JADE Semantics Add-on

tutorial
& practical exercises

Vincent Louis
Orange Labs

June 2nd, 2007

orange™

research & development

1 preliminaries

- introduction
- the FIPA-ACL formal specifications
- handling FIPA-SL expressions

# what is the JSA?

- a JADE extension to automate the interpretation of the meaning of messages exchanged by agents (according to the semantics of the FIPA-ACL standard)

- a framework to build more flexible agents

- a set of classes, which makes simpler the coding of JADE agents

- agents built on top of JSA = semantic agents

research & development

the most simple semantic agent

- the class SemanticAgentBase provides a default implementation for semantic agents
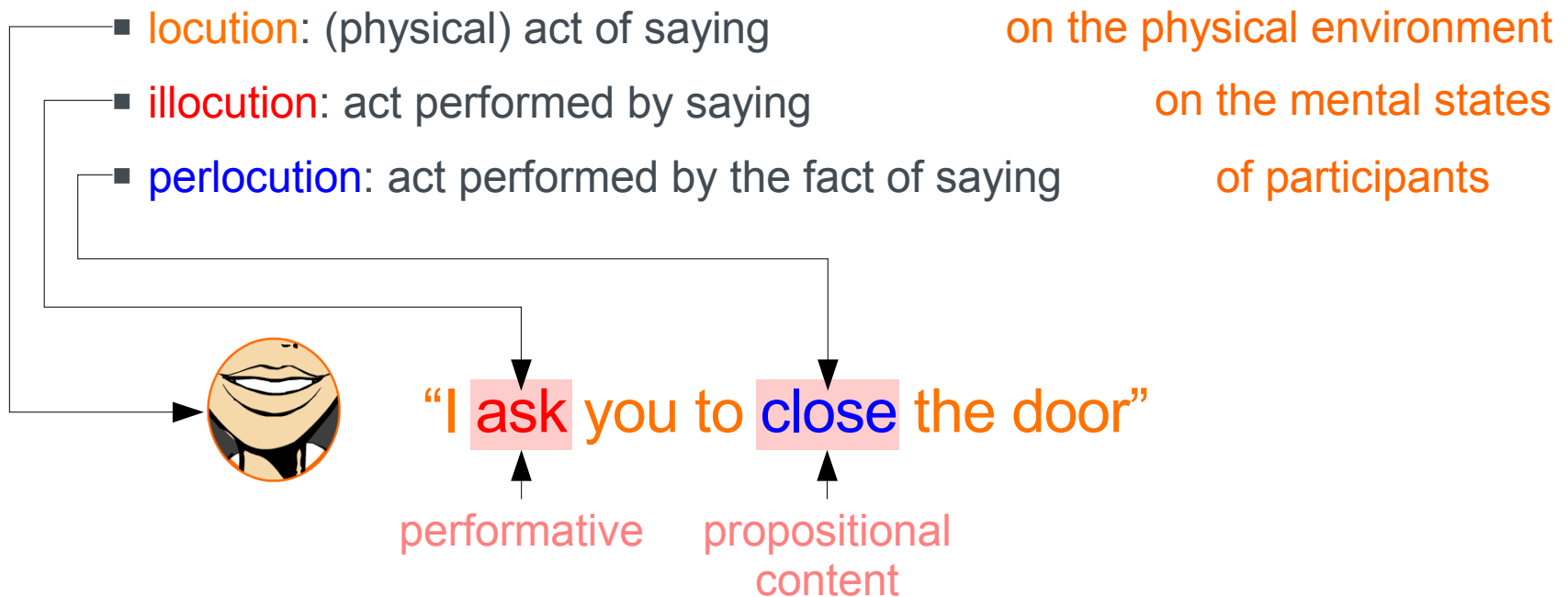- it makes it possible to interpret all FIPA-ACL messages (but proxy)

java jade.Boot mySemanticAgent:SemanticAgentBase

inform (age smith 15)

query-ref ((any ?x (age smith ?x)))

inform (age smith 15)
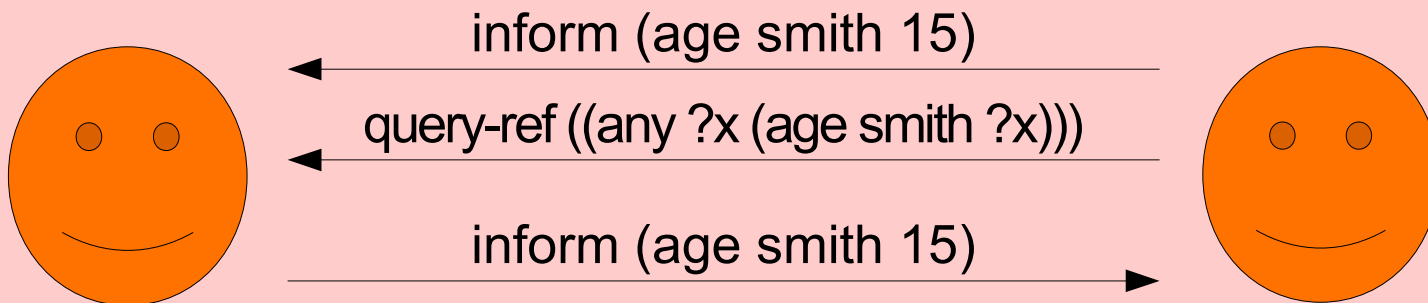
FIPA-ACL: a language for understanding each other

■ logical formalization of the philosophical theory of speech acts [Austin, Searle, Vandervecken, Sadek]

■ "communication is action"
An utterance involves 3 levels

▪ locution: (physical) act of saying   on the physical environment

▪ illocution: act performed by saying   on the mental states

▪ perlocution: act performed by the fact of saying   of participants

"I ask you to close the door"

performative   propositional content

research & development

France Telecom Group

communicative acts

- syntax   example

(performative*   INFORM

    :sender   sending agent*   (agent-identifier :name me)

    :receiver   receiving agents*   (set ...)

    :content   propositional content* "((sunny))"
                              (action, proposition or IRE)

    :content-language   content language   fipa-sl

    :ontology   content vocabulary   weather-forecast

      …other parameters…

)

- semantics: formal definition (in a logical framework) of
  - FP : feasibility precondition
  - RE : rational effect or perlocutionary effect

example: query-if

(query-if

    :sender s

    :receiver r

    :content "((age smith 10))"

)

the agent s asks
the agent r
whether the proposition
(age smith 10) *holds*

- **precondition**: s does not know the truth of (age smith 10)
- **rational effect**: r performs the action
    (inform-if :sender r :receiver s :content "((age smith 10))")

- the JSA interpretation engine entails the reaction to a message from its formal features

research & development

France Telecom Group

FIPA-ACL: about 20 performatives

- see http://www.fipa.org/specs/fipa00037
  - information transmission
    inform, inform-if, inform-ref, confirm, disconfirm
  - request on information   on actions
    query-if, query-ref, subscribe,   request, request-when(ever)
  - negotiation
    cfp, propose, accept-proposal, reject-proposal
  - action management
    cancel, agree, refuse
  - task delegation
    propagate, proxy
  - error management
    failure, not-understood

research & development   France Telecom Group

FIPA-SL

- logical language, including
  - a first order predicate logic (FOL)
  - a modal logic, with modalities that represent
    - agents' mental states: believes (B, U) and intentions (I)
    - action occurrences: past (done) and future (feasible) ones
- prefixed syntax like in LISP: (and sunny cold)
- 2 main types of expressions
  - terms: represent domain objects
    instances, actions, object descriptions (IRE), ...
  - formulas: represent facts, which can be true or false
- see http://www.fipa.org/specs/fipa00008

research & development

France Telecom Group

FIPA-SL terms (1/2)

- **constants**

  numbers: 1, -6.5E1   strings: "this is a \"FIPA-SL\" string"

  dates: *YYYYMMDD*T*HHMMSSmmm*z, 20060331T093000000z

  binary constants: #*N*"*byte-sequence*

- **sets and sequences**

  (**set** *elem1 elem2 ...*)   duplicates and order **are not** significant

  (**sequence** *elem1 elem2 ...*)   duplicates and order **are** significant

- **functional terms (e.g. class instances)**

  (*funct-symbol* :*param_name param_value ...*)

  (person :name john :age 20)

research & development

France Telecom Group

FIPA-SL terms (2/2)

- ■ actions (including communicative acts)

  (**action** *actor act*)   *act* is usually given as a functional term

  (action s (inform :sender s :content "((sunny))" :receiver (set r)))

  - ■ action composition

    (**;** *a1 a2*)   **sequence**: do *a1*, then do *a2*

    (**|** *a1 a2*)   **indeterministic choice**: do either *a1* or *a2*

- ■ identifying reference expressions (IRE)

  (*quant term formula*)   where *quant* ∈ {**any**, **iota**, **some**, **all**}

  (**iota** ?x (age john ?x))   **the only value** related to john by age
                    (i.e. *the* age of john)

  (**any** ?x (age ?x 20))   **any value** representing a 20 years old person

  (**all** (sequence ?x ?y) (age ?x ?y))   **exactly all** pairs (person, age)

  (**some** (sequence ?x ?y) (age ?x ?y))   **any set** of pairs (person, age)

FIPA-SL formulas (1/2)

- **atomic formulas**
  - (*pred-name param1 param2 ...*)   all *paramN* are terms
  - (age (person :name john :age 20) 20)
  - predefined predicates and constants: =, result, true, false
- **FOL logical connectors**
  - **not** (unary), **and**, **or**, **implies**, **equiv** (binaries)
  - (and sunny cold), (equiv (not cold) hot)
  - for convenience, **or** and **and** operators are n-ary operators in the JSA fw.
  - (and sunny cold winter) is read as (and sunny (and cold winter))
- **FOL quantifiers**
  - (**exists** *var formula*)   there is at least one object *var* satisfying *formula*
  - (**forall** *var formula*)   all objects *var* satisfy *formula*
  - (forall ?x (implies (person ?x)   all persons
              (exists ?y (age ?x ?y))))   have an age

research & development   France Telecom Group

FIPA-SL formulas (2/2)

- **mental state modalities**

    (*modal-op agent formula*)　　　where　　　*agent* is a term,
    　　　　　　　　　　　　　　　　　　　　　　　modal-op $\in$ {**B**, **U**, **I**}

