Software agent computing

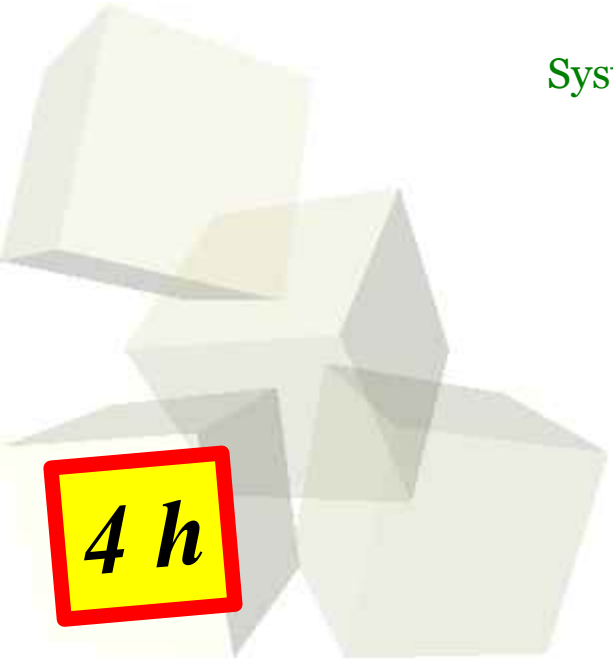
3rd laboratory activities
at Warsaw University of Technology

Maciej Gawinecki

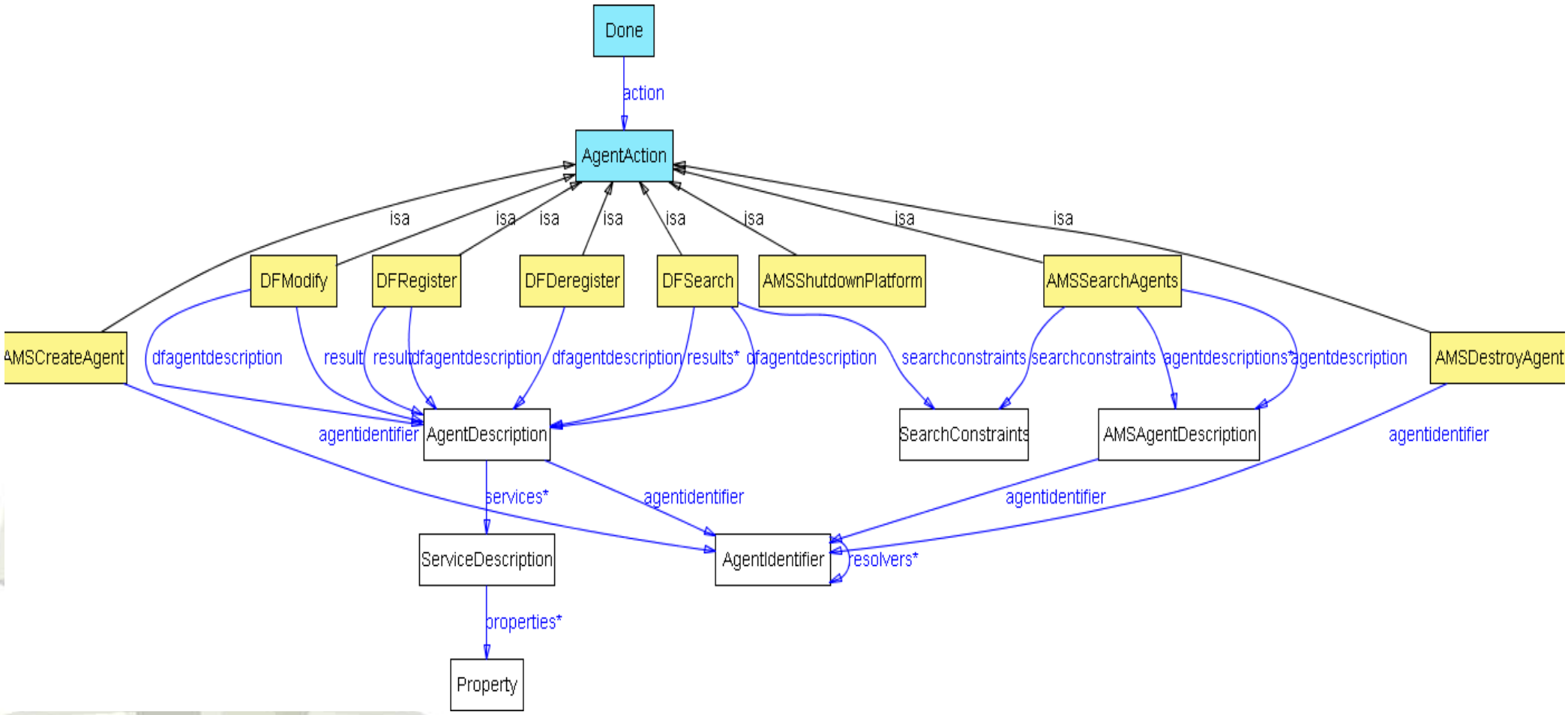
Systems Research Institute, Polish Academy of Sciences

maciej.gawinecki@ibspan.waw.pl

http://www.ibspan.waw.pl/~gawinec

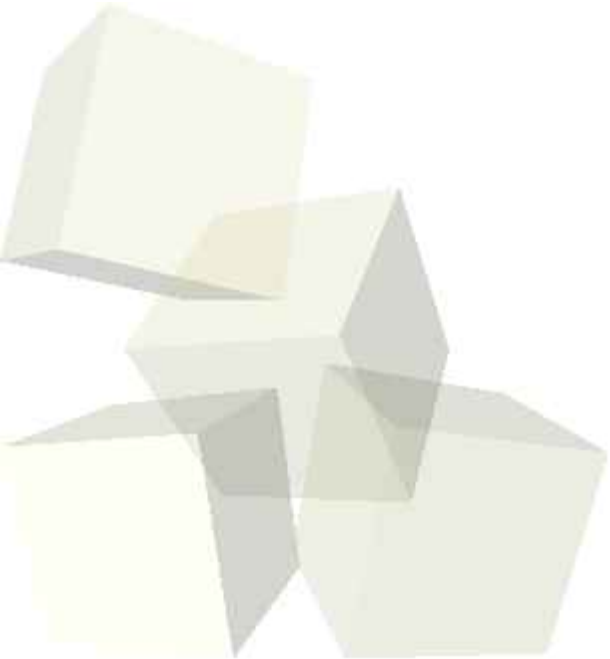


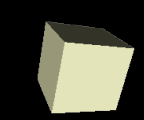
FIPA AMS Ontology



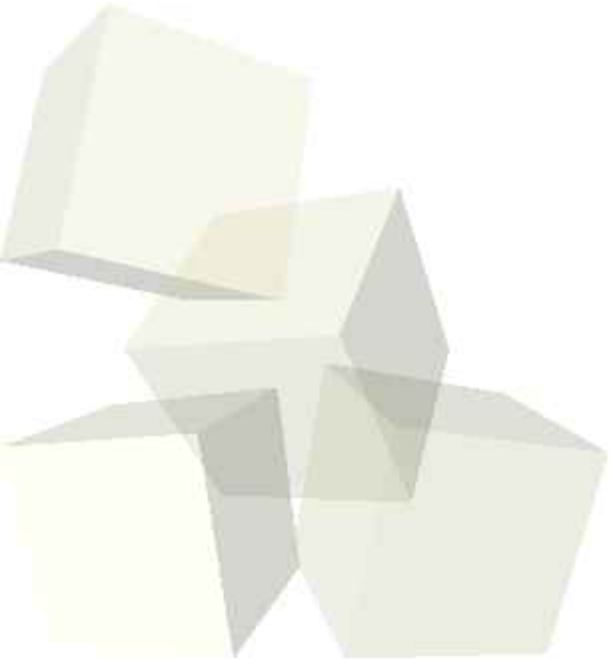
Agenda

- Combining protocols and ontology by JADE agents
- Goal-directed agents build with use Jadex BDI reasoning engine



