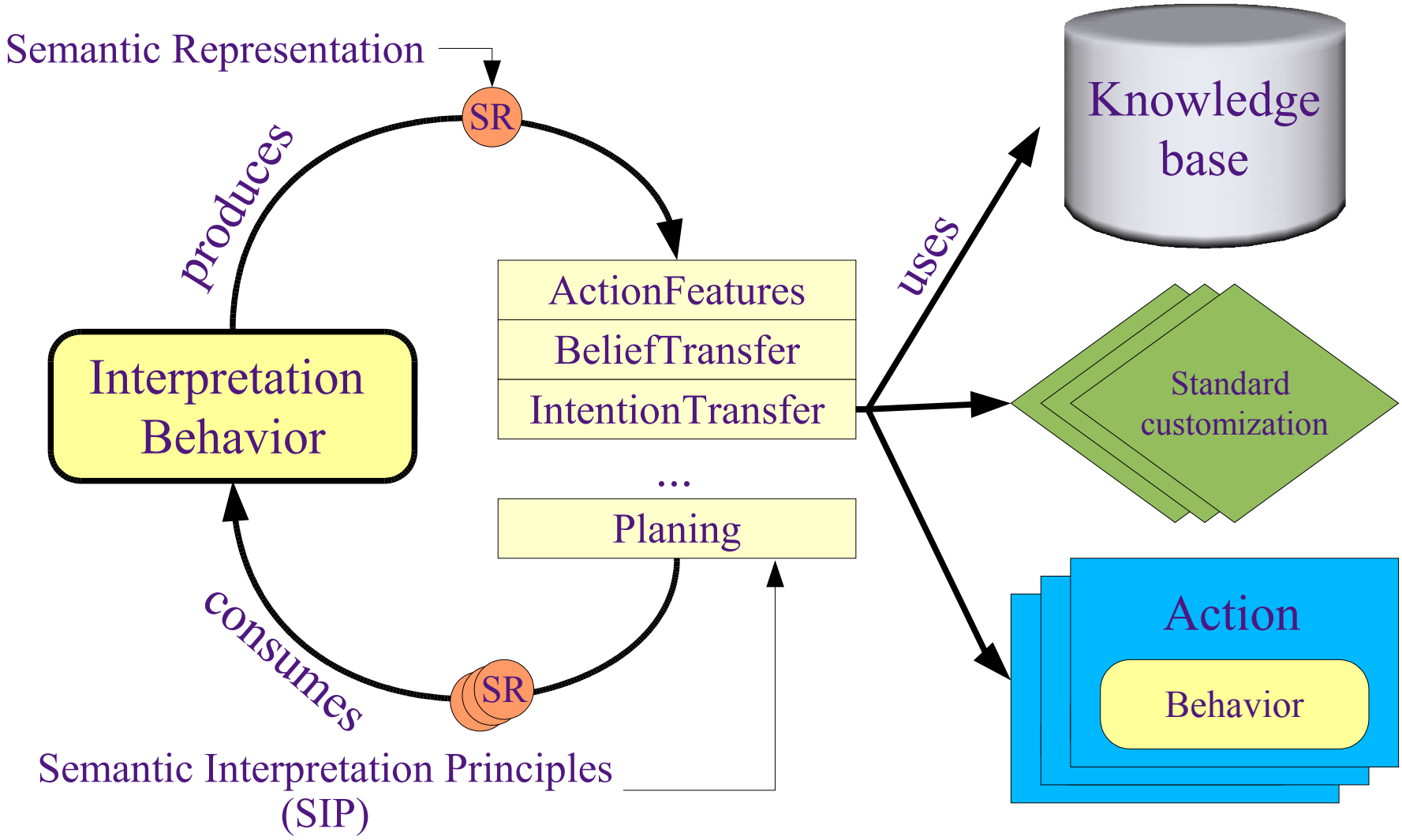
What is the JADE Semantics Add-on?

- A framework based upon JADE to interpret the meaning of the exchanged speech acts, according to their formal semantics as specified by FIPA-ACL
- A framework to make agents more flexible, in order to better interact in open environments
- A framework to simplify the coding of JADE agents

What is a JADE Semantic Agent (JSA)?



The main components of a JSA



Interpretation algorithm

- **Implemented by the SemanticInterpreterBehaviour**
- **Applied to each event (received message or internal event)**

```
For all SR in the list do
        if the SR is false then break;
        Try to apply all the SIPs to the current SR;
        Add the new produced SR to the list;
done
```

- **The interpretation finishes when:**
  - The SR list becomes empty
  - No SIPs can be furhter applied → the remaining SRs are asserted
  - A false SR is produced → a NOT-UNDERSTOOD is sent

The simplest JSA

```
public class TheSimplestJSA
    extends SemanticAgentBase
{
    // Empty
}
```

A more general template

```
public class MyJSA extends SemanticAgentBase {
    class MySematicCapabilities extends SemanticCapabilities {
       protected void setupSemanticActions() {...}
       protected void setupKbase() {...}
       protected void setupStandardCustomization() {...}
       ...
    }
    public MyJSA() {
       semanticCapabilities = new MySematicCapabilities();
    }
    public void setup() {
       super.setup();
       ...
    }
}
```