    (**B** (agent-identifier :name john) sunny)
    (**B** (agent-identifier :name john) (**not** sunny))　⎫ 3 different
    (**not** (**B** (agent-identifier :name john) sunny))　⎭ kinds of belief

- **action occurrence modalities**

    (*modal-op action formula*)　　　where　　　*action* is a term of kind action,
    　　　　　　　　　　　　　　　　　　　　　　　modal-op $\in$ {**done**, **feasible**}

    (**done** (| a1 a2) sunny)　　　either a1 or a2 has just occurred,
    　　　　　　　　　　　　　　　　and sunny was true just before

    (**feasible** (action s (inform :receiver (set r) :content "(sunny)")) (B r sunny))
    　　　　　　　it is possible to perform the inform act, and if so,
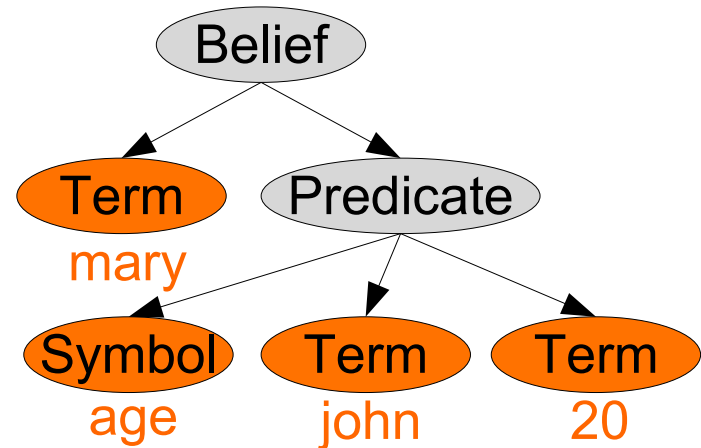   　　　　　　　r will believe its content just after its performance

handling SL expressions with the JSA

■ SL expressions are represented by Directed Acyclic Graphs of Node objects (counterparts of JADE AbsXXX objects)

(B mary (age john 20))

see package
jade.semantics.lang.sl.grammar

■ Some Node objects have specific computation methods
   e.g. getSimplifiedFormula() on Formula instances
■ The main class to handle Node objects is
   jade.semantics.lang.sl.tools.SL

parsing/unparsing SL expressions

- methods **fromTerm(String)** and **fromFormula(String)**
  - parse a string expressed in SL syntax (into a **Node** object)
- method **toString()**
  - unparse a **Node** object into a string expressed in SL syntax

```
Term john = SL.fromTerm("(agent-identifier :name john)");
Formula f = SL.fromFormula("(age " + john + " 20)");

System.out.println("f = " + f);
System.out.println("agent = " +
        ((FunctionalTermParamNode)john).getParameter("name"));
```

```
prints            f = (age (agent-identifier :name john) 20)
                  agent = john
```

SL expression patterns

- the Node hierarchy extends FIPA-SL with "meta-references"
    - it is possible to build "patterns" of expressions
    - meta-references (MR) within a pattern are prefixed by "??"
    - MR may be replaced with expressions of the proper type
    - 2 occurrences of the same MR denote the same expression
    - example   (I ??agent (B ??agent ??formula))

        ??agent may be replaced with a term   ??formula may be replaced with a formula

        (I   john   (B   john   sunny   ))

- 2 fundamental operations on patterns
    - instantiation   replace each occurrence of a MR within a pattern with the same expression
    - matching   check whether an expression may result from the instantiation of a pattern

creating and instantiating patterns

■ **creating patterns:** SL.fromXXX
                            as for creating regular expressions

```
Formula pattern = SL.fromFormula
                        ("(I ??agent (B ??agent ??formula))");
```

■ **instantiating patterns:** aNode.instantiate(aString,anotherNode)
            or SL.instantiate(aNode, [aString, anotherNode]*)

```
Term john = SL.fromTerm("(agent-identifier :name john)");
Formula f = SL.fromFormula("sunny");

Formula myFormula = (Formula)SL.instantiate(pattern,
                                    "agent", john, "formula", f);
```

very useful to create expressions

research & development   France Telecom Group

matching patterns

- **aNode.match(Node)**
  returns   a **MatchResult** if a matching is possible
                      or **null** if no matching is possible

- **aMatchResult.getXXX(String)** and **aMatchResult.XXX(String)**
  get the value of a given MR satisfying the matching

```
MatchResult result = pattern.match(myFormula);
if (result != null) {
        System.out.println("agent = " + result.term("agent"));
        System.out.println("formula = " + result.formula("formula"));
}
else System.out.println("no match");
```

fundamental to recognize or filter expressions

practical exercises developing an album application

- 4 progressive exercises

- under the tutorials directory
  - exercises/
    - img*.jpg: predefined images for the album application
    - src/album/tools: predefined GUI classes
    - src/album/versionX/Album.java: album class to develop
    - src/album/versionX/Viewer.java : viewer class to develop
    - within build.xml, set variable "tuto-home" to the tutorial directory
    - compile with   ant X jar   (X = 1, 2, 3 or 4)
    - run with   ant X album
    -     ant X viewer
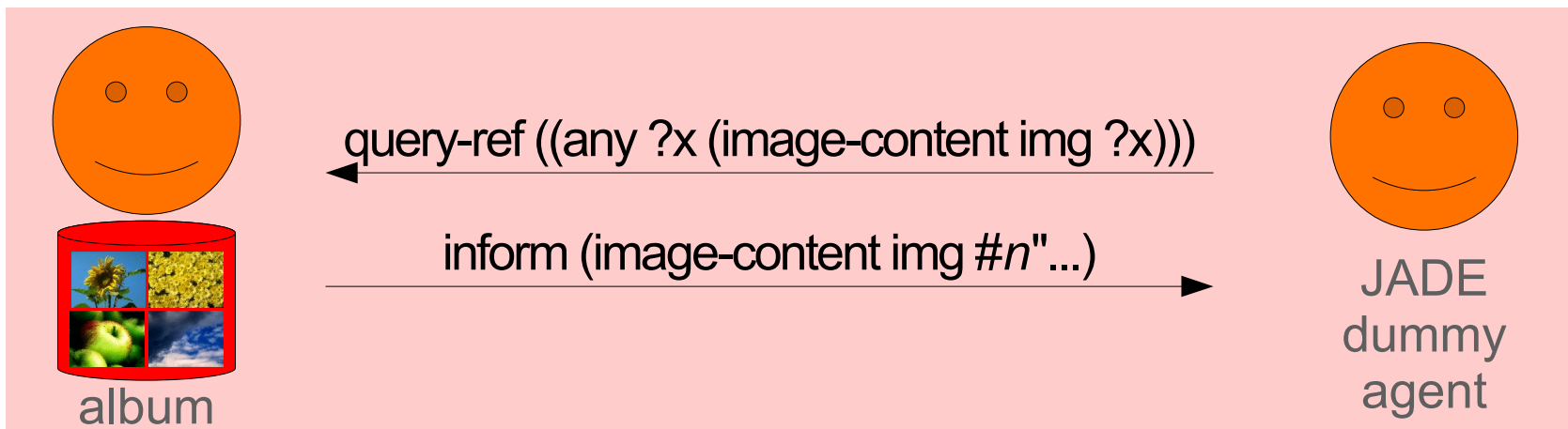   - develop all yourself or start from the *.java.sqel templates
  - solutions/
    - same structure, with completed *.java files

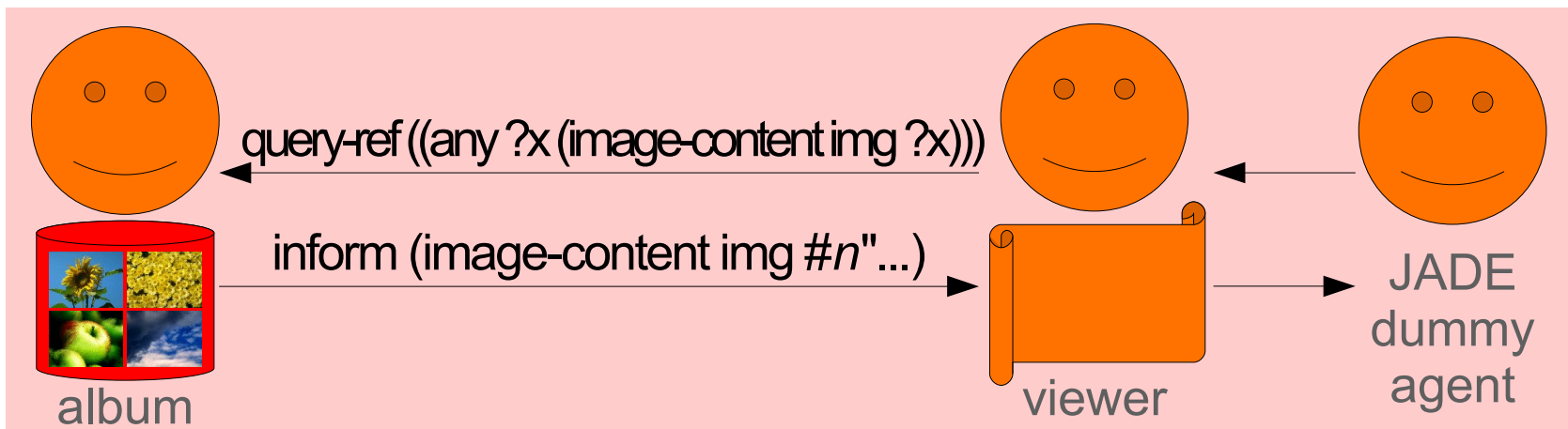# album application – exercise 1 handling SL expressions

- register a picture within the album agent
  - use the application-specific predicate (image-content *id byte-content*)
  - read the byte content from the file given as an agent's argument
  - use SemanticCapabilities.interpret(Formula) within the setup() method of the agent

- get the picture (as a byte content) with a JADE dummy agent

query-ref ((any ?x (image-content img ?x)))

inform (image-content img #*n*"...)

album

JADE dummy agent

research & development

album application – exercise 1 handling SL expressions

- make the request performed by the viewer agent
  - read the name of the album agent from the agent's arguments
  - use SemanticCapabilities.queryRef(IdentifyingExpression) within the setup() method of the agent