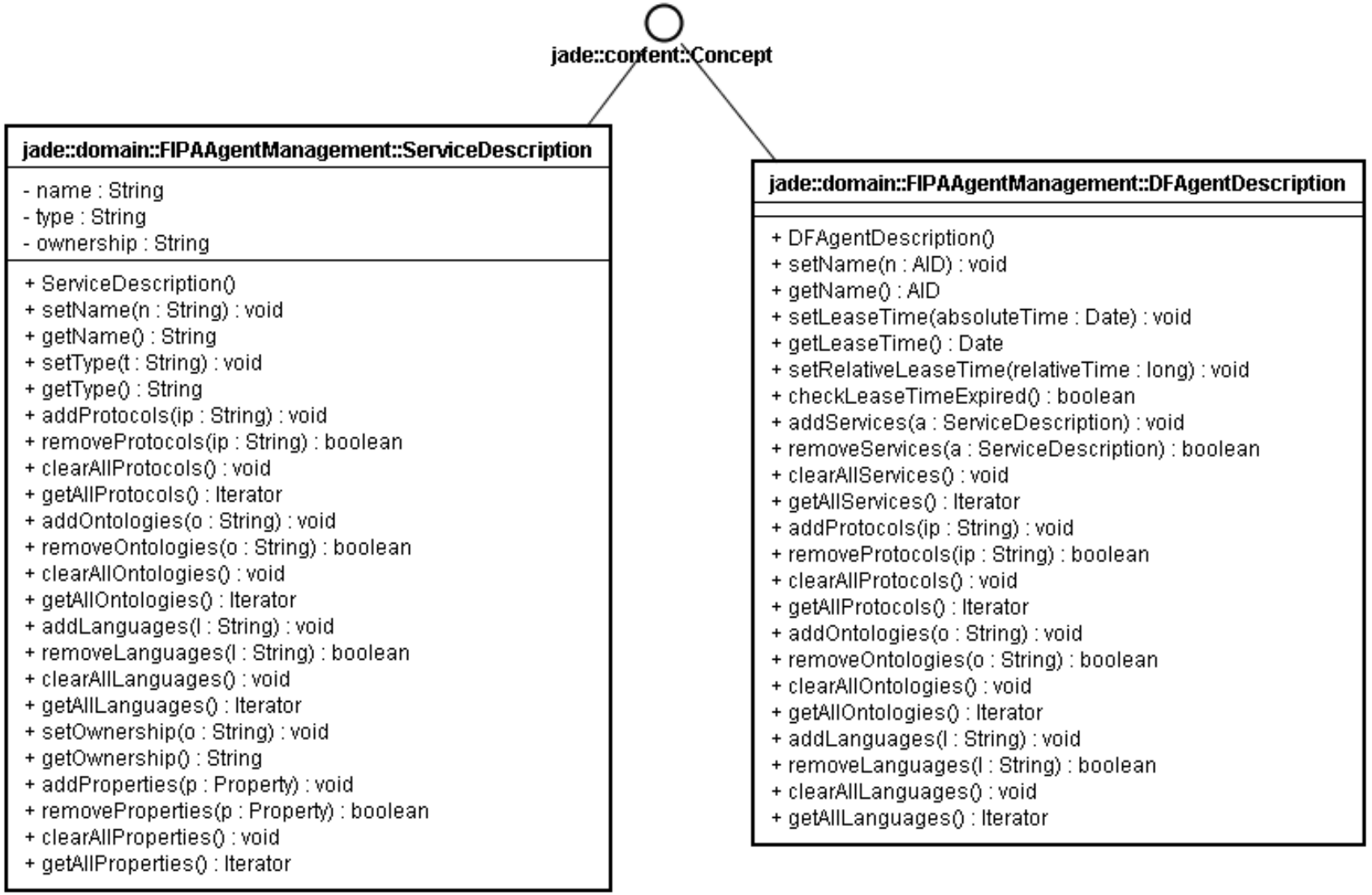


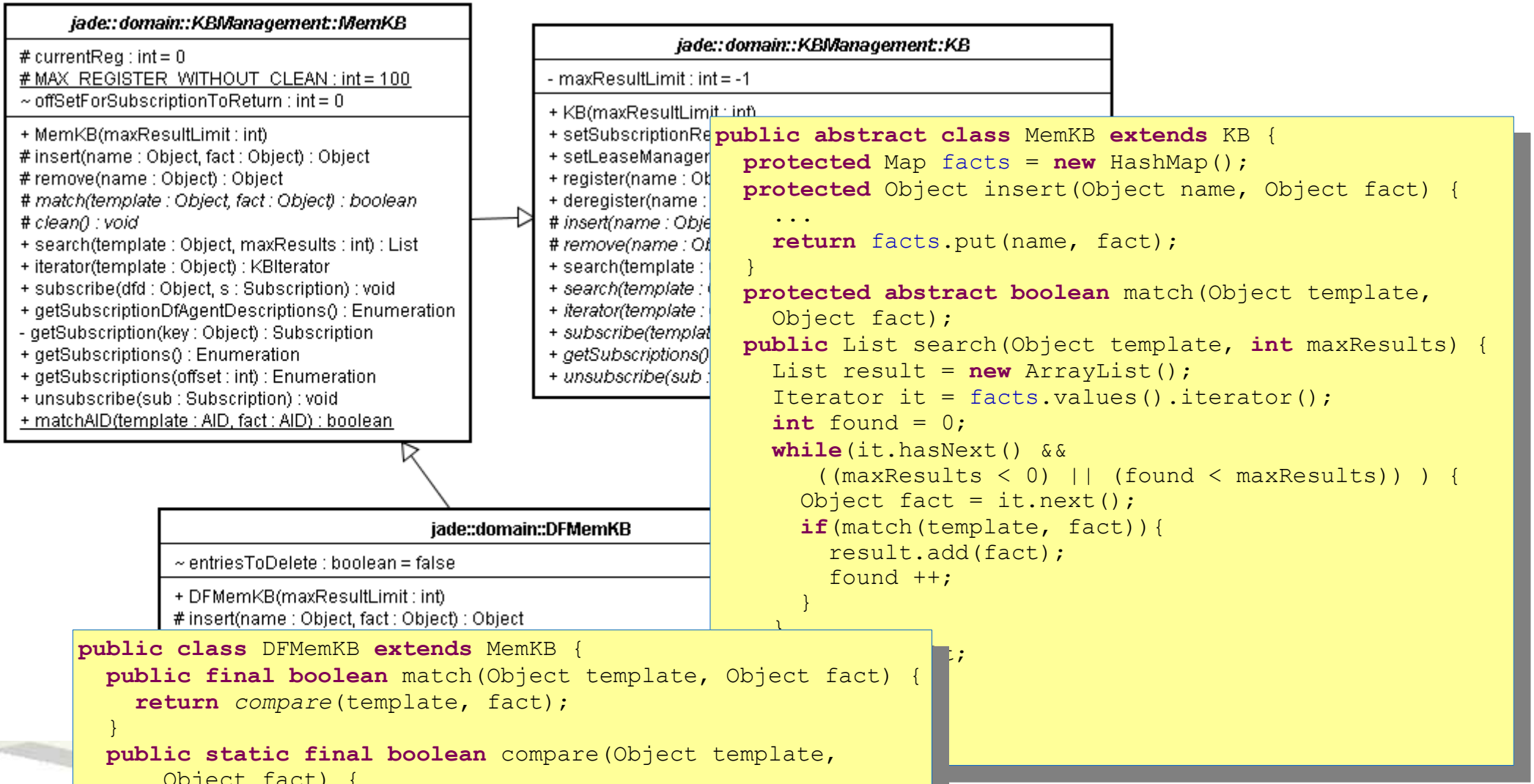
Ontology and protocols in JADE



Utilizing in ontology

1. Creating JADE-compliant ontology
   - Tools:
     - Protege
   - Templates:
2. Converting ontology into Java beans
   - Tools:
     - Jadex Ontology Beananizer
     - Acklin's Ontology BeanGenerator
3. Creating knowledge-base
   1. Tools:
      1. JADE Semantic Add-On
4. Communicating and processing knowledge
   1. Tools:
      1. JADE
      2. JADE Semantic Add-On


jade::content::Concept
jade::domain::FIPAAgentManagement::ServiceDescription
- name : String
- type : String
- ownership : String
+ ServiceDescription()
+ setName(n : String) : void
+ getName() : String
+ setType(t : String) : void
+ getType() : String
+ addProtocols(ip : String) : void
+ removeProtocols(ip : String) : boolean
+ clearAllProtocols() : void
+ getAllProtocols() : Iterator
+ addOntologies(o : String) : void
+ removeOntologies(o : String) : boolean
+ clearAllOntologies() : void
+ getAllOntologies() : Iterator
+ addLanguages(l : String) : void
+ removeLanguages(l : String) : boolean
+ clearAllLanguages() : void
+ getAllLanguages() : Iterator
+ setOwnership(o : String) : void
+ getOwnership() : String
+ addProperties(p : Property) : void
+ removeProperties(p : Property) : boolean
+ clearAllProperties() : void
+ getAllProperties() : Iterator
jade::domain::FIPAAgentManagement::DFAgentDescription
+ DFAgentDescription()
+ setName(n : AID) : void
+ getName() : AID
+ setLeaseTime(absoluteTime : Date) : void
+ getLeaseTime() : Date
+ setRelativeLeaseTime(relativeTime : long) : void
+ checkLeaseTimeExpired() : boolean
+ addServices(a : ServiceDescription) : void
+ removeServices(a : ServiceDescription) : boolean
+ clearAllServices() : void
+ getAllServices() : Iterator
+ addProtocols(ip : String) : void
+ removeProtocols(ip : String) : boolean
+ clearAllProtocols() : void
+ getAllProtocols() : Iterator
+ addOntologies(o : String) : void
+ removeOntologies(o : String) : boolean
+ clearAllOntologies() : void
+ getAllOntologies() : Iterator
+ addLanguages(l : String) : void
+ removeLanguages(l : String) : boolean
+ clearAllLanguages() : void
+ getAllLanguages() : Iterator