General mechanisms

- Principles
  - JADE Semantics Add-on & JADE Semantic Agent (JSA)?
  - Components of a JSA
  - The semantic interpretation behavior
  - What does a JSA look like ?
- **General mechanisms**
  - **SL expressions handling**
  - **Handling Inform, Query and Request acts**
  - **Handling Query and Request protocols**
- Demonstration
- Coding a JADE Semantic Agent
  - Setting up the knowledge base
  - Setting up the action table
  - Setting up the standard customization

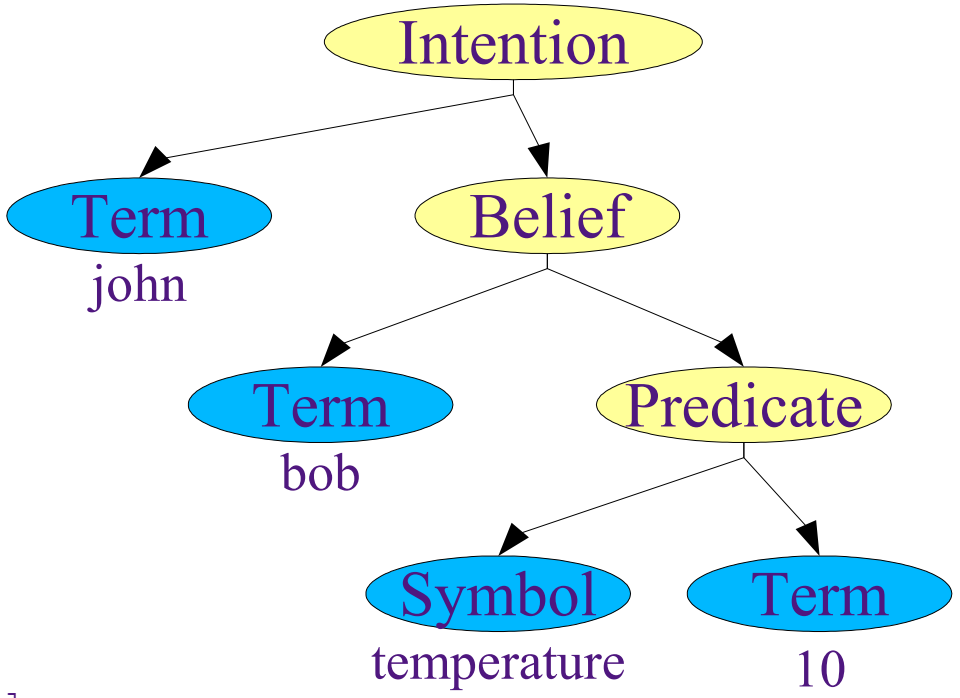
SL expressions and patterns (1)

```
FORMULA ::= ATOMIC_FORMULA
        | UNARY_LOGICAL_FORMULA
        | MODAL_LOGIC_FORMULA
        | ACTION_FORMULA
        | QUANTIFIED_FORMULA
        | BINARY_LOGICAL_FORMULA
        | meta_formula_reference;

ATOMIC_FORMULA ::= proposition_symbol
                 | result
                 | predicate
                 | true
                 | false
                 | equals;

MODAL_LOGIC_FORMULA ::= belief
                      |  uncertainty
                      |  intention
                      |  persistent_goal;
```

(I john (B bob (temperature 10)))



SL expressions and patterns (2)

Creating a pattern

```
Formula fPattern = SLPatternManip.fromFormula("(B ??agent ??phi)");
```

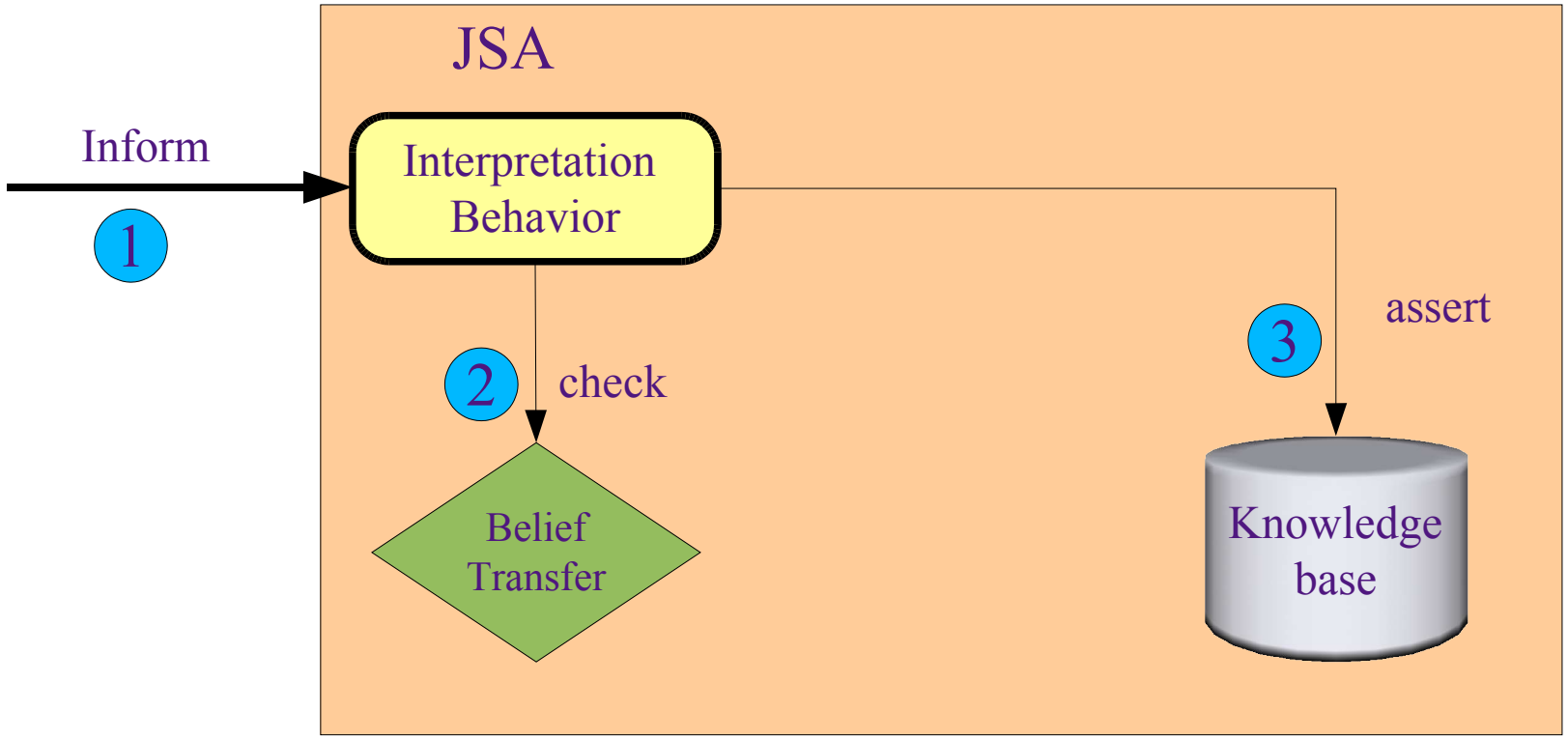
Instantiating a pattern

```
Term agent = SLPatternManip.fromTerm("(agent-identifier :name foo)");
Formula phi = SLPatternManip.fromFormula("true");
Formula formula = SLPatternManip.instantiate(fPattern,
                                             "agent", agent,
                                             "phi", phi);
```