- check the exchanged messages thanks to the JADE sniffer
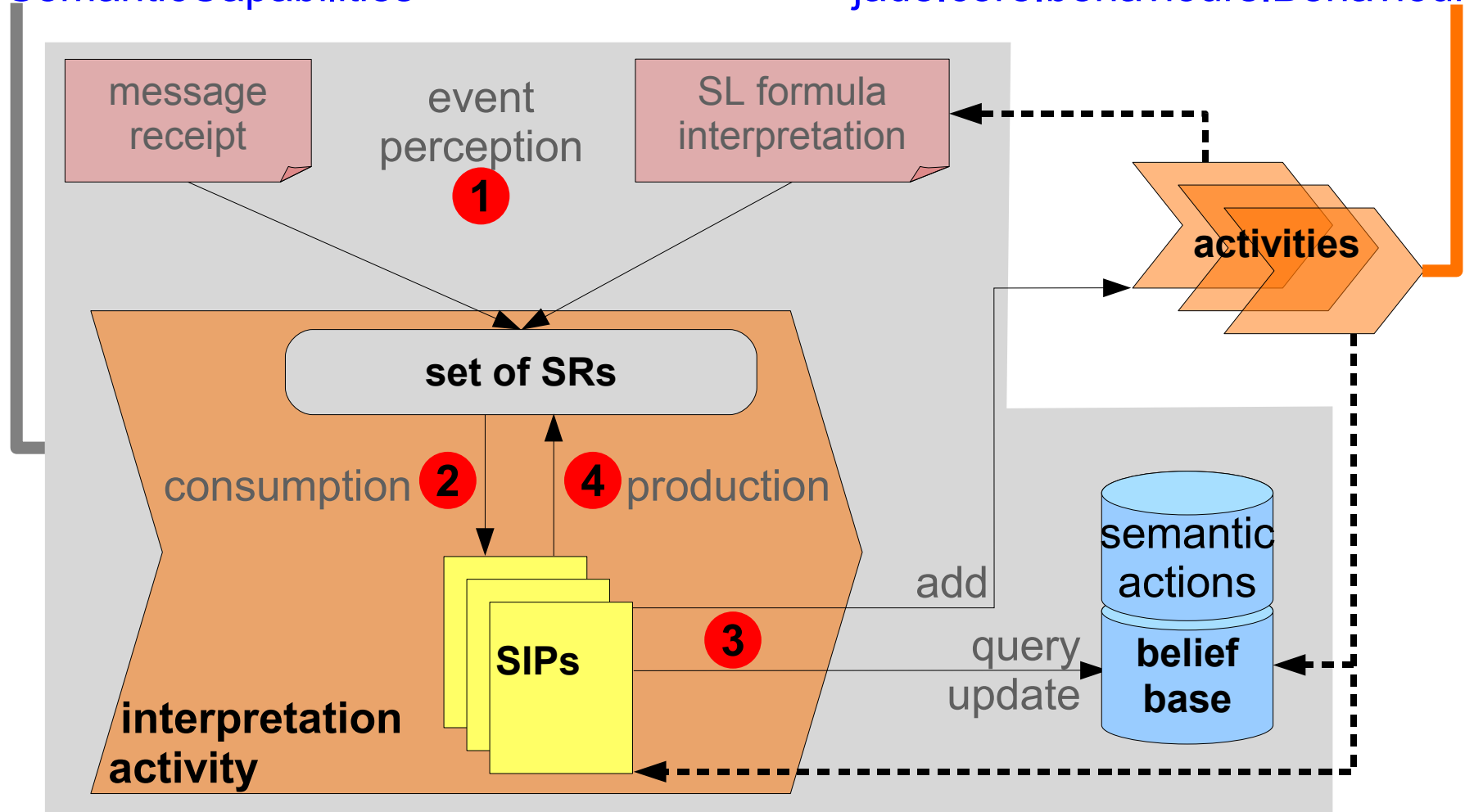- request the viewer with a dummy agent

query-ref ((any ?x (image-content img ?x)))

inform (image-content img #*n*"...)

album

viewer

JADE dummy agent

research & development

2 JSA interpretation engine

- semantic agent functioning
- semantic agent software structure
- semantic interpretation principles (SIPs)

research & development

France Telecom Group

JSA interpretation engine

SemanticCapabilities

jade.core.behaviours.Behaviour

message receipt

event perception **1**

SL formula interpretation

activities

set of SRs

consumption **2**

4 production

SIPs

interpretation activity

add

3

query update

semantic actions

belief base

research & development

France Telecom Group

interpretation algorithm

- event perception: produce an initial SR
  - receipt of a message $m \rightarrow$ (B agent (done (action *sender m*)))
  - interpretation of a formula $\rightarrow$ *f*
- **while** the list of SRs is not empty, **do**

         remove a SR from the list;

         if the SR is logically equivalent to **false**, then exit;

         apply all possible SIPs to the SR;

         add all produced SRs to the list;

  end while

- the interpretation finishes when
  - the list of SRs is empty $\rightarrow$ "normal" case
  - a SR equivalent to **false** is produced $\rightarrow$ sending of a **not-understood**
  - no SIP is applicable $\rightarrow$ assertion of remaining SRs into the belief base

France Telecom Group

software architecture

research & development

France Telecom Group
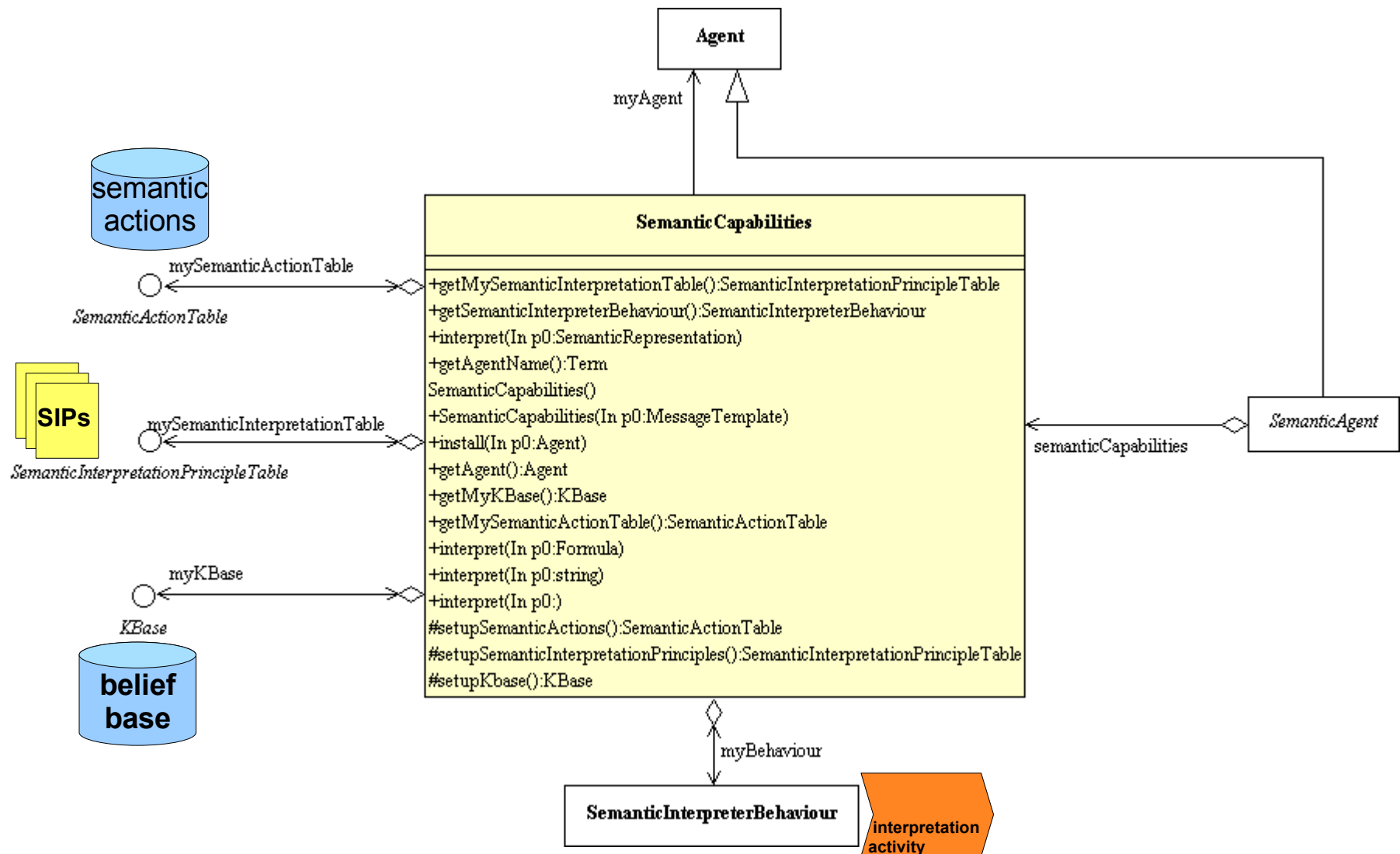
semantic agent skeleton

- semantic agent = JADE agent + SemanticCapabilities

  this attribute specifies the interpretation engine functioning

```
public class MyJSA extends SemanticAgent {
    class MySematicCapabilities extends SemanticCapabilities {
        protected SemanticInterpretationPrincipleTable
                        setupSemanticInterpretationPrinciples() {...}
        protected KBase setupKbase() {...}
        protected SemanticActionTable setupSemanticAction() {...} ...
    }
    public MyJSA() {
        setSemanticCapabilities(new MySematicCapabilities());
    }
    public void setup() {
        super.setup();    ...
    }
}
```

research & development

main SemanticCapabilities operations

- **general operations**
  - getAgent()
    returns the JADE agent instance wrapping the semantic agent
  - getAgentName()
    returns a SL term representing the semantic agent AID
  - getSemanticInterpreterBehaviour()
    returns the Behaviour running the semantic interpretation engine
  - interpret(Formula/String/SR)
    runs the semantic interpretation engine on a given formula

- **operations to perform communicative acts**
 - *performative*(*propositional_content_params*,...,*receiver*)

  Formula, ActionExpression or IdentifyingExpression   Term or Term[]

  - example: inform(Formula,Term), request(ActionExpression,Term), ...