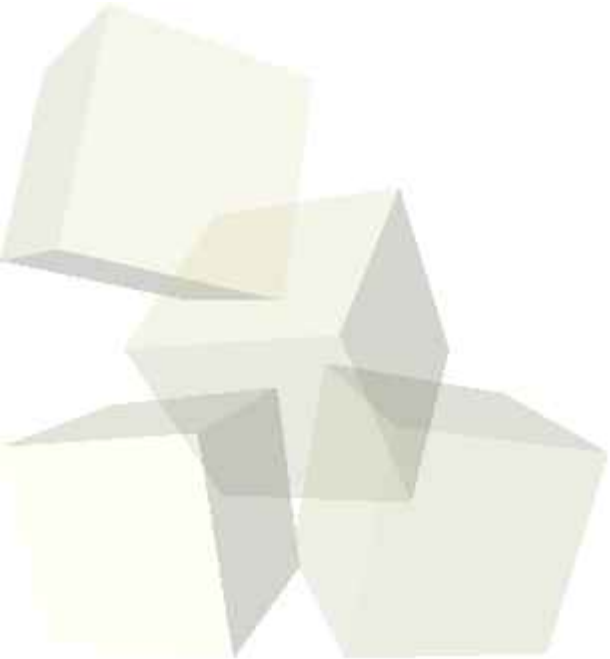



```
public abstract class MemKB extends KB {
  protected Map facts = new HashMap();
  protected Object insert(Object name, Object fact) {
    ...
    return facts.put(name, fact);
  }
  protected abstract boolean match(Object template,
    Object fact);
  public List search(Object template, int maxResults) {
    List result = new ArrayList();
    Iterator it = facts.values().iterator();
    int found = 0;
    while(it.hasNext() &&
       ((maxResults < 0) || (found < maxResults)) ) {
      Object fact = it.next();
      if(match(template, fact)){
        result.add(fact);
        found ++;
      }
    }
```

```
public class DFMemKB extends MemKB {
  public final boolean match(Object template, Object fact) {
    return compare(template, fact);
  }
  public static final boolean compare(Object template,
      Object fact) {
    // Match name
    // Match protocol set
    // Match ontologies set
    // Match languages set
    // Match services set
    return ...;
  }
  ...
}
```

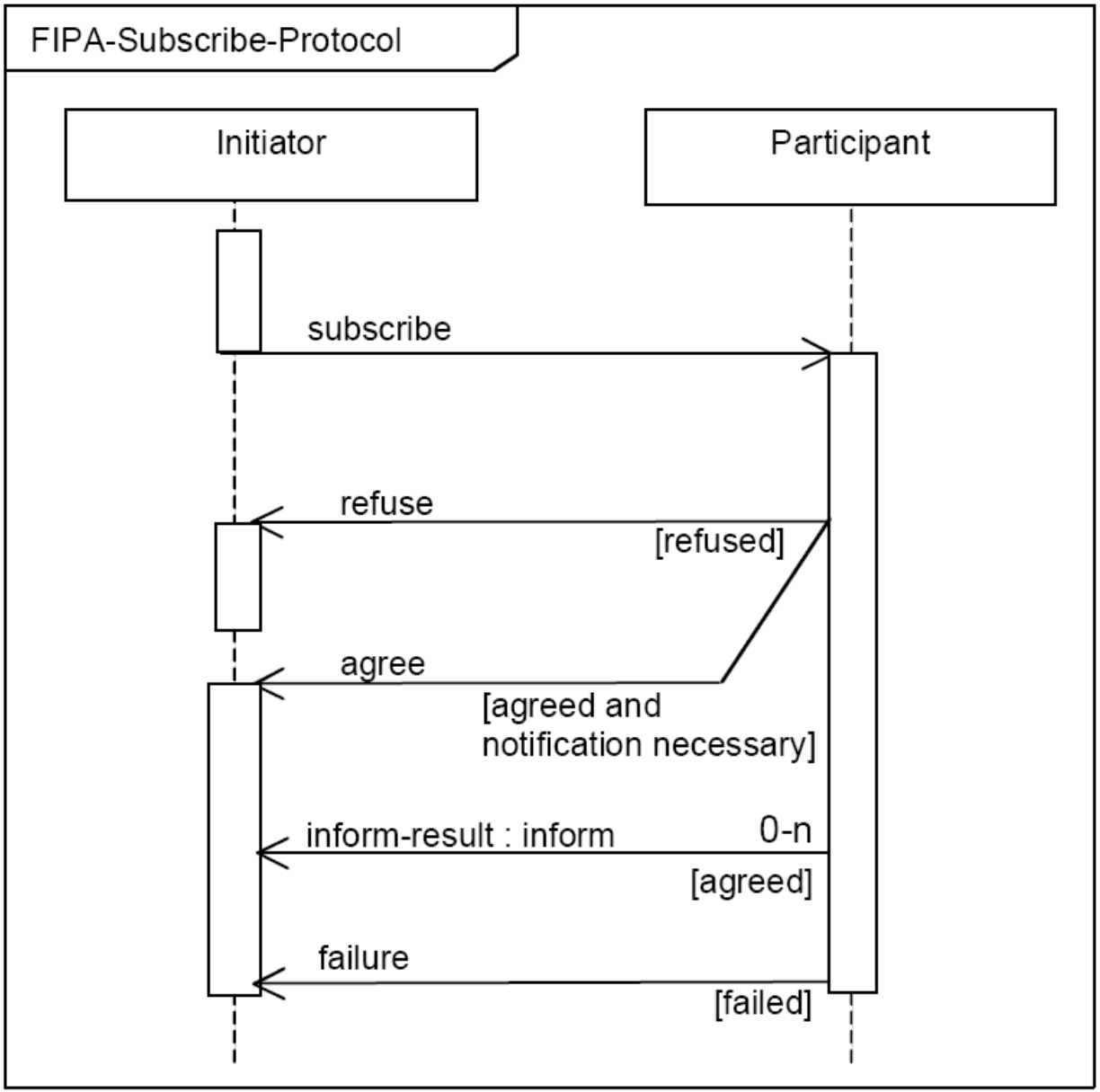
Subscription

- Protocol documentation
- JADE implementation support



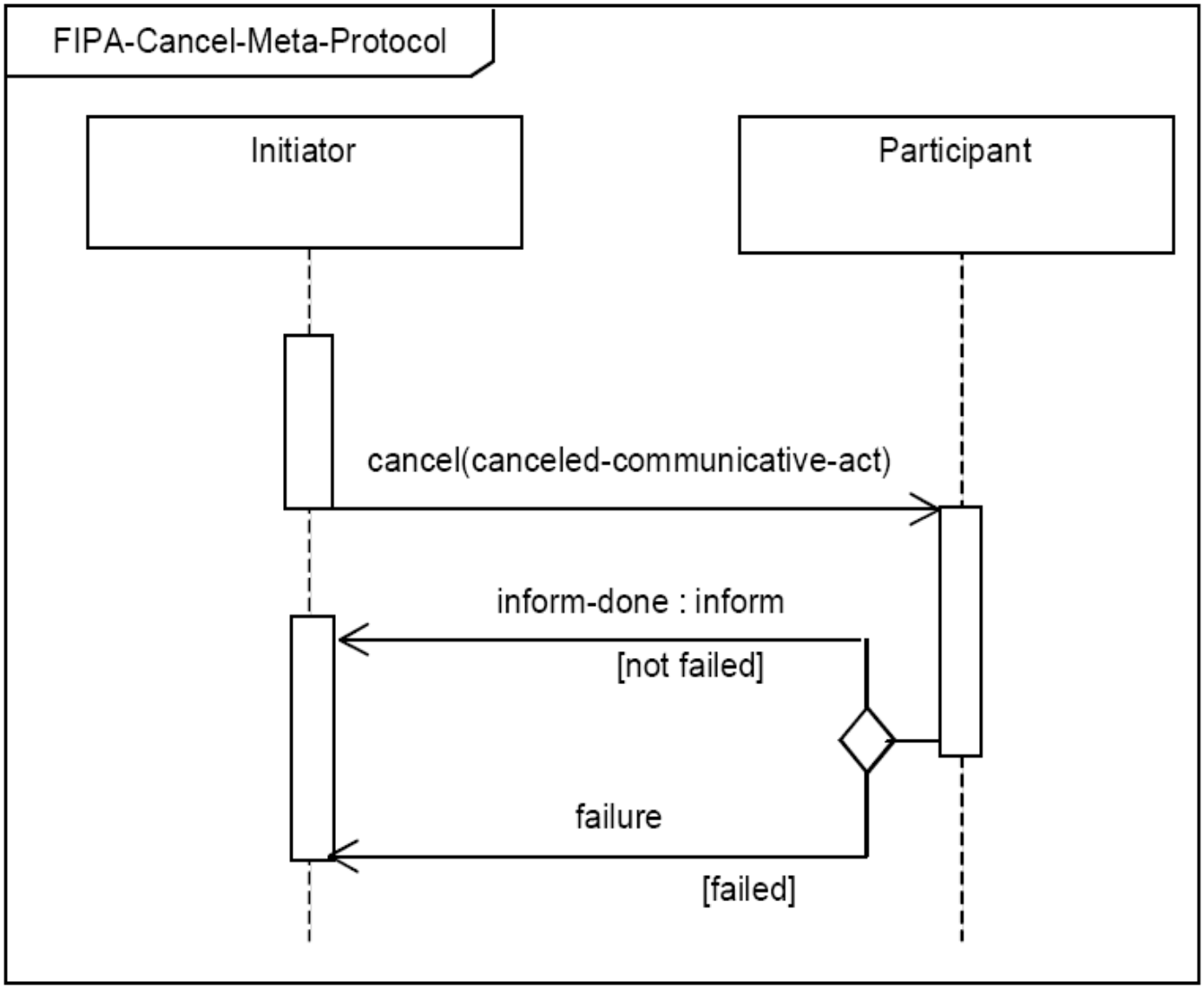
FIPA Subscribe Protocol

- Allows the *Initiator* to send a subscribe message to the *Participant* indicating its desired subscription.
- The Participant:
  - responds to the query request by either accepting (*agree*) or rejecting (*refuse*) the subscription
  - communicates all content matching the *subscriptions condition* using an an *inform* communicative act with a *result* predicate as content



FIPA Cancel Meta Protocol

- The *Participant* continues to send inform-results until either
  - the *Initiator* cancels, communicated by sending a *cancel* message,
  - or the *Participant* experiences a failure, communicated with a *failure* message.