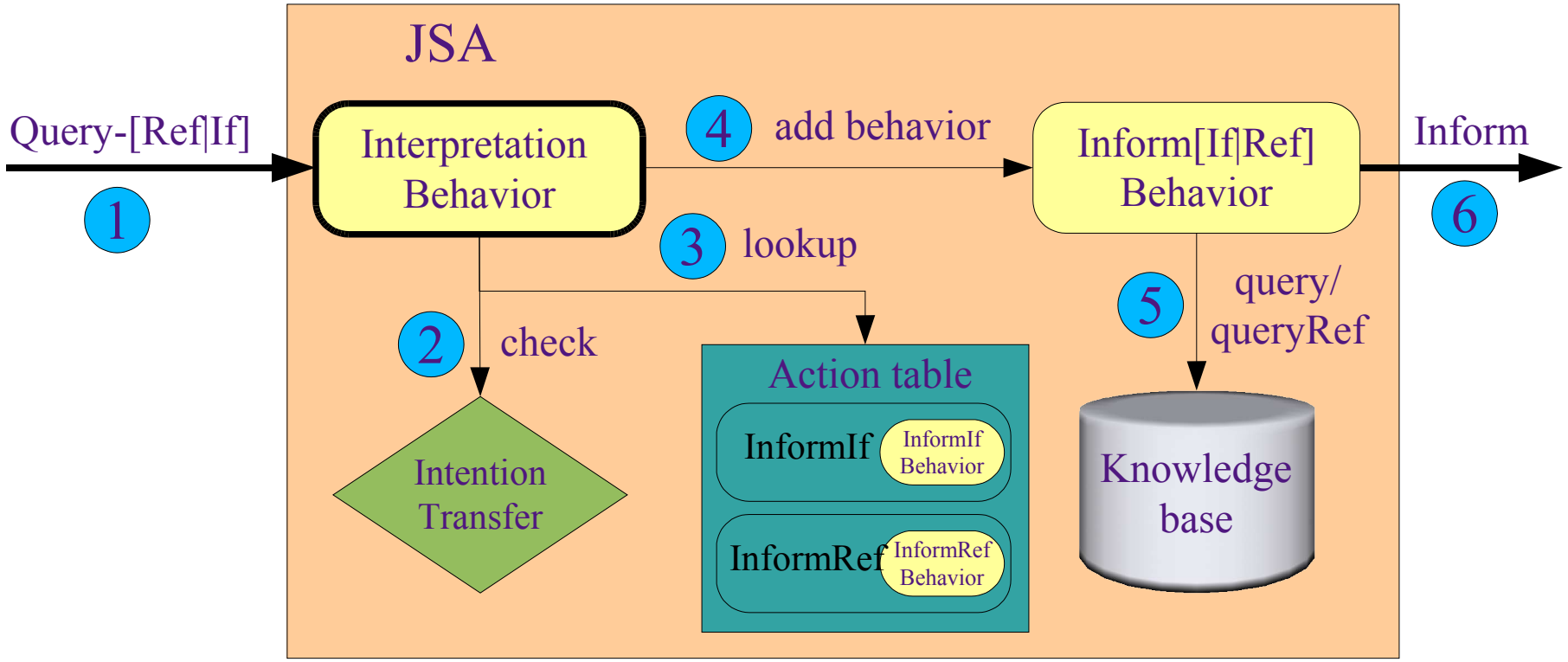
Matching a pattern

```
MatchResult result = null;
if ( (result = SLPatternManip.match(fPattern, formula)) != null ) {
        System.out.println("agent = "+result.getTerm("agent"));
}
```