SIPs in the heart of interpretation

- **a "Semantic Representation" (SR)**
  - represents a part of the meaning of an event
  - conveys a subjective meaning with respect to the agent
    - of the form (B *myself* ??phi) or (B *myself* (I *myself* ??phi))

- **a "Semantic Interpretation Principle" (SIP)**
  - elaborates a part of the meaning of an event by
    - consuming a SR (the SIP is said to be applied to the SR)
    - possibly modifying the agent's internal state
    - possibly producing new SRs
  - has an application index, which makes it possible to
    - order the application of SIPs
    - apply SIPs only to relevant SRs (such that *SR index ≥ SIP index*)

consumption **2** **4** production

add

SIPs **3**

query
update

- **the interpretation algorithm is an ad-hoc rule engine**
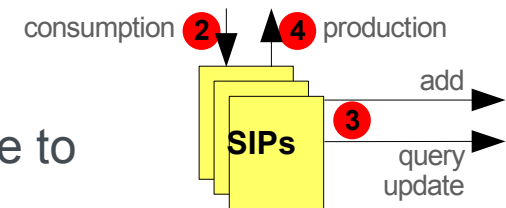
standard SIPs (1/2)

- standard SIPs implement the **generic principles** of the rational agent theory, which FIPA relies on

- **ActionFeature**   (B myself (done *??message* true))

  upon receipt of a *message*, produces SRs representing the formal FP and RE of the corresponding communicative act
  (uses the table of SemanticAction, which includes all FIPA acts)

- **BeliefTransfer**   (B myself (I *??agent* (B myself *??belief*)))

  decides to adopt a *belief* suggested by another *agent*
  (e.g. upon interpretation of an inform)

- **IntentionTransfer**   (B myself (I *??agent ??goal*))

  decides to adopt the intention of another *agent*'s *goal*
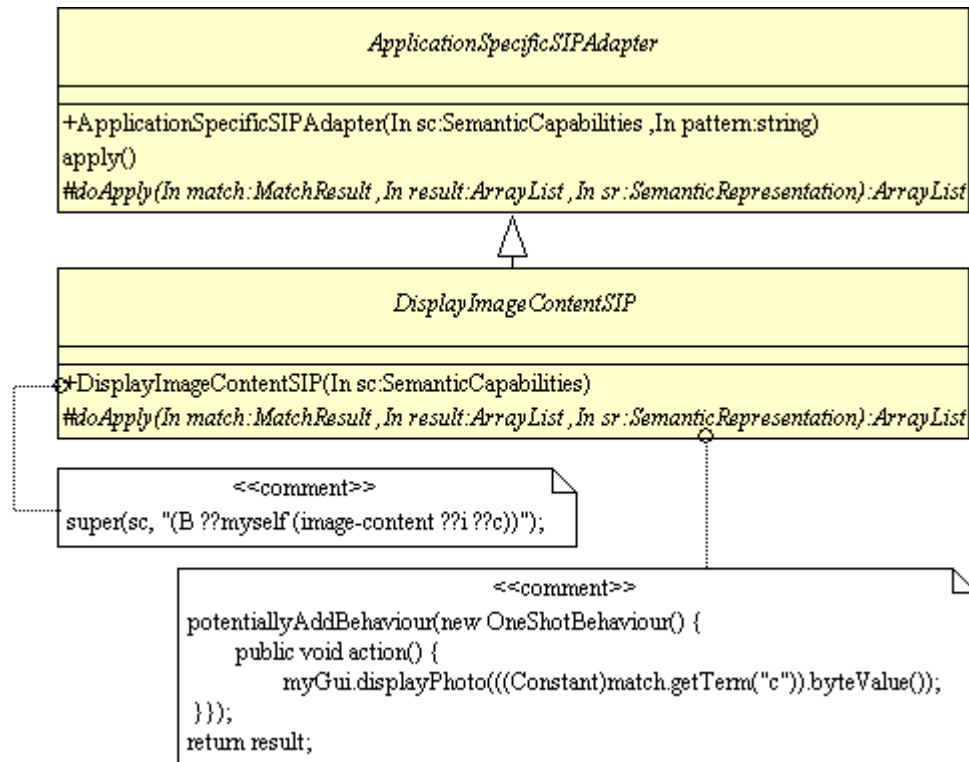 (elementary form of cooperation, e.g. upon interpretation of a request)

standard SIPs (2/2)

- **Planning**   (I myself *??goal*)

  creates (and adds to the agent) an activity to reach an intended *goal*
  the JSA is provided with no default planning SIP, however, the
  PlanningSIPAdapter is intended to plug external planers if needed

- **ActionPerformance**   (I myself (done *??action* true))

  creates (and adds to the agent) an activity to perform an intended
  action (uses the table of SemanticAction)

- **RationalityPrinciple**   (I myself *??goal*)

  creates (and adds to the agent) an activity to perform an action, whose
  effect matches an intended *goal* (uses the table of SemanticAction)

research & development

application-specific SIPs

- customize the semantic agents' behaviour with specific SIPs
  3 main cases
    - reactive production of an applicative "piece of meaning" (resulting from the interpretation of SL formulas): e.g. production of an intention
    - triggering of applicative "notifications", e.g. to control a GUI
    - specialization of standard SIPs (e.g. BeliefTransferSIPAdapter)

```
class MySematicCapabilities extends SemanticCapabilities {
    protected SemanticInterpretationPrincipleTable
                            setupSemanticInterpretationPrinciples() {
        table = super.setupSemanticInterpretationPinciples();
        table.addSemanticInterpretationPrinciple(mySIP);
        ...
        return table;
    }   ...
}
```

research & development

defining an application-specific SIP

```
ApplicationSpecificSIPAdapter

+ApplicationSpecificSIPAdapter(In sc:SemanticCapabilities ,In pattern:string)
apply()
#doApply(In match:MatchResult , In result:ArrayList , In sr:SemanticRepresentation):ArrayList
```

```
DisplayImageContentSIP

+DisplayImageContentSIP(In sc:SemanticCapabilities)
#doApply(In match:MatchResult , In result:ArrayList , In sr:SemanticRepresentation):ArrayList
```

```
<<comment>>
super(sc, "(B ??myself (image-content ??i ??c))");
```

```
<<comment>>
potentiallyAddBehaviour(new OneShotBehaviour() {
    public void action() {
        myGui.displayPhoto(((Constant)match.getTerm("c")).byteValue());
    }});
return result;
```

- **method apply**
  - consumes / produces SRs
  - returns **null** if not applicable
  - add activities with method **potentiallyAddBehaviour**
 - update the belief base with **potentiallyAssertFormula**

- **method apply first matches the input SR with a pattern**
  - the SIP application is then specified in method **doApply**
  - if not applicable, **return null**
  - if no SR to produce, **return result**

research & development

France Telecom Group

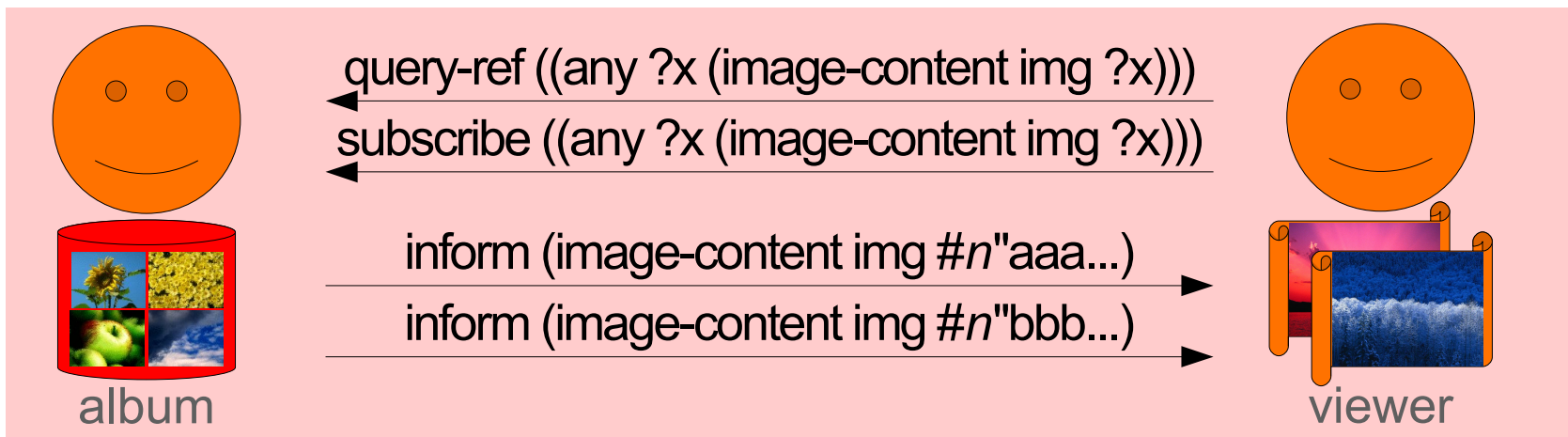# album application – exercise 2 implementing an applicative SIP

■ add a SIP to the viewer agent to display received pictures
  - use the provided implementation of the ViewerGUI interface
  - define the inner class ViewerSemanticCapabilities, instantiate it within the viewer constructor
  - overload the setupSemanticInterpretationPrinciples() method
  - create an ApplicationSpecificSIPAdapter, which adds a OneShotBehaviour that calls ViewerGUI.displayPhoto(byte[])

query-ref ((any ?x (image-content img ?x)))

inform (image-content img #*n*"...)

album

viewer
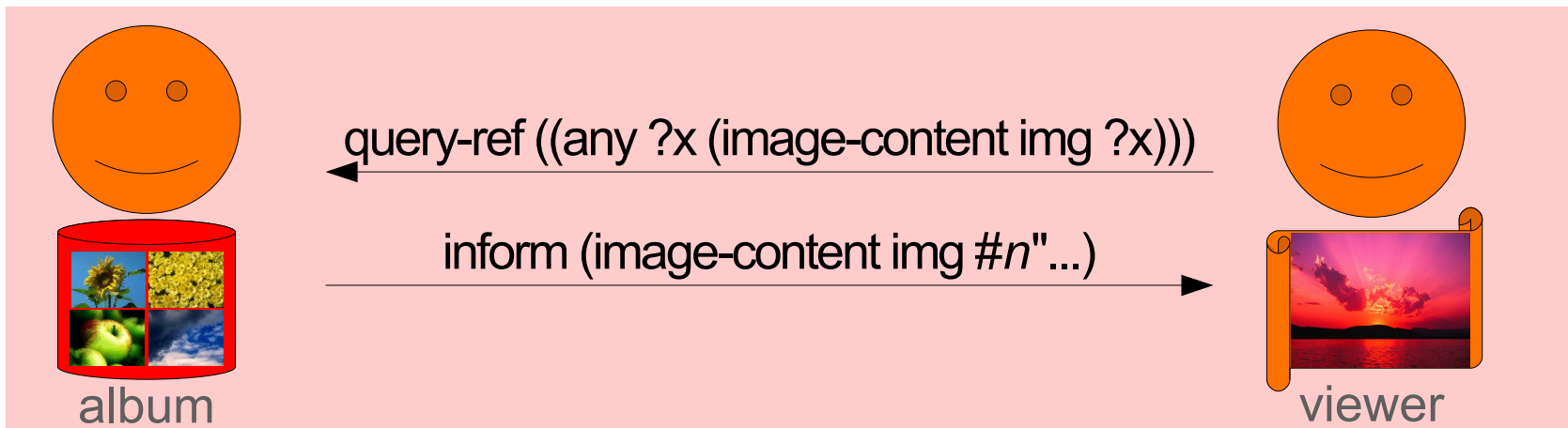
album application – exercise 2.bis implementing a subscribe