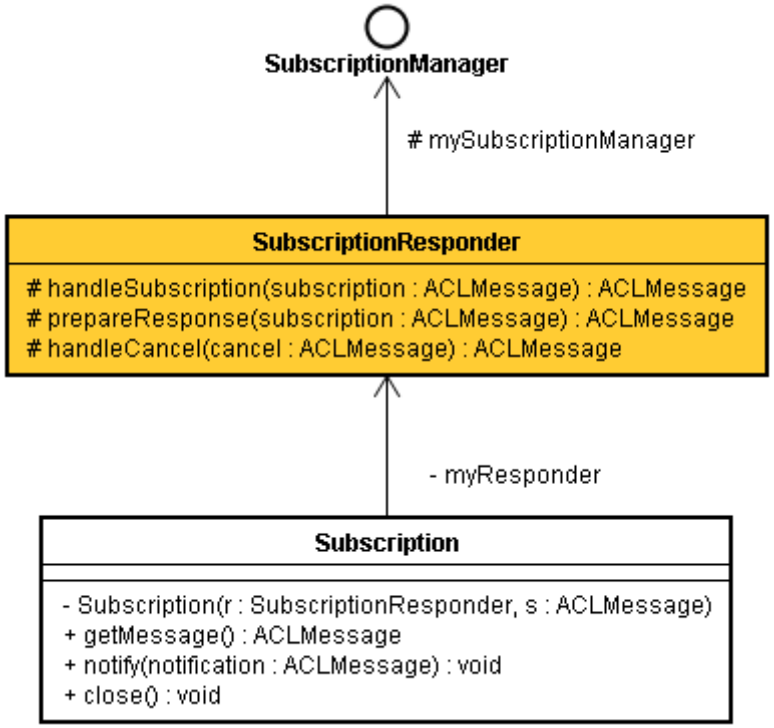
Subscription Initiator behaviour

- The implementation of the interaction provides a set of callback methods to handle each state of the protocol
  - these are called when a certain type of message (based on its communicative act) is received.
    - `protected handleAgree(ACLMessage agree)`
    - `protected handleRefuse(ACLMessage refuse)`
    - `protected handleInform(ACLMessage inform)`

- Method for canceling the subscription:
  - `public cancel(AID receiver, boolean ignoreResponse)`
  - Cancel the subscription to agent receiver. This method retrieves the subscription message sent to receiver and sends a suitable *cancel* message.
  - The content slot of this *cancel* message is filled in by means of the *fillCancelContent()* method.

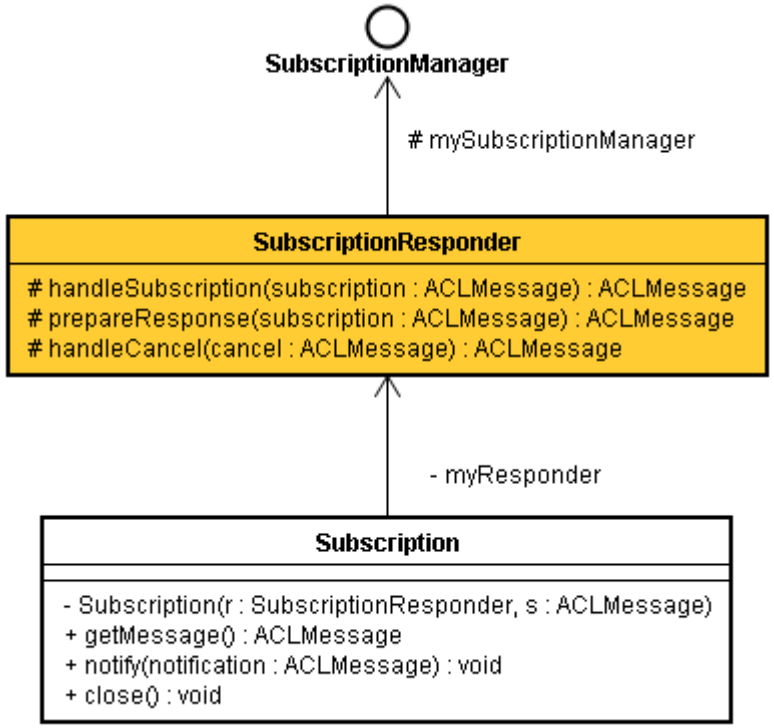
Subscription Responder behaviour

- Implements the *FIPA-Subscribe* interaction protocol from the point of view of a *responder* to subscription message.
- It is very important to pass the right *message template* to its constructor as it is used to select the **`ACLMessage`** to be served.
- Examples
  - **Directory Facilitator Agent** (**`jade.domain.df.java`**)
  - **JMSPubSub Agent** add-on utility



Subscription Manager

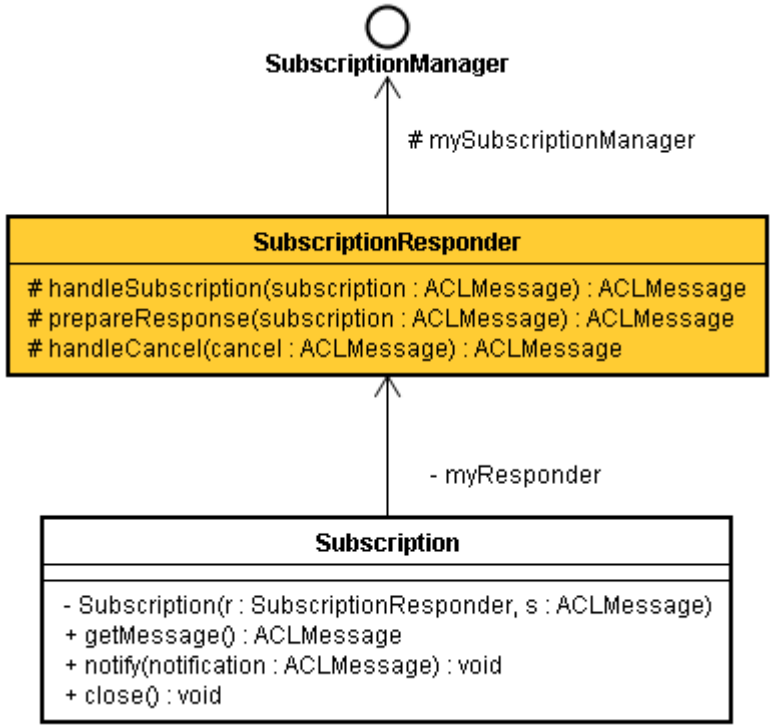
- When a new subscription message arrives, the **`SubscriptionResponder`** invokes the **`register()`** method of its **`SubscriptionManager`**.
- When a cancel message is received the **`deregister()`** method is called.
- The applications **`SubscriptionManager`** is expected to implement the **`register()`** and **`deregister()`** methods.