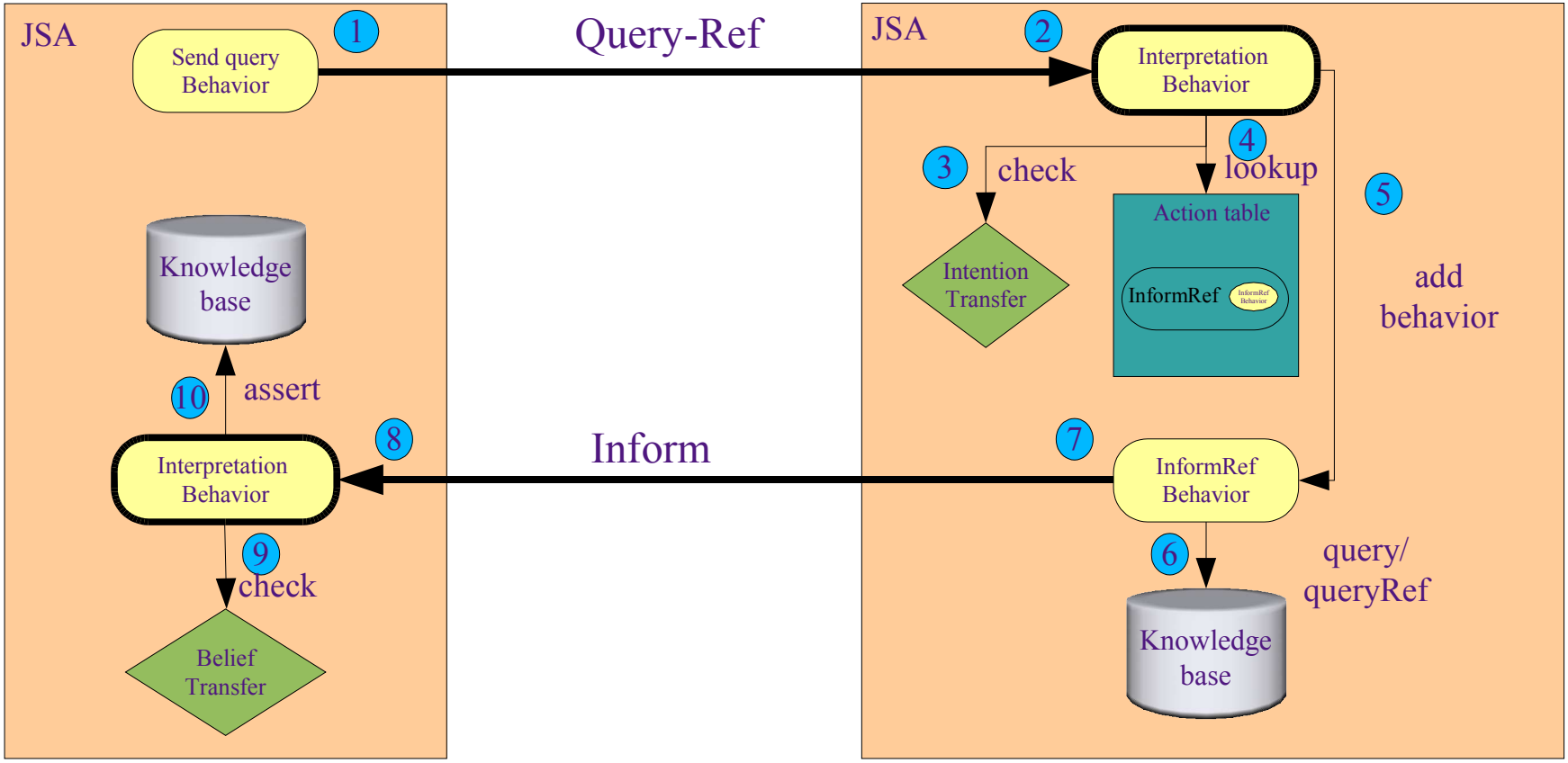
Handling an Inform act



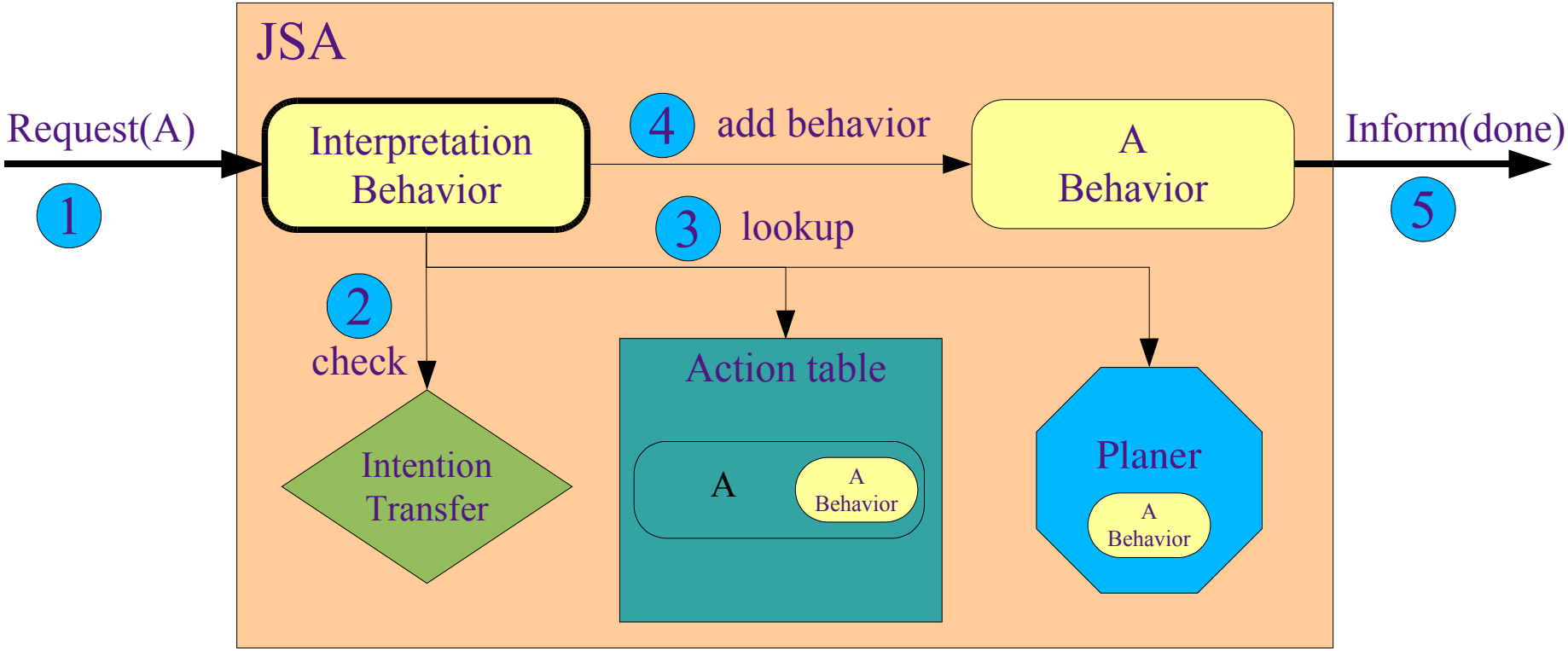