- within the setup() method of the album agent
  - add a TickerBehaviour, which periodically changes the image content (the agent's arguments give the available pictures)
    - 1st implementation: use retractFormula, then interpret
    - 2nd implementation: use interpret on (= (iota ?x (image-content img ?x)) *value*)

- within the setup() method of the viewer agent
  - send a subscribe message, identical to the previously sent query-ref

query-ref ((any ?x (image-content img ?x)))

subscribe ((any ?x (image-content img ?x)))

inform (image-content img #*n*"aaa...)

inform (image-content img #*n*"bbb...)

album

viewer

3 defining semantic agents' beliefs

- belief base generic specifications
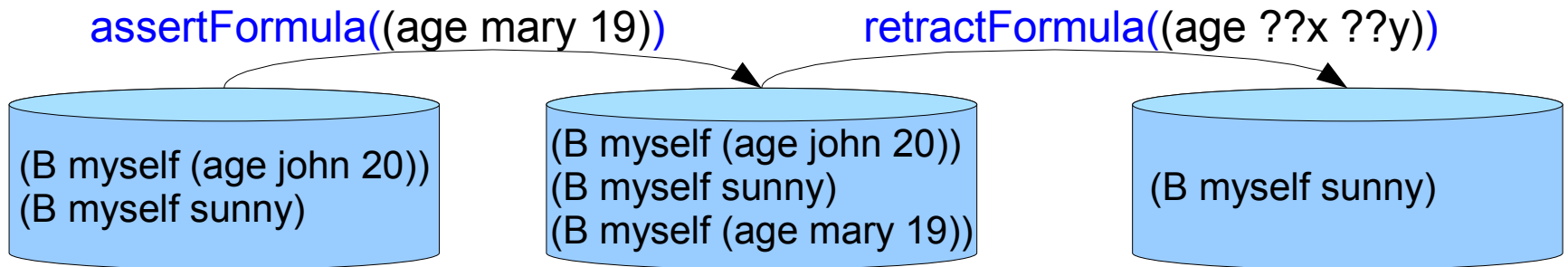- default filter-based implementation

belief base (BB)

- representation of a semantic agent's **internal state**
  - **update** of the internal state
  - **information retrieval** on the internal state
  - **notification of changes** on the internal state
- **subjective** internal state
  - all stored facts are believed by the agent

    (B myself (age john 20)), (I myself (B mary (age john 20)))
  - any fact that is not stored is not believed

    (not (B myself sunny)), (not (B myself (not sunny)))
- **logically consistent** internal state
  - e.g., cannot store both (B myself cold) and (B myself (not cold))

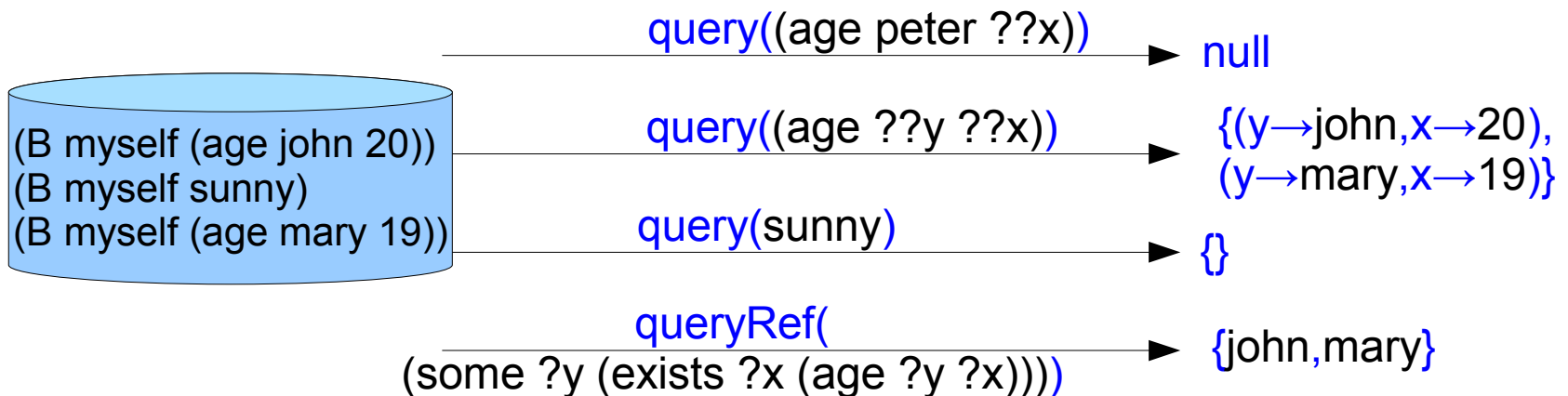- **jade.semantics.kbase.KBase interface**
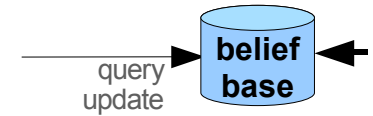
updating beliefs

- **assertFormula(Formula f)**

  consistently assert (B myself f)

- **retractFormula(Formula f)**

  consistently assert (not (B myself f))

  f may include meta-references

assertFormula((age mary 19))   retractFormula((age ??x ??y))

(B myself (age john 20))
(B myself sunny)

(B myself (age john 20))
(B myself sunny)
(B myself (age mary 19))

(B myself sunny)

- such asserted formulas are **not interpreted by the SIPs**
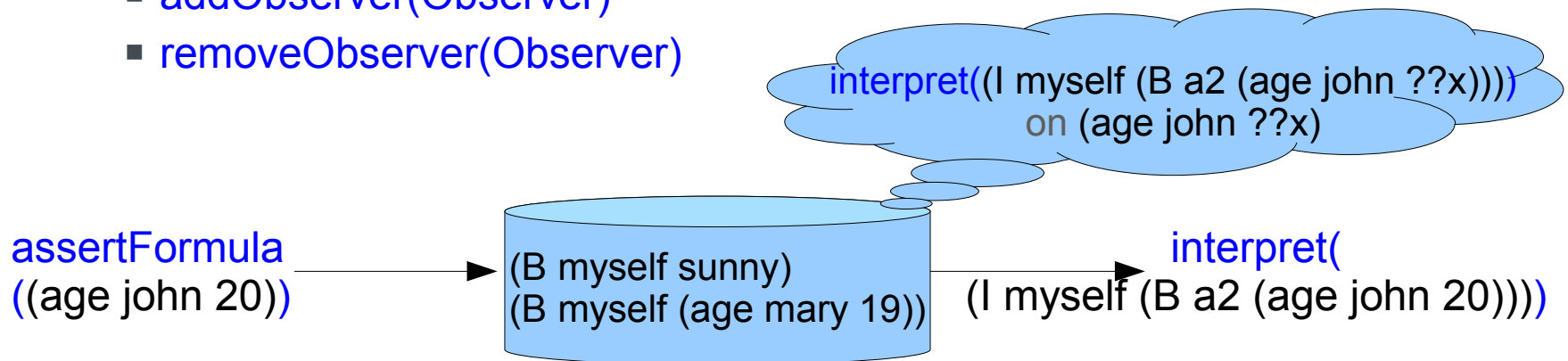 use rather aSemanticCapabilities.interpret(Formula)

querying beliefs

- **query(Formula f): ListOfMatchResults**, returns
  - **null** if (B myself f) is false
  - a list of **MatchResult** objects, which provides the values of the meta-references such that (B myself f) is true

- **queryRef(IdentifyingExpression ire): ListOfTerm**, returns
  - **null** if no object o satisfies (B myself (= o ire))
  - the list of objects o satisfying (B myself (= o ire))


```
                              query((age peter ??x))
                    ──────────────────────────────────────►  null

(B myself (age john 20))      query((age ??y ??x))            {(y→john,x→20),
(B myself sunny)            ──────────────────────────────►   (y→mary,x→19)}
(B myself (age mary 19))      query(sunny)
                           ──────────────────────────────►   {}

                              queryRef(
                    ──────────────────────────────────────►  {john,mary}
                    (some ?y (exists ?x (age ?y ?x))))
```

notification of belief changes

- the Observer interface defines
  - a pattern of formula to monitor
  - a Java code to execute as soon as this pattern becomes believed
- EventCreationObserver implementation
  - the code to execute calls interpret on a given formula ("event")
  - the observer may be permanent or "one shot"
- useful methods of the KBase interface
  - addObserver(Observer)
  - removeObserver(Observer)

interpret((I myself (B a2 (age john ??x))))
on (age john ??x)

assertFormula
((age john 20))

(B myself sunny)
(B myself (age mary 19))

interpret(
(I myself (B a2 (age john 20))))

France Telecom Group

implementing a belief base

■ developers may implement their own BB (according to the KBase interface)
🟠 *hard task!*

■ the JSA comes with a default BB, which provides a good trade-off between efficiency and expressiveness

```
class MySematicCapabilities extends SemanticCapabilities {
    protected KBase setupKbase() {
        KBase base;                        KBase base;
        base = new MyKBase(...);           base = super.setupKbase();
        ...                                ...
        return base;                       return base;
    }
    ...
}
```

research & development

France Telecom Group

default belief base (1/2)

- the jade.semantics.kbase.FilterKBase interface is based on a filter mechanism to manage
  - the storage and consistency of beliefs (assertion operations)
  - the retrieval of beliefs (query operations)
- a set of standard filters handles the generic FIPA-SL predicates and logical operators
- specific filters must be added to manage the storage, the consistency and the retrieval of applicative predicates

```
protected KBase setupKbase() {
        FilterKBase base = (FilterKBase)super.setupKbase();
        base.addKBAssertFilter(myAssertFilter);
        base.addKBQueryFilter(myQueryFilter);
        ...
        return base;
}
```

default belief base (2/2)