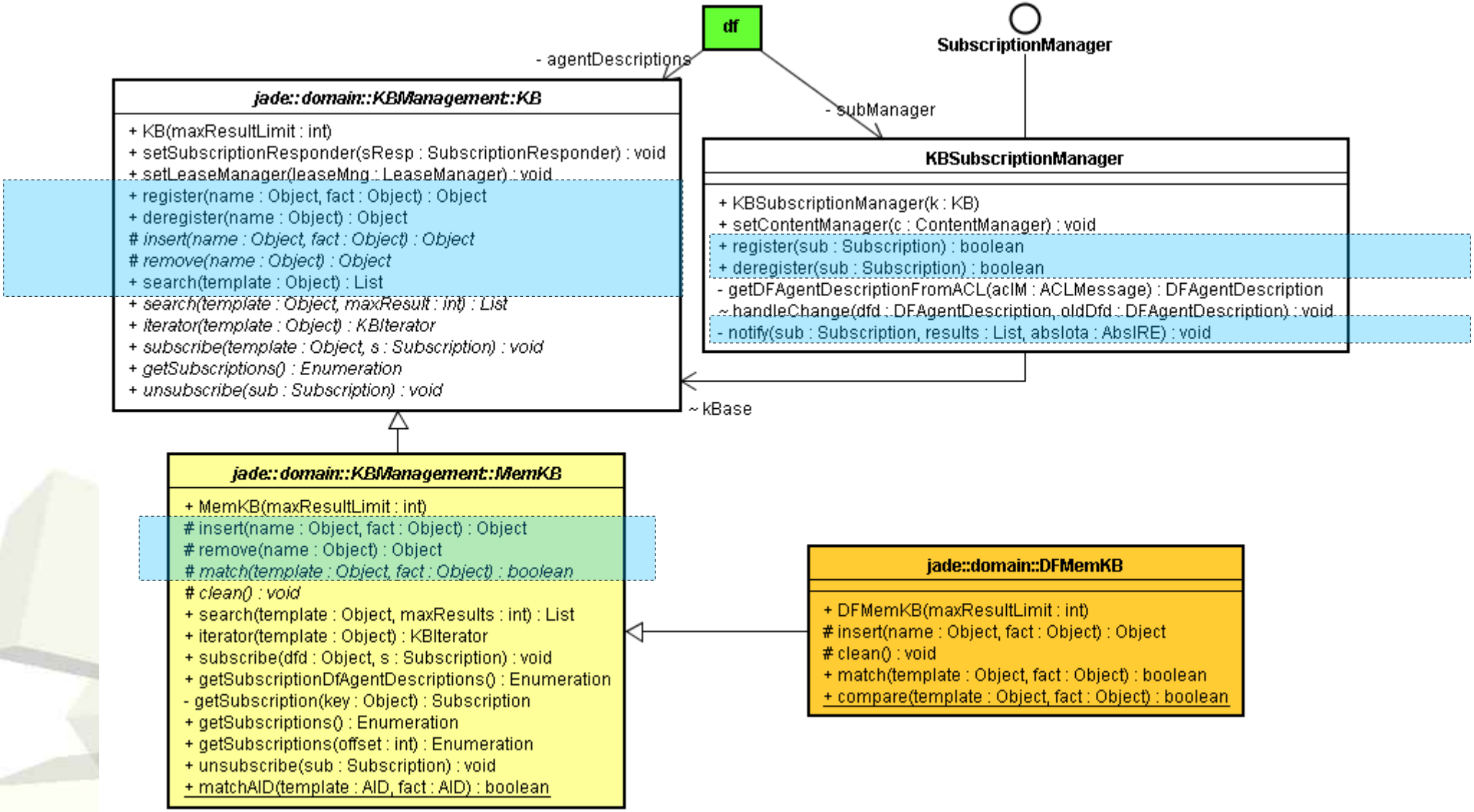
Subscription

- When a notification has to be sent to a subscribed agent the notification message should not be directly sent to the subscribed agent, but should be passed to the **`Subscription`** object representing the subscription of that agent by means of its **`notify()`** method.

- This method should be call instead of directly using the **`send()`** method of the **`Agent`** class, as it automatically handles sequencing and protocol fields appropriately.



Example: DF Agent



Communicative Act Categotegories

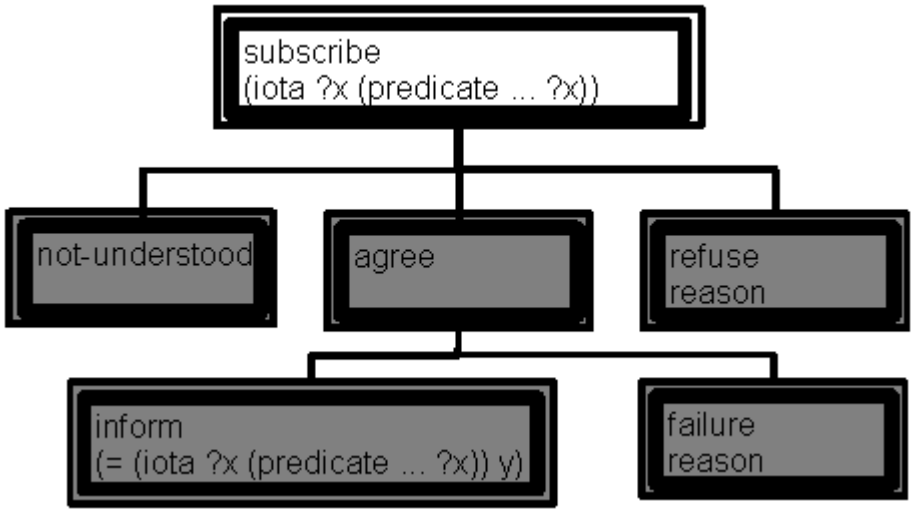
Communicative act	Information passing	Requesting information	Negotiation	Action performing	Error handling
accept-proposal			✓		
agree				✓	
cancel				✓	
cfp			✓		
confirm	✓				
disconfirm	✓				
failure					✓
inform	✓				
inform-if (macro act)	✓				
inform-ref (macro act)	✓				
not-understood					✓
propose			✓		
query-if		✓			
query-ref		✓			
refuse				✓	
reject-proposal			✓		
request				✓	
request-when				✓	
request-whenever				✓	
subscribe		✓			

Protocol Notation

- Rectangle with double edges – communication *act*
- White rectlange – action performed by *initiator*
- Gray rectangle – action performed by *others participants* of a protocol

Subscribe Protocol Syntax

Requesting general information fulfilling given predicate



`(iota x (P x))`

the *x* such that *P* [is true] of *x*

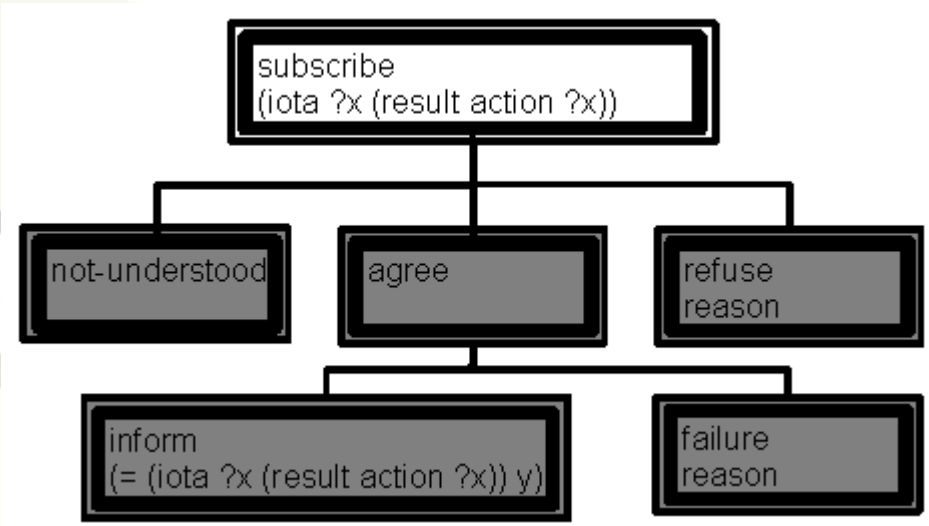
`(= x y)`

binary predicate of equality

`?x`

notion of variable

Requesting result of given action



`action`
`= "action" actor`
`action-name`
`actor`

AID of agent requested to act

`result action y`

y is result computational action *a*

Example: Subscribing at DF

subscribe
content

```
((iota ?x (result (action ( df )
  (search (df-agent-description
    :name ( agent-identifier :name ta@beethoven:1099/JADE
    :addresses (sequence http://beethoven:7778/acc )
    :X-JADE-agent-classname MyAgent )
    :services (set (service-description
      :name JADE-book-trading
      :type book-selling)))
  (search-constraints :max-results -1))) ?x)))
```

inform
content

```
((= (iota ?x (result (action ( df )
  (search (df-agent-description
    :name ( agent-identifier :name ta@beethoven:1099/JADE
    :addresses (sequence http://beethoven:7778/acc )
    :X-JADE-agent-classname MyAgent )
    :services (set (service-description
      :name JADE-book-trading
      :type book-selling)))
    (search-constraints :max-results -1))) ?x)))
  (sequence (df-agent-description
    :name (agent-identifier
      :name ta@beethoven:1099/JADE
      :addresses (sequence http://beethoven:7778/acc))
    :services (set (service-description
      :name JADE-book-trading
      :type book-selling))))))
```

result of
search action

DF Agent Description frame

Frame	df-agent-description
Ontology	fipa-agent-management

Parameter	Description	Presence	Type	Reserved Values
name	The identifier of the agent.	Optional	agent-identifier [8]	
services	A list of services supported by this agent.	Optional	Set of service-description	
protocols	A list of interaction protocols supported by the agent.	Optional	Set of string	See [FIPA00025]
ontologies	A list of ontologies known by the agent.	Optional	Set of string	fipa-agent-management
languages	A list of content languages known by the agent.	Optional	Set of string	fipa-sl fipa-sl0 fipa-sl1 fipa-sl2
lease-time	The duration or time at which the lease for this registration will expire [9].	Optional	datetime [10]	

FIPA Agent Management Specification, http://www.fipa.org/specs/fipa00023/index.html

Service Description frame

Frame Ontology	service-description fipa-agent-management			
Parameter	**Description**	**Presence**	**Type**	**Reserved Values**
name	The name of the service.	Optional	string	
type	The type of the service.	Optional	string	[11] fipa-df fipa-ams
protocols	A list of interaction protocols supported by the service.	Optional	Set of string	
ontologies	A list of ontologies supported by the service.	Optional	Set of string	fipa-agent-management
languages	A list of content languages supported by the service.	Optional	Set of string	
ownership	The owner of the service	Optional	string	
properties	A list of properties that discriminate the service.	Optional	Set of property	

FIPA Agent Management Specification, http://www.fipa.org/specs/fipa00023/index.html

Subscribe Communicative Act

- Summary:
  The act of requesting a persistent intention to notify the sender of the value of a reference, and to notify again whenever the object identified by the reference changes.