Handling a Query act



Handling a Query protocol



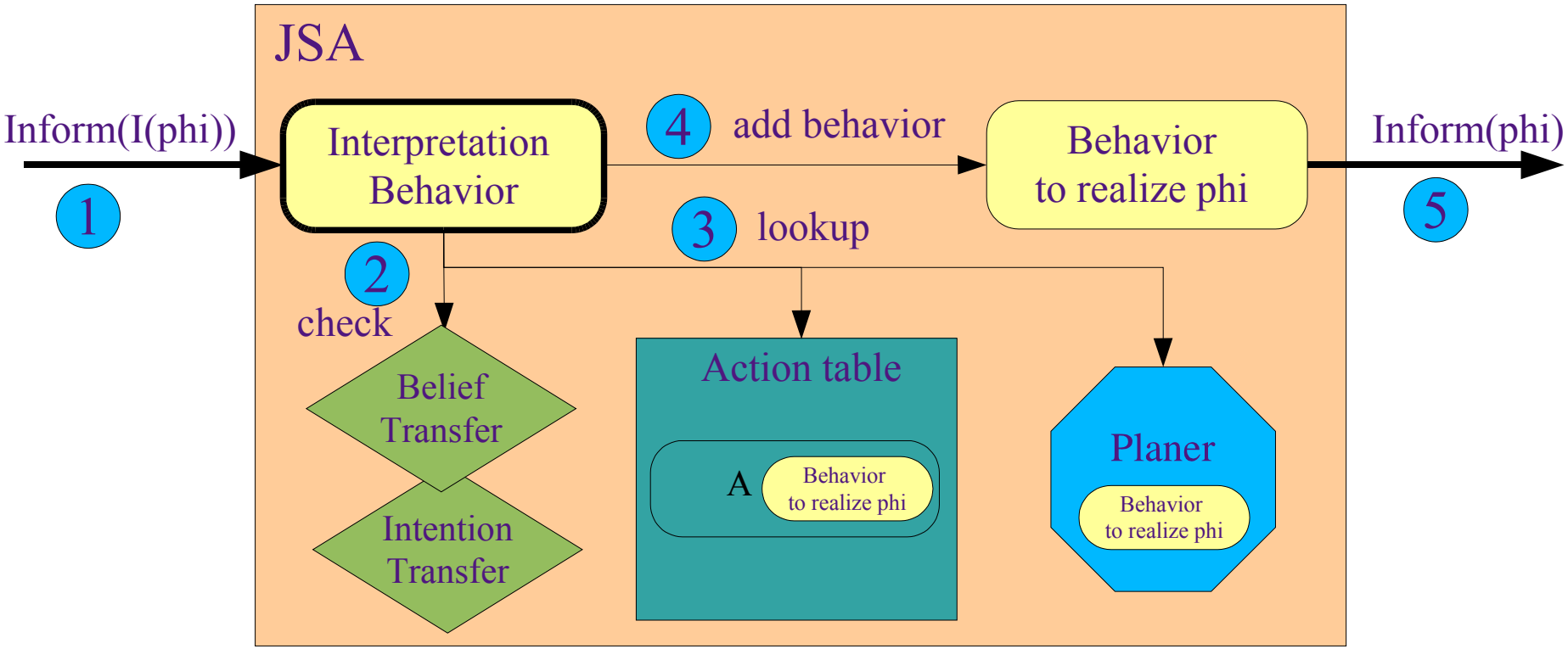
Handling a Request act