- use class jade.semantics.kbase.FiltersDefinition to add a set of filters (assertion filters, query filters or both)

- share filters between several semantic agents

```
class MyFilters extends FiltersDefinition {
    MyFilters() {
        defineFilter(myAssertFilter);
        defineFilter(myQueryFilter);
        ...
    }
}
protected KBase setupKbase() {
    FilterKBase base = (FilterKBase)super.setupKbase();
    base.addFiltersDefinition(new MyFilters());
    return base;
}
}
```

research & development

France Telecom Group

assertion filters
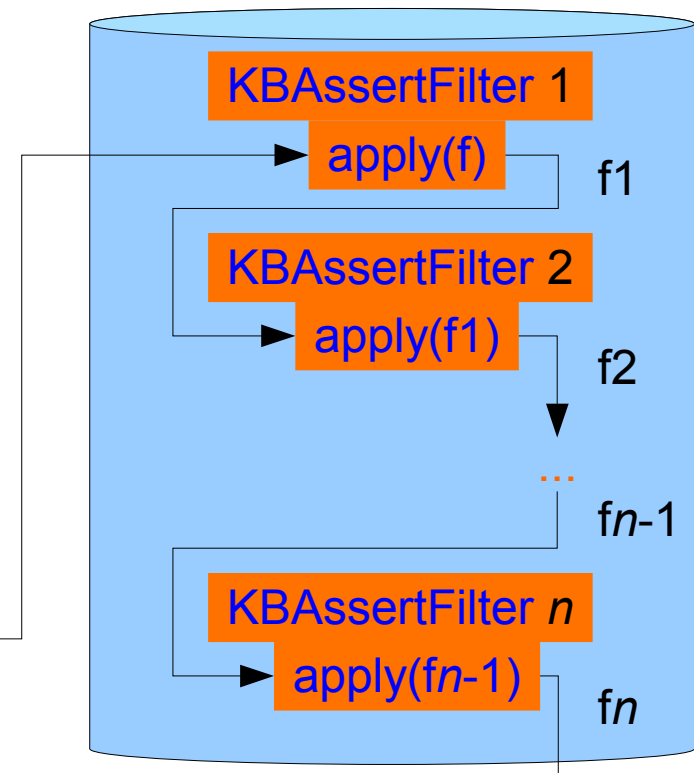
- **jade.semantics.kbase.filter.KBAssertFilter**
  - the apply(Formula) method modifies the formula to assert into the BB
  - if not applicable, return null
  - to block the assertion, return the **true** formula
- **KBAssertFilterAdapter**
  - applicability determined by a pattern
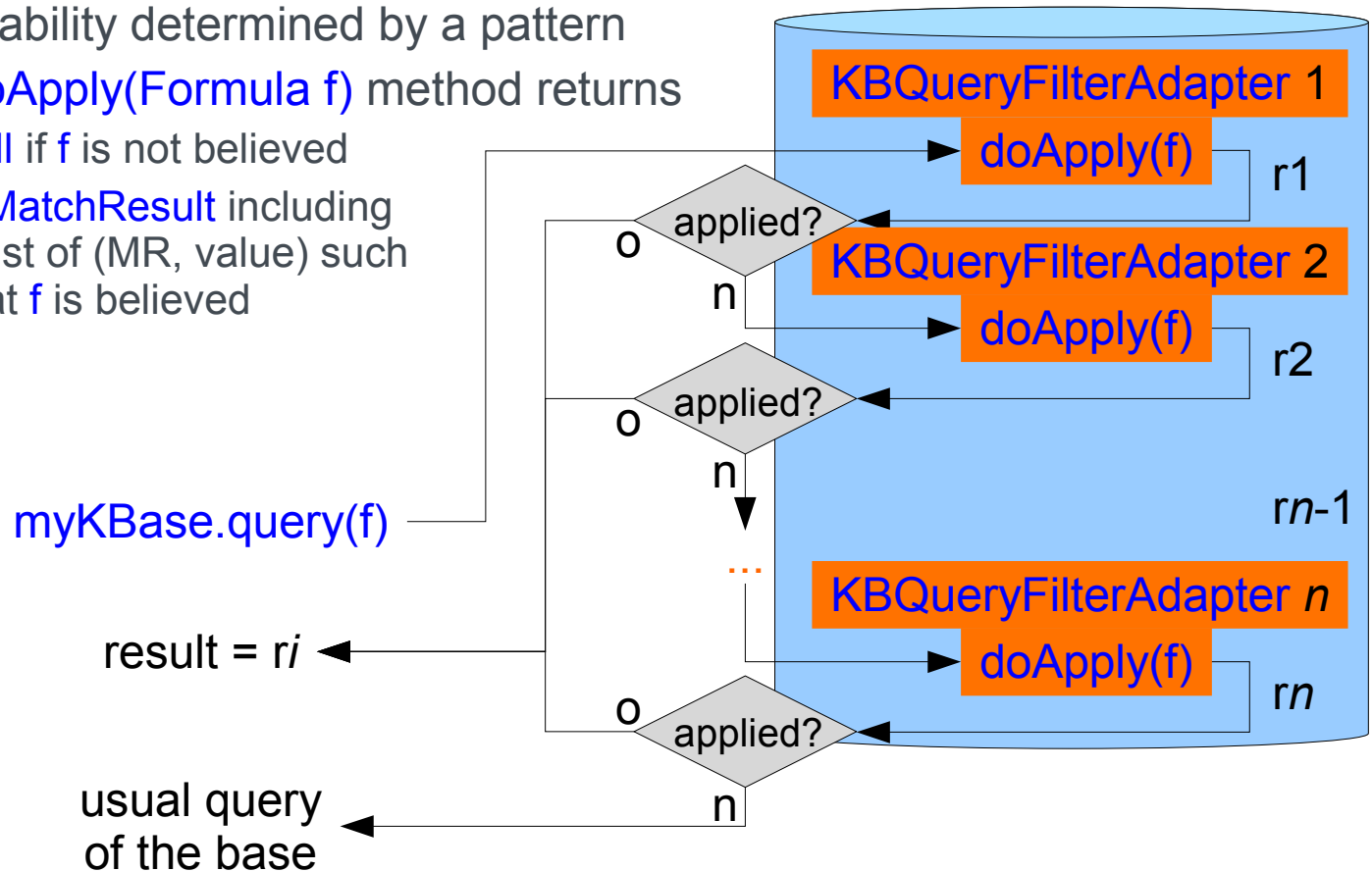 - override the doApply(Formula) method instead of apply

myKBase.assert(f)

KBAssertFilter 1
apply(f)
f1

KBAssertFilter 2
apply(f1)
f2

...

fn-1

KBAssertFilter n
apply(fn-1)
fn

formula actually asserted = fn

query filters

- **jade.semantics.kbase.filter.KBQueryFilterAdapter**
  - applicability determined by a pattern
  - the doApply(Formula f) method returns
    - null if f is not believed
    - a MatchResult including a list of (MR, value) such that f is believed

KBQueryFilterAdapter 1

doApply(f)   r1

applied?   o   n

KBQueryFilterAdapter 2

doApply(f)   r2

applied?   o   n

myKBase.query(f)

rn-1

...

KBQueryFilterAdapter n

doApply(f)   rn

result = ri

applied?   o   n

usual query of the base

query filters: a step further

- the KBQueryFilterAdapter class has limits
  - cannot control the filter applicability (entirely determined by the pattern)
  - cannot return more than one solution (only one MatchResult object returned by the doApply method)
- the KBQueryFilter is more general
  - method apply(Formula f) instead of doApply(Formula f)
  - returns a QueryResult object
    - filterApplied field   true if the filter is applicable, false otherwise
    - result field   null if f is not believed
                       otherwise, list of MatchResult that make f believed
  - method getObserverTriggerPatterns(Formula,Set)
    in order to optimize the notification mechanism of the BB
- improvements of KBQueryFilter expected in future versions

research & development

album application – exercise 3 coding a query filter

- within the album agent
  - remove the TickerBehaviour and the content of the setup() method
  - create a KBQueryFilterAdapter, which reads the content of queried pictures from their URL and not from the BB
- within the viewer agent
  - remove the subscribe sending
  - fill the query-ref content from the URL given by the agent's argument

query-ref ((any ?x (image-content file://... ?x)))

inform (image-content file://... #*n*"...)
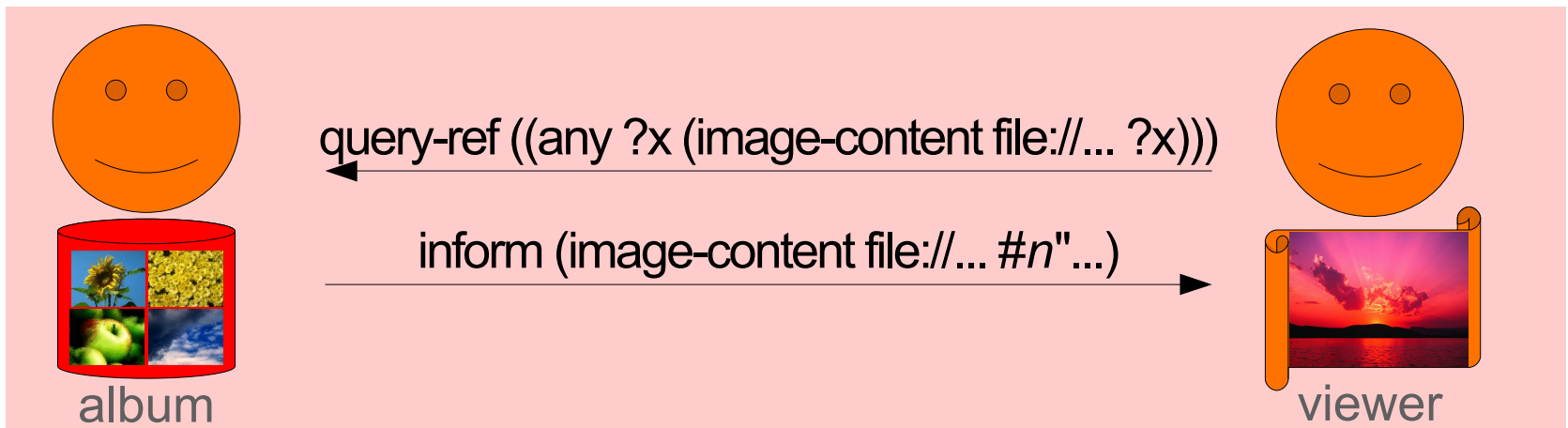
album

viewer

album application – exercise 3.bis coding an assertion filter

■ within the viewer agent

  ▪ create a KBAssertFilterAdapter, which prevents actual assertion of image contents into the BB

■ request the viewer agent with a dummy agent
 to check the former knows no image content any longer

query-ref ((any ?x (image-content ...)))

inform (image-content file://... #*n*"...)

album

viewer

inform ???