- Message content:
  A definite descriptor

- Description:
  [...]
  A subscription set up by a subscribe act is terminated by a cancel act.

Subscriptions condition

- Example
  Agent i *wishes to be updated on the exchange rate of Francs to Dollars, and makes a subscription agreement with* j *(an exchange rate server)*

```
(subscribe
    :sender i
    :receiver j
    :content (iota ?x (xch-rate FFr USD ?x))
)
```

- General
  - T. R. Gruber. *A translation approach to portable ontologies*. Knowledge Acquisition, 5(2):199-220, 1993. http://www-ksl.stanford.edu/kst/what-is-an-ontology.html

- N. F. Noy, D. L. McGuinness (2001), *Tutorial: Ontology Development 101* http://protege.stanford.edu/publications/ontology_development/ontology101.html

Why create an ontology?

- An ontology provides a common vocabulary for researchers who need to share information in the domain. Some of the reasons to create an ontology are:
  - To share common understanding of the structure of information among people or software agents
  - To enable reuse of domain knowledge
  - To make domain assumptions explicit
  - To separate domain knowledge from operational knowledge
  - To analyze domain knowledge
  -
-

Ontology in Protégé

- Ontology:

  *"Specification of conceptualization"**

  - i.e. specification of
    - what **exists**
    - what are **relations** among parts

- In context of Protégé**:
  - Description of:
    - **classes** (*concepts*) in a domain of discourse
    - **slots** (*properties*) of each class describing various features and attributes of the class
  - An ontology together with a set of individual **instances** of classes constitutes a **knowledge base**.

* T. R. Gruber. *A translation approach to portable ontologies*. Knowledge Acquisition, 5(2):199-220, 1993.

** N. F. Noy, D. L. McGuinness (2001), *Tutorial: Ontology Development 101*

- *Getting Started with Protégé-Frames*
  http://protege.stanford.edu/doc/tutorial/get_started/get-started.pdf

Importing JADE base ontology

- Open Project->Manage Included Projects... menu item
- Click Add Project button (with + icon)
- Choose beanynizer_default.pprj project and click OK.

Creating own ontology

- Choose Create New Project button or open File->New Project... menu item.
- Select a project type: Protégé Files (.pont and .pins) and click OK.
- Save project (enter only desired project path name) and click OK.

Switching Jadex Beanynizer plugin

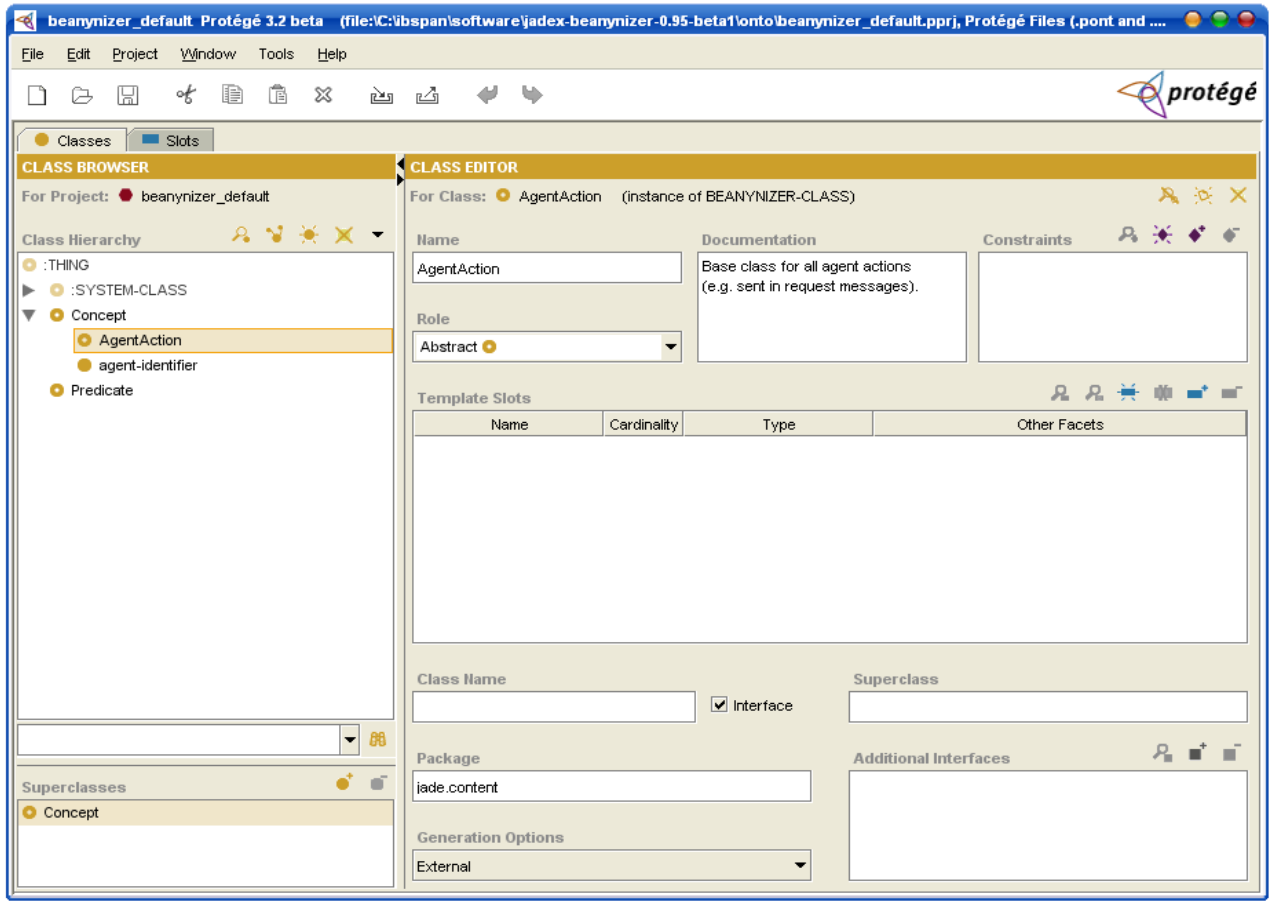
- Open Project->Configure... menu item.
- Turn on Beaninizer option in Widgets tab and click OK.
- Switch into Jadex Beanynizer tab.

Bean base ontology in Protégé

Base ontology is defined in

`beanynizer_default.pprj`

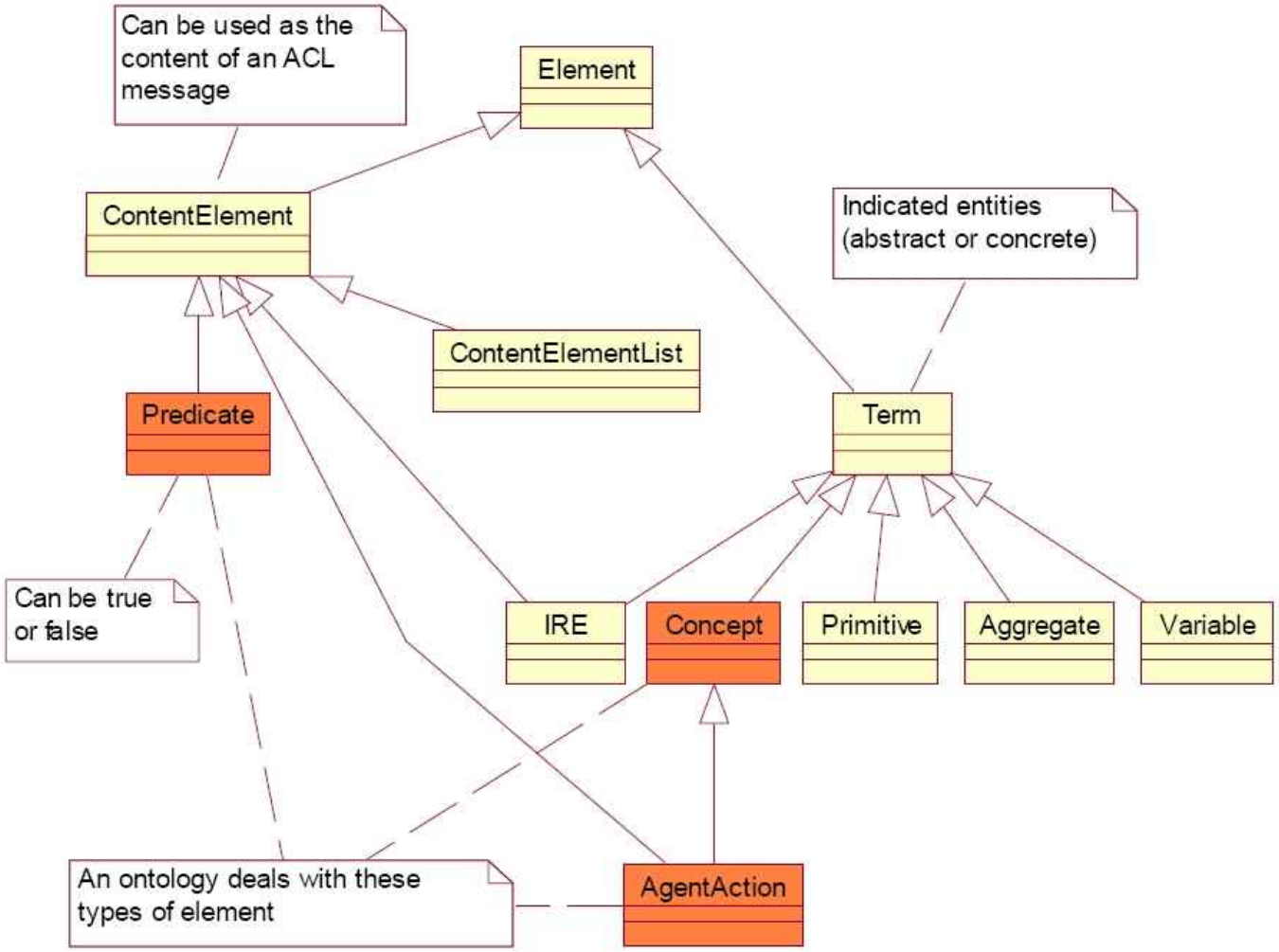
Protégé project file from Jader Beanynizer plugin.