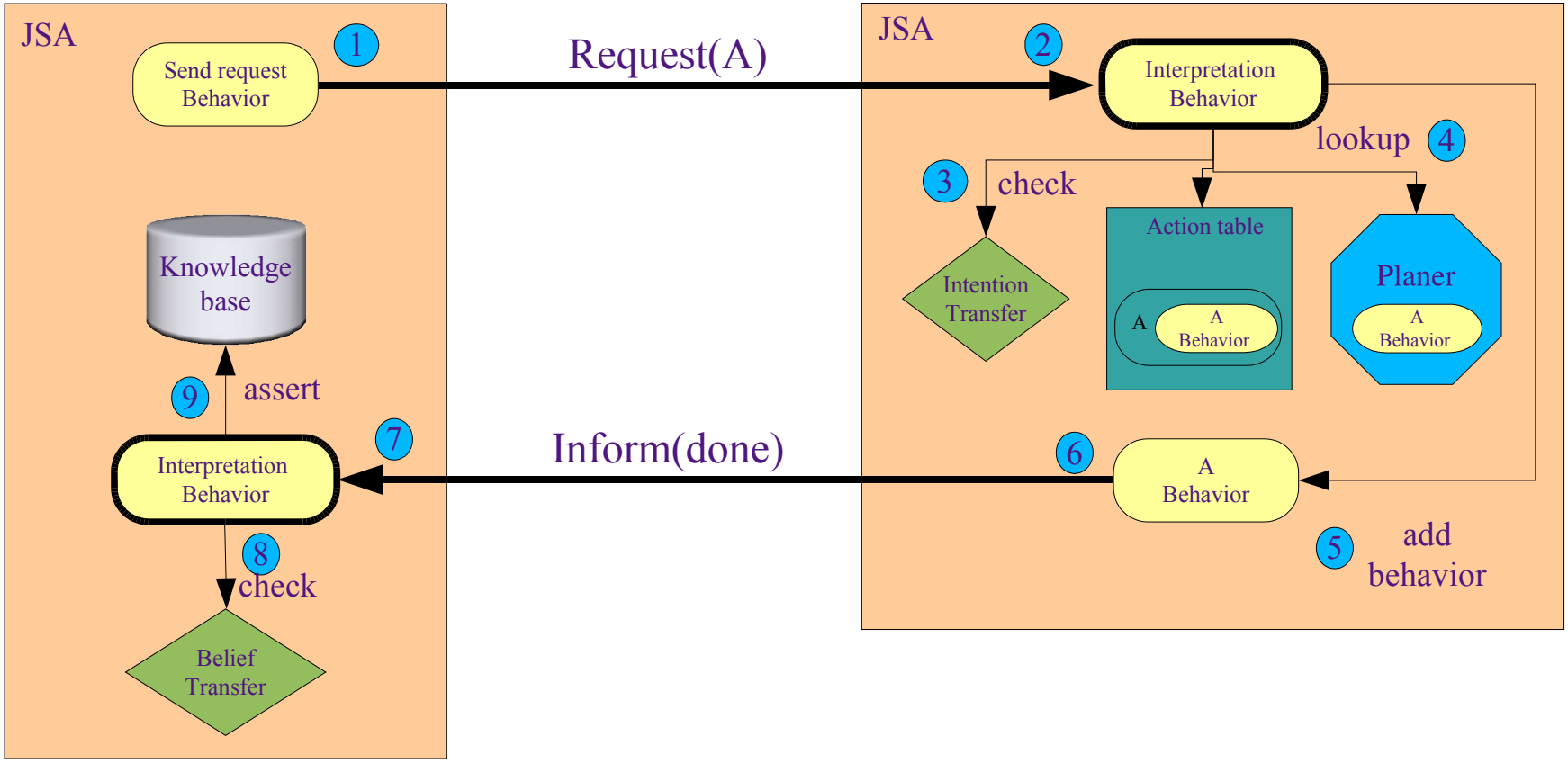
Remember Query-Ref is handled as a Request(InformRef)!!!