JADE dummy agent
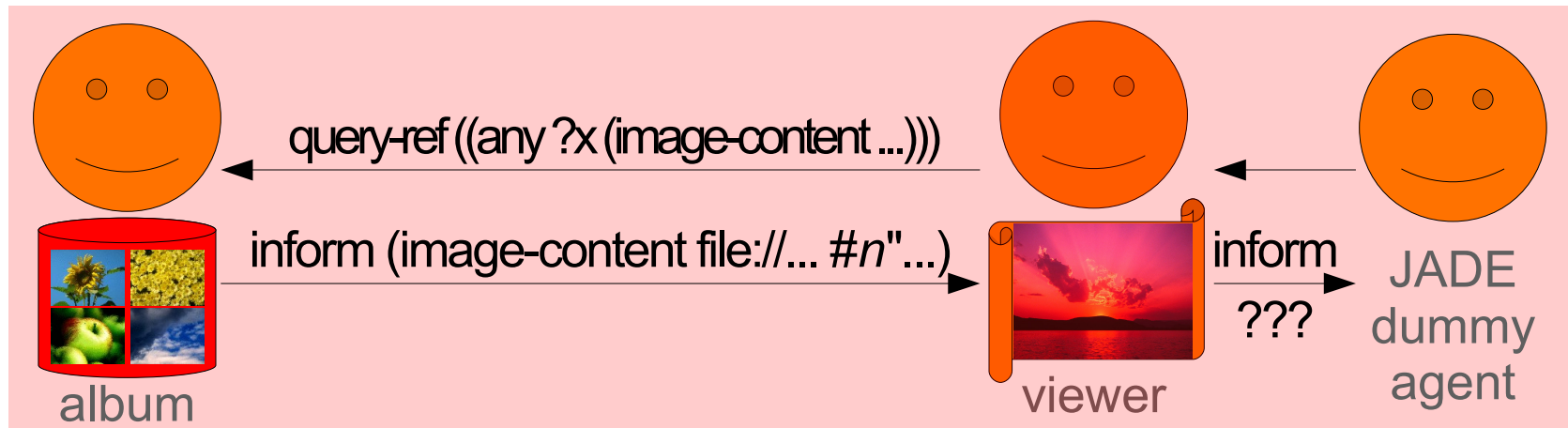
4 defining semantic agents' behaviour

- semantic actions
- customizing standard SIPs
- using SIP adapters

research & development

France Telecom Group

semantic actions

■ formal representations of a semantic agent's elementary "know-how"

  ■ feasibility precondition (SL formula): must be true **just before** the action performance *if not true, the action is considered not feasible and any attempt to perform it fails*

  ■ postcondition (SL formula): will be true **just after** the action performance *if the action is successfully performed, the postcondition is asserted into the BB*

  ■ body (JADE behaviour): "concrete" code to perform the action

■ stored in the SemanticActionTable of each semantic agent

  ■ includes all FIPA-ACL communicative acts (by default)

  ■ plus application-specific actions (to be defined by developers)
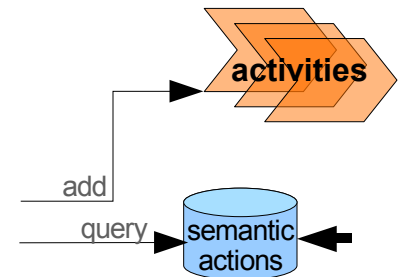
application-specific semantic actions

- extend semantic agents' "know-how"
- used by planning-related SIPs
    - standard ones: ActionPerformance, RationalityPrinciple
    - applicative ones: subclasses of PlanningSIPAdapter
- coding

```java
class MySemanticCapabilities extends SemanticCapabilities {
    protected SemanticActionTable setupSemanticAction() {
        SemanticActionTable table=super.setupSemanticAction();
        table.addSemanticAction(myAction);
        ...
        return table;
    }
    ...
}
```

defining applicative semantic actions

- **construct OntologicalAction with**
  - an action expression, which
    - specifies the **pattern** of functional term that represents the action
    - may include MRs

    (lock :what ??o (::? :delay ??d))
  - two SL formulas, which
    - specify a precondition and a postcondition
    - may include the MRs occurring in the action expression (if needed, ??actor represents the actor of the action)

    (owns-a-key ??actor ??o)

    (locked ??o)
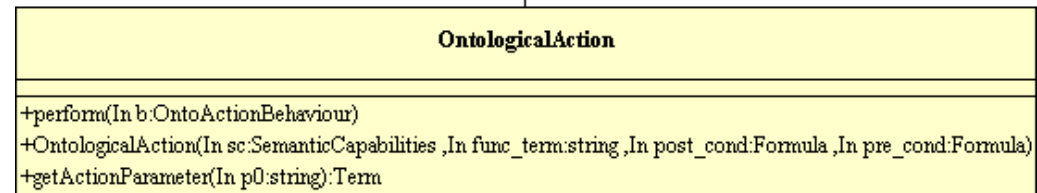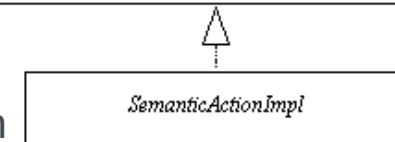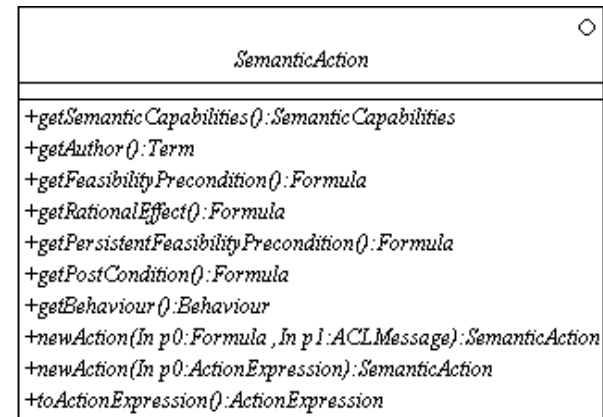
- **when an action is performed (scheduled in a JADE behaviour)**
  - the BB is queried **before** performance to check the **precondition**
  - the **postcondition** is asserted into the BB **after** performance

SemanticAction

+getSemanticCapabilities():SemanticCapabilities
+getAuthor():Term
+getFeasibilityPrecondition():Formula
+getRationalEffect():Formula
+getPersistentFeasibilityPrecondition():Formula
+getPostCondition():Formula
+getBehaviour():Behaviour
+newAction(In p0:Formula ,In p1:ACLMessage):SemanticAction
+newAction(In p0:ActionExpression):SemanticAction
+toActionExpression():ActionExpression

SemanticActionImpl

OntologicalAction

+perform(In b:OntoActionBehaviour)
+OntologicalAction(In sc:SemanticCapabilities ,In func_term:string ,In post_cond:Formula ,In pre_cond:Formula)
+getActionParameter(In p0:string):Term

defining the body of a semantic action

- **SemanticBehaviour** maintain a state of performance…
  - when the performance fails
    - **FEASIBILITY_FAILURE** unsatisfied precondition
    - **EXECUTION_FAILURE** failure during execution
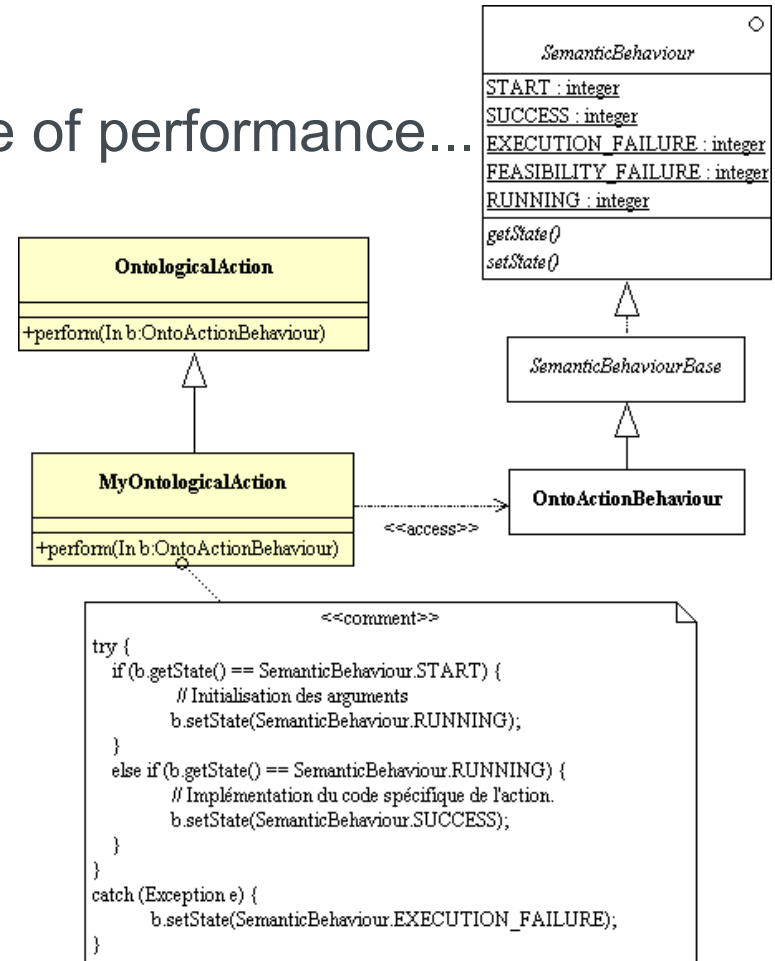 - when it succeeds: **SUCCESS**
- …to manage the execution of semantic actions (inc. comm. acts)
- Case of applicative actions
  - the **action()** method of **OntoActionBehaviour** is defined upon the **perform()** method of the corresponding **OntologicalAction**
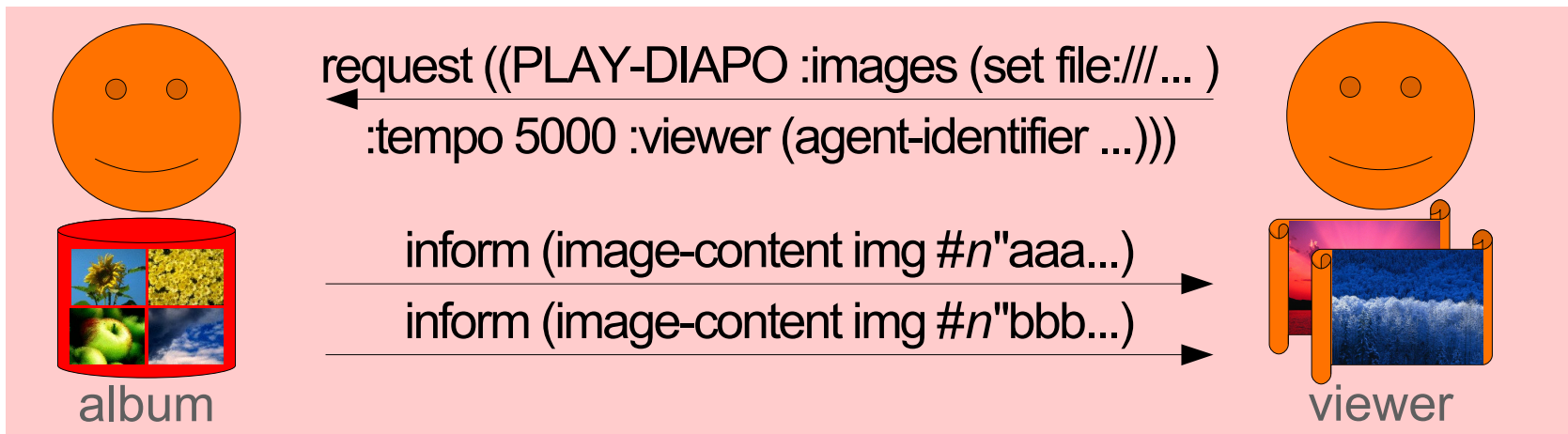 - same programming style as the **action()** method of JADE **Behaviour**