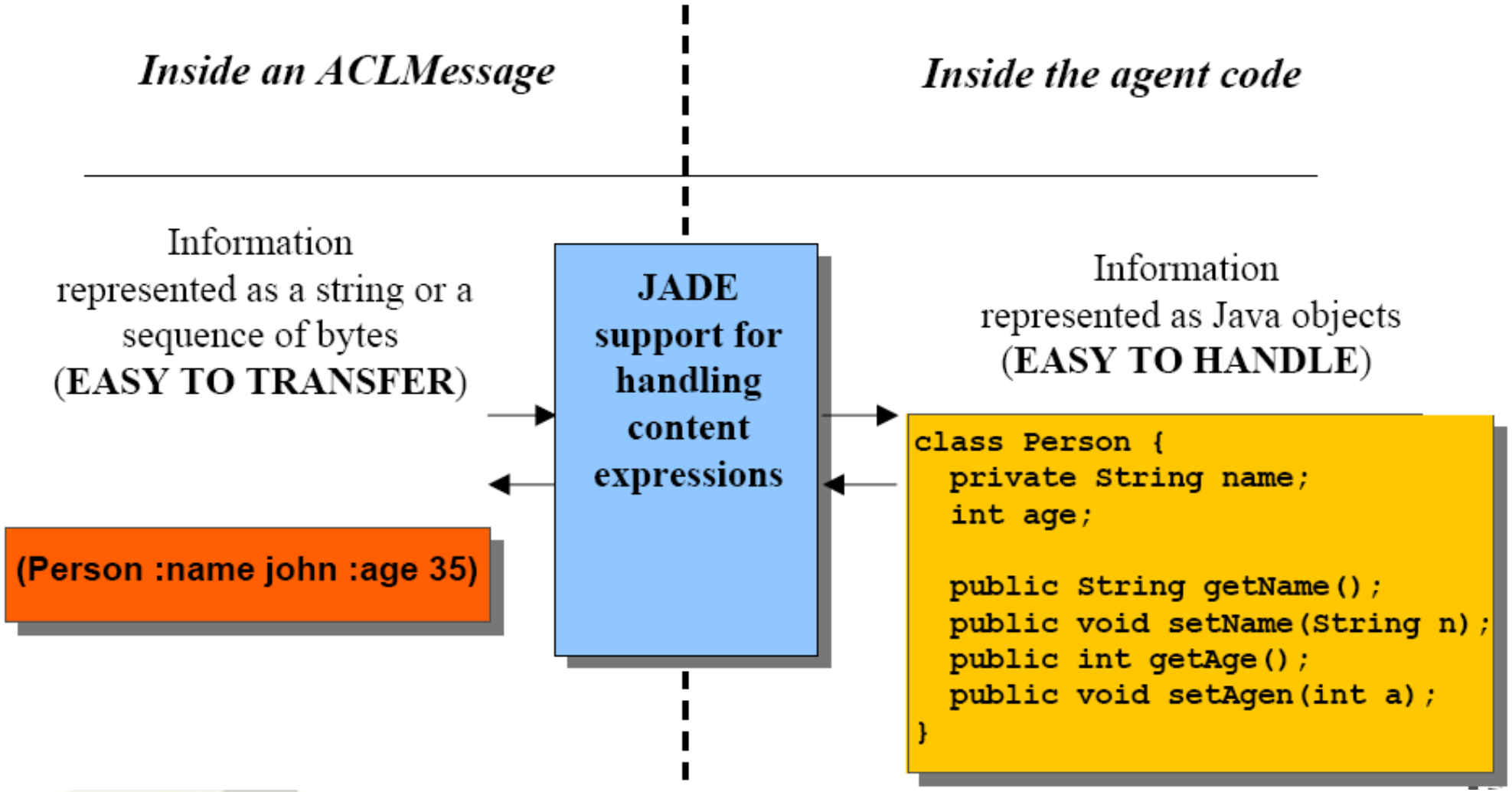
The *Classes Tab* is an ontology editor which you can use to define *classes* and class *hierarchy*, *slots* and *slot-value restrictions*, *relationships* between classes and properties of these relationships.

JADE Content Reference Model

Base ontology is realized by the following concepts in JADE:



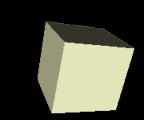
Handling content expressions



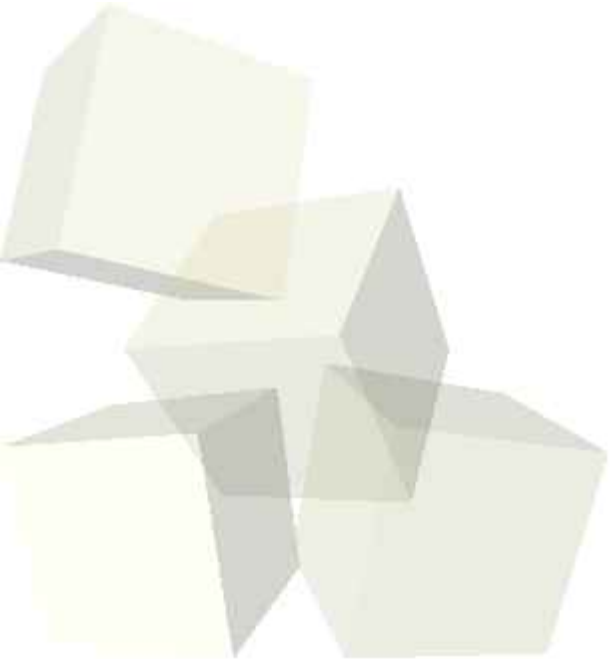
Semantic Add-on vs. Jadex

- **Semantic Add-on**
  - allows for communications on a semantic level, which means that the agent can understand each other
  - +JADE = a step towards a real **communication-oriented middleware**

- **Jadex**
  - addresses the internal reasoning process of agents
  - an implementation of a **hybrid** (reactive and deliberative) agent architecture for representing mental states in JADE agents following the BDI model.



Jadex as realization of BDI Architecture



- Deciding on what goals to achieve and how to achieve them
  - **Beliefs**: the information an agent has about its surroundings
  - **Desires**: the things that an agent would like to see achieved
  - **Intentions**: the desires that an agent is working on; also involves a deeper personal commitment
- A **BDI architecture** addresses how beliefs, desires and intentions are represented, updated, and processed

Jadex, BDI extension to JADE

- Jadex open source project
  http://vsis-www.informatik.uni-hamburg.de/projects/jadex/

- *Background*: two different type agent platform exists:
  - **FIPA-compliant platforms** mainly addressing openness and middleware issues (with to respect FIPA standards)
  - **Reasoning-centered platforms** focusing on the behaviour model of a single agent, e.g. trying to achieve rationality and goal-directedness.

- Jadex fill the gap between middleware and reasoning-centered systems.