Unrestricted

An other kind of request



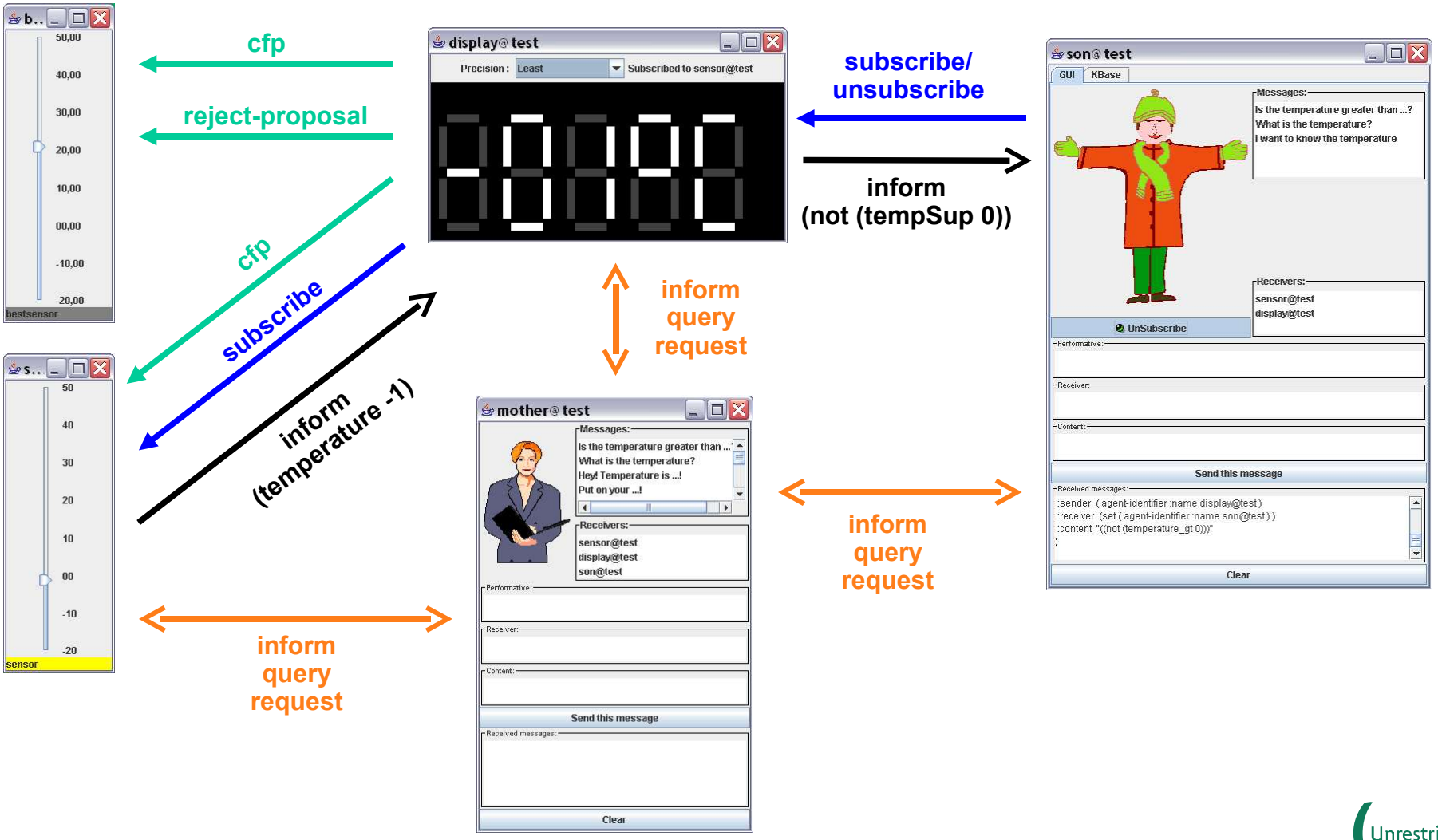
Handling a Request protocol



Demonstration

- Principles
  - JADE Semantics Add-on & JADE Semantic Agent (JSA)?
  - Components of a JSA
  - The semantic interpretation behavior
  - What does a JSA look like ?
- General mechanisms
  - SL expressions handling
  - Handling Inform, Query and Request acts
  - Handling Query and Request protocols
- **Demonstration**
- Coding a JADE Semantic Agent
  - Setting up the knowledge base
  - Setting up the action table
  - Setting up the standard customization

The temperature demo



Coding a JADE Semantic Agent

- Principles
  - JADE Semantics Add-on & JADE Semantic Agent (JSA)?
  - Components of a JSA
  - The semantic interpretation behavior
  - What does a JSA look like ?
- General mechanisms
  - SL expressions handling
  - Handling Inform, Query and Request acts
  - Handling Query and Request protocols
- Demonstration
- **Coding a JADE Semantic Agent**
  - **Setting up the knowledge base**
  - **Setting up the action table**
  - **Setting up the standard customization**

Setting up the knowledge base (1)

- **Identify pieces of information to be handled by agents**
  - (temperature **x**) means temperature value is **x**
  - (temperature_gt **x**) means temperature value is greater than **x**
  - (wearing **agent clothing**) means **agent** is wearing **clothing**
- **Identify the consistency rules and the inference rules**
  - (temperature **x**) & (temperature **y**) → **x = y**
    - **x** can have one single value at a time
  - (temperature **y**) & **x < y** → (temperature_gt **x**)
    - Temperature value is greater than **x**, for any **x** lesser than the current temperature value (**y**)

Setting up the knowledge base (2)

- The KBase interface defines several operations called by the framework
  - void **assertFormula**(Formula formula)
  - Bindings **query**(Formula)
  - ListOfTerm **queryRef**(IdentifyingExpression)
- The framework provides the FilterKBase class that implements the KBase interface
- A FilterKBase stores simple beliefs, and can be customized by adding filters
  - **KBAssertFilter**
  - **KBQueryFilter**
  - **KBQueryRefFilter**

Setting up the knowledge base (3)

Example: using the KBAssertFilter to ensure the uniqueness of the temperature value

```
((KBFilterManagment)myKBase).addKBAssertFilter(
    new KBAssertFilterAdapter("(B ??agent (temperature ??x))") {
        public Formula applyBefore(Formula formula) {
            if ((myKBase.query(formula) != null)) {
                return new TrueNode();
            }
            else {
                kbase.removeAllFormulae("(temperature ??x)");
                kbase.removeAllFormulae("(not (temperature ??x))");
                kbase.removeAllFormulae("(temperature_gt ??x)");
                kbase.removeAllFormulae("(not (temperature_gt ??x))");
                return formula;
            }}});
```

Setting up the knowledge base (4)