SemanticBehaviour

START : integer
SUCCESS : integer
EXECUTION_FAILURE : integer
FEASIBILITY_FAILURE : integer
RUNNING : integer

getState()
setState()

OntologicalAction

+perform(In b:OntoActionBehaviour)

MyOntologicalAction

+perform(In b:OntoActionBehaviour)

SemanticBehaviourBase

OntoActionBehaviour

<<access>>

<<comment>>

```
try {
    if (b.getState() == SemanticBehaviour.START) {
        // Initialisation des arguments
        b.setState(SemanticBehaviour.RUNNING);
    }
    else if (b.getState() == SemanticBehaviour.RUNNING) {
        // Implémentation du code spécifique de l'action.
        b.setState(SemanticBehaviour.SUCCESS);
    }
}
catch (Exception e) {
    b.setState(SemanticBehaviour.EXECUTION_FAILURE);
}
```

research & development

album application – exercise 4 coding a diaporama semantic action

- within the album agent, implement an ontological action

    consisting in sending to another agent (:viewer parameter) a set of pictures to display (:images parameter), with an optional delay (:tempo parameter) between pictures

- within the viewer agent

  - send a request on this action instead of the previous query-ref
  - the list of pictures is read from the agent's arguments

request ((PLAY-DIAPO :images (set file:///...) :tempo 5000 :viewer (agent-identifier ...)))

inform (image-content img #*n*"aaa...)

inform (image-content img #*n*"bbb...)

album

viewer

customizing standard SIPs (1/2)

- most of standard SIPs may be customized
  - add an instance of the proper SIP adapter to the agent's SIP table
  - pass proper arguments to the constructor (generally a SL pattern to match) and/or override the proper method (generally doApply())
  - see the jade.semantics.interpreter.sips.adapters package

```
protected SemanticInterpretationPrincipleTable setupSemanticInterpretationPrinciples() {
        table = super.setupSemanticInterpretationPinciples();
        table.addSemanticInterpretationPrinciple(
                new PlanningSIPAdapter(this, "??goal") {
                        public ActionExpression doApply(...) {
                                ...
                        }
                });    ...
        return table;
}
```

research & development

France Telecom Group

customizing standard SIPs (2/2)

- the doApply() method of most of the SIP adapters
  - provides **pre-computed arrays of SR** to return, corresponding to the various possible results of the SIP
  - e.g., the BeliefTransferSIPAdapter provides 2 pre-computed results: one to accept the controlled belief and one to reject it (see below)

- when the result of the SIP cannot be decided at once
  - return an empty array of SR (to "absorb" the input SR)
  - install a proper behaviour, which
    - makes the decision (for example, by interacting with other agents)
    - finally interprets the pre-computed result corresponding to the made decision

```
ArrayList doApply(MatchResult matchFormula, MatchResult matchAgent,
                ArrayList acceptResult, ArrayList refuseResult,
                SemanticRepresentation sr) {
        potentiallyAddBehaviour(new MyDecisionBehaviour(...));
        return new ArrayList();
}
```

research & development

belief transfer SIP adapter

- controls the adoption of beliefs coming from other agents
- constructor
    - formulaToBelievePattern: the pattern of belief to control
    - originatingAgentPattern: the pattern of agent originating the belief to control (beliefs from other agents will not be controlled by the SIP)

    Optional arguments (set to true by default)

    - notPattern: if true, also controls the adoption of (not *formulaPattern*)
    - allPattern: if true, also controls (= (all ??X *formulaPattern*) (set))
            (pattern used to retract all instances of the belief)
- doApply() method
    - the first 2 arguments give the results of the matching of the belief to control and the originating agent against the specified patterns
    - acceptResult: array of SR to return if the belief can be adopted
    - refuseResult: array of SR to return if the belief must not be adopted

intention transfer SIP adapter

- controls the adoption of intentions of other agents
- constructor
  - goalPattern: the pattern of goal (to intend) to control
  - agentPattern: the pattern of external agent intending the goal to control (intentions of other agents will not be controlled by the SIP)

  Optional argument (set to true by default)
  - feedBackRequired: if true, generates a feedback towards the external agent
    - intention adopted: acknowledges the adoption, then the goal achievement
    - intention not adopted: acknowledges the adoption refusal
- doApply() method
  - the first 2 arguments give the results of the matching of the goal to control and the originating agent against the specified patterns
  - acceptResult: array of SR to return if the goal can be intended
  - refuseResult: array of SR to return if the goal must not be intended

research & development

planning SIP adapter

- computes a plan to reach an intended goal
- constructor
  - goalPattern: the pattern of goal, for which the SIP may find a plan
- doApply() method
  - returns an action expression representing the computed plan (instead of an array of SR) – if null, the SIP is considered not applicable
  - matchResult: result of the matching of the intended goal against the specified pattern
- the returned plan is performed
  - if it ends out to be not feasible, the next matching planning SIP in the SIP table is tried (in the order of the SIP table) to find a new plan
- several SIPs can be defined (for different goals as well as for the same goal)

research & development

CFP SIP adapter (1/2)

- controls the answer to a CFP

- a CFP expects 2 content elements
  - a requested action (expressed as an action expression)
  - a condition (expressed as an IRE)

- default adapter constructor (with no argument)
  - automatically answers CFPs by evaluating the condition **independently from the action** (this is a simplifying assumption)

- regular constructor, to control specific patterns of CFP
  - ireQuantifierPattern: the pattern of the IRE quantifier (given as a constant, see QueryRefPreparationSIPAdapter.ANY/IOTA/SOME/ALL)
  - ireVariablesPattern: the pattern of the IRE quantified variables
  - conditionPattern: the pattern of the condition formula
  - actPattern: the pattern of action
  - agentPattern: the pattern of agent (AID) originating the CFP

CFP SIP adapter (2/2)

- **prepareProposal()** method
  - works along the same principle as the **doApply()** methods
  - the first 4 arguments give the elements defining the CFP to control,
  - the following 3 arguments give the results of the matching of these elements against the specified patterns
  - **result**: array of SR to return if the SIP is not absorbent (or to interpret later if the SIP is absorbent and delays its processing)

- this method is expected to set up in the belief base proper values of the condition to perform the requested actions
  - use the **assertProposals()** method to do so
  - the first 4 arguments give the elements defining the condition/action
  - the last argument gives the list of proper values for these condition/action

research & development

France Telecom Group

5 concluding remarks and synthesis

research & development

France Telecom Group

what about protocols, conversation-id, ...?

- **■** semantic agents **genuinely interpret** received messages

  - **▪** such an interpretation is **consistent** with FIPA interaction protocols

    handling complex protocols, such as CFP, consists
    in specializing the proper SIP adapter(s)

  - **▪** such an interpretation is **more flexible**, so that agents may naturally engage in intermediate exchanges, without the need of making them explicit in a protocol specification

- **■** no need to make the used protocol explicit

- **■** no need to make the conversation-id explicit

research & development

France Telecom Group

and ontologies?

- there is no explicit support for a specific ontology model
- developers have to define the way of representing classes, properties, instances, ... by SL expressions. For example:
  - SL functional terms may represent frames with slots
    - (Person :name john :age 20)
    - use the setParameter(String,Term) and getParameter(String,Term) methods to handle directly slot values
    - see also the jade.semantics.kbase.FunctionalTable class (experimental)
  - SL predicates generally represent properties
    - (hasFather i1 i2), (is_a i1 Person), (subclass Mother Female), ...
- no ContentManager, because the JSA automatically analyses the content of incoming messages
- under study: a "mapper" between JADE ontologies and SL patterns (for reusing some JADE features with the JSA)

differences between JADE and JSA

- an empty JSA agent is not so empty
  it can react properly to many requests

- SL is the "internal" language to programme JSA agents
  take advantage of the SL pattern mechanisms

- Basic programming advices

  1. **Never use** the receive() method, avoid the send() method

  2. Programme your agent's **observable behaviours** through **SIPs**

  3. Programme your agent's **skills** through **semantic actions**

  4. Reasoning on **facts** and fact storage are managed by the **belief base**

research & development

France Telecom Group