Launching sample

- Unpack the distribution
- Set the *classpath*:
  - **Java**: `.;C:\Java\jdk1.5.0_04\lib;`
  - **JADE**: `C:\jade\lib\jade.jar; C:\jade\lib\jadeTools.jar; C:\jade\lib\Base64.jar; C:\jade\lib\http.jar; C:\jade\lib\iiop.jar;`
  - **Jadex**: `C:\jadex-0.94\lib\jadex_rt.jar; C:\jadex-0.94\lib\jibx-run.jar; C:\jadex-0.94\lib\xpp3.jar; C:\jadex-0.94\lib\jadex_standalone.jar; C:\jadex-0.94\lib\jadex_tools.jar; C:\jadex-0.94\lib\GraphLayout.jar; C:\jadex-0.94\lib\jhall.jar; C:\jadex-0.94\lib\jadex_examples.jar`
- Start the *platform*:

  ```
  java jadex.adapter.standalone.Platform
  ```

- The **Jadex Control Center** will appear

Example: World Cleaner Scenario

The cleanerworld is based on the idea that an autonomous cleaning robot has the task to clean up dirt in some environment.

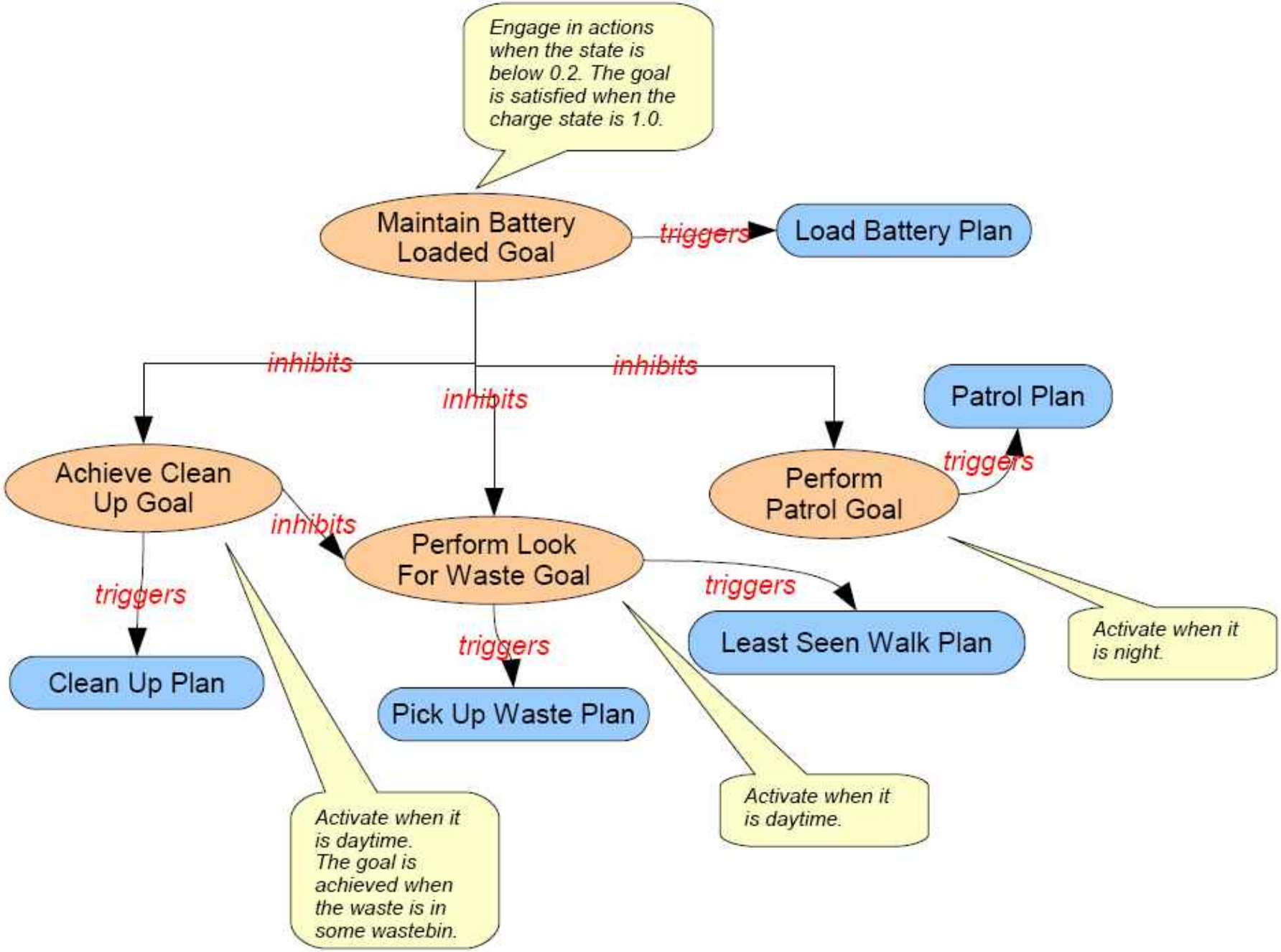
In our scenario of the cleaner world the main system objectives are to keep clean a building at day, e.g. a museum, and to guard the building at night.

To be more concise we think of a group of cleaning robots that are located in the building and try to accomplish the overall system goals by pursuing their own goals in coordination with other individuals.

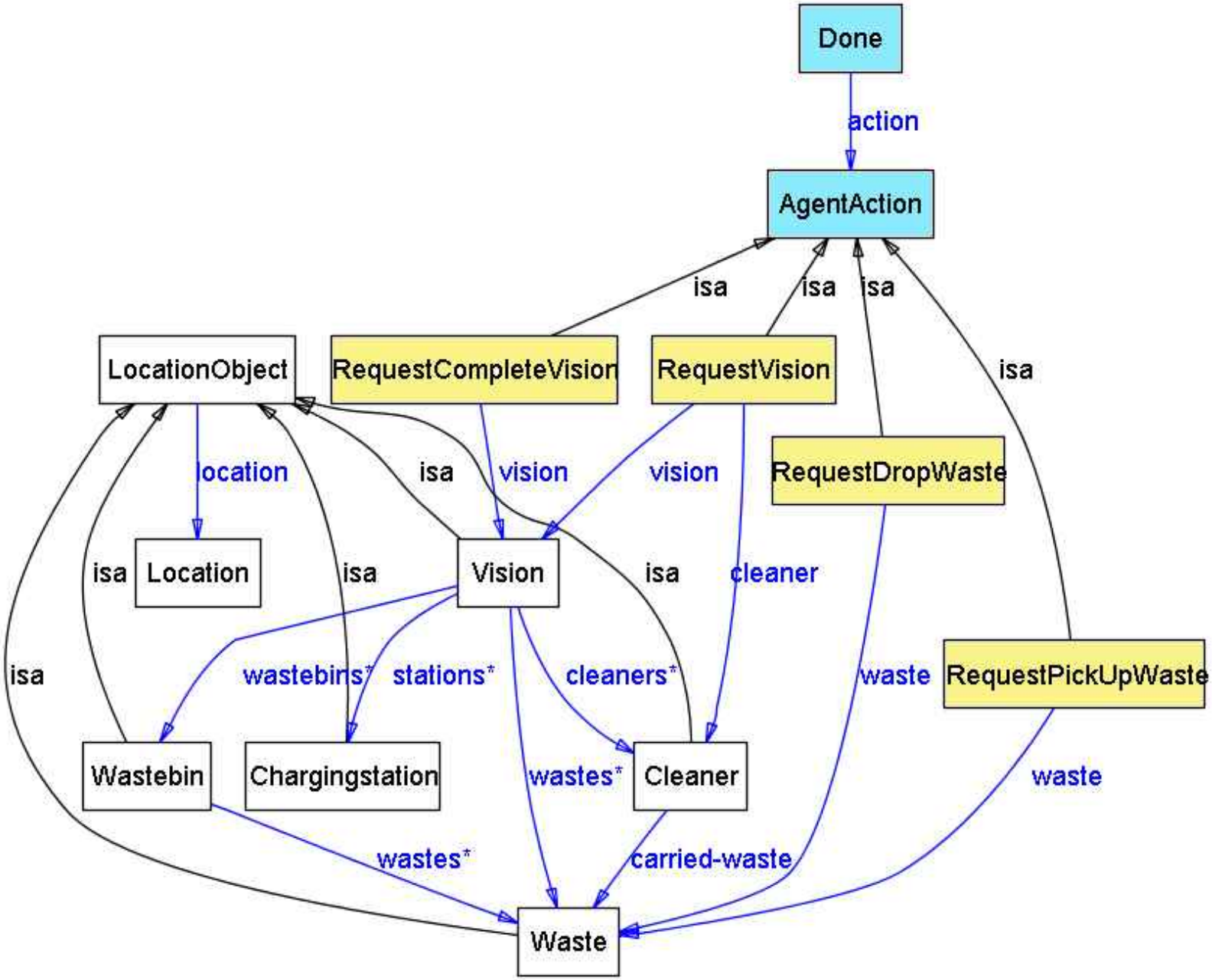
Key goals of World Cleaner

- Therefore, three key goals for an individual cleaning robot were identified.
  - First, it should **clean its environment at day** by removing dirt whenever possible. The cleaning robot therefore has to pick-up any garbage and carry it to a near waste bin.
  - Secondly, it has to **guard the building at night** by performing patrols that should be based on varying routes. Any suspicious occurrences that it recognises during its patrols should be reported to some superordinated authority.
  - Thirdly, it should keep operational by **monitoring its internal states** such as the charge state of its battery or recognised malfunctions. Whenever its battery state is low it has to move to the charging station.

Cleaner: Top Level Goals & Plans