Example: using a KBQueryFilter to implement simple inference rules for the temperature_gt predicate

```
((KBFilterManagment)myKBase).addKBQueryFilter(
  new KBQueryFilterAdapter("(B ??agent (temperature_gt ??x))") {
    public Bindings apply(Formula formula) {
      Long x = ((Constant)applyResult.getTerm("x")).intValue();
      ListOfTerm terms = myKBase.queryRef("(iota ?y (temperature ?y))");
      if ( terms.size() != 0 && ((Constant)terms.get(0)).intValue() > x ) {
        return new BindingsImpl();
      } else {
        terms = myKBase.queryRef("(iota ?y (temperature_gt ?y))");
        if ( terms.size() != 0 && ((Constant)terms.get(0)).intValue() >= x ) {
          return new BindingsImpl();
        }
      }
      return null;}});
```

Setting up the action table (1)

- **Identify the actions to be handled by agents**
  - (**PUT-ON** :clothing ??clothing)
  - (**TAKE-OFF** :clothing ??clothing)
  - (**WAIT** :time ??time)
- **Define the pre- and post-condition of each actions**
  - (**PUT-ON** :clothing **??clothing**)
    - Pre-condition: (not (wearing ??agent **??clothing**))
    - Post-condition:(wearing ??agent **??clothing**)
  - (**WAIT** :time ??time)
    - Pre-condition: true
    - Post-condition:true
- **Implement the actions**

Setting up the action table (2)

Implementing the PUT_ON action

```
getMySemanticActionTable().addSemanticAction(
  new OntologicalAction(
      getMySemanticActionTable(),
      "(PUT-ON :clothing ??clothing)",
      SLPatternManip.fromFormula("(wearing ??agent ??clothing)"),
      SLPatternManip.fromFormula("(not (wearing ??agent ??clothing))"))   {
    public void perform(OntoActionBehaviour behaviour) {
      ((ManAgent)myAgent).putOn(getActionParameter("clothing").toString());
      behaviour.setState(SemanticBehaviour.SUCCESS);
    }
  });
```

Setting up the action table (3)

Implementing the WAIT action

```
getMySemanticActionTable().addSemanticAction(
  new OntologicalAction(
      getMySemanticActionTable(),
      "(WAIT :time ??time)",
      SLPatternManip.fromFormula("true"),
      SLPatternManip.fromFormula("true"))        {
    public void perform(OntoActionBehaviour behaviour) {
      if (behaviour.getState() == SemanticBehaviour.START) {
        behaviour.block(new Long(getActionParameter("time").toString()));
        behaviour.setState(WAITING);
      }
      else {
        behaviour.setState(SemanticBehaviour.SUCCESS);
      }
    });
```

Setting up the standard customization (1)

- **The behavior of a JSA can be customized by implementing the interface StandardCustomization**
- **In particular, this customization includes the Belief and Intention Transfer cooperative attitudes**

The 2 following operations have to be implemented

boolean **acceptBeliefTransfer**(Formula **formula**, Term **agent**)

boolean **acceptIntentionTransfer**(Formula **formula**, Term **agent**)

- the **formula** argument corresponds to the belief or intention to accept, such as (temperature 12)
- the **agent** argument identifies the agent the belief or intention comes from

Unrestricted

Setting up the standard customization (2)

Implement the belief transfer of the sensor agent

```
setMyStandardCustomization(new StandardCustomizationAdapter() {
      public boolean acceptBeliefTransfer(Formula formula, Term agent) {
          return  (SLPatternManip.match("(temperature ??x)", formula)==null)
                &&(SLPatternManip.match("(not (temperature ??x))", formula)==null)
                &&(SLPatternManip.match("(temperature_gt ??x)", formula)==null)
                &&(SLPatternManip.match("(not (temperature_gt ??x))", formula)==null);
      }});
```

Implement the intention transfer of the son agent

```
setMyStandardCustomization(new StandardCustomizationAdapter() {
      public boolean acceptIntentionTransfer(Formula goal, Term agent) {
          String motherID = "(agent-identifier :name "+motherAID.getName()+")";
          return agent.equals(SLPatternManip.fromTerm(motherID))
      }});
```

Unrestricted

Setting up the standard customization (3)

- **Standard customization makes it also possible to handle acts like CFP or Propose**

```
setMyStandardCustomization(new StandardCustomizationAdapter() {
public boolean handleProposal(Term agent, ActionExpression action, Formula formula) {
    Term act = SLPatternManip.fromTerm("(action ??receiver (INFORM-REF :content
\"((any ?x (temperature ?x)))\" :receiver (set ??agent) :sender ??receiver))");
    MatchResult matchResult = SLPatternManip.match(act, action);
    if ( matchResult != null ) {
        Formula cond = SLPatternManip.fromFormula("(precision ??x)");
        matchResult = SLPatternManip.match(cond, formula);
        if ( matchResult != null ) {
            ((DisplayAgent)myAgent)
                .handleProposal((IntegerConstantNode)matchResult.getTerm("x"),
                                agent, action, formula);
        }
    }
    return true;}});
```

Summary

- **About FIPA-ACL acts and protocols**
  - → They are intrinsically handled by a JSA
  - → No receive to implement
  - → The only tasks to be performed is to define and implement the **knowledge base,** the **actions,** and the **cooperation principles**
- **About the architecture of a JSA**
  - → **Other knowledge base implementation can be used**
  - → **A planer can also be used**

→ **Finally, JADE + Semantics Add-on = a step towards a real communication-oriented middleware**

Unrestricted

Thanks!

Download the Semantics Add-on
http://jade.tilab.com

Provide your feed-back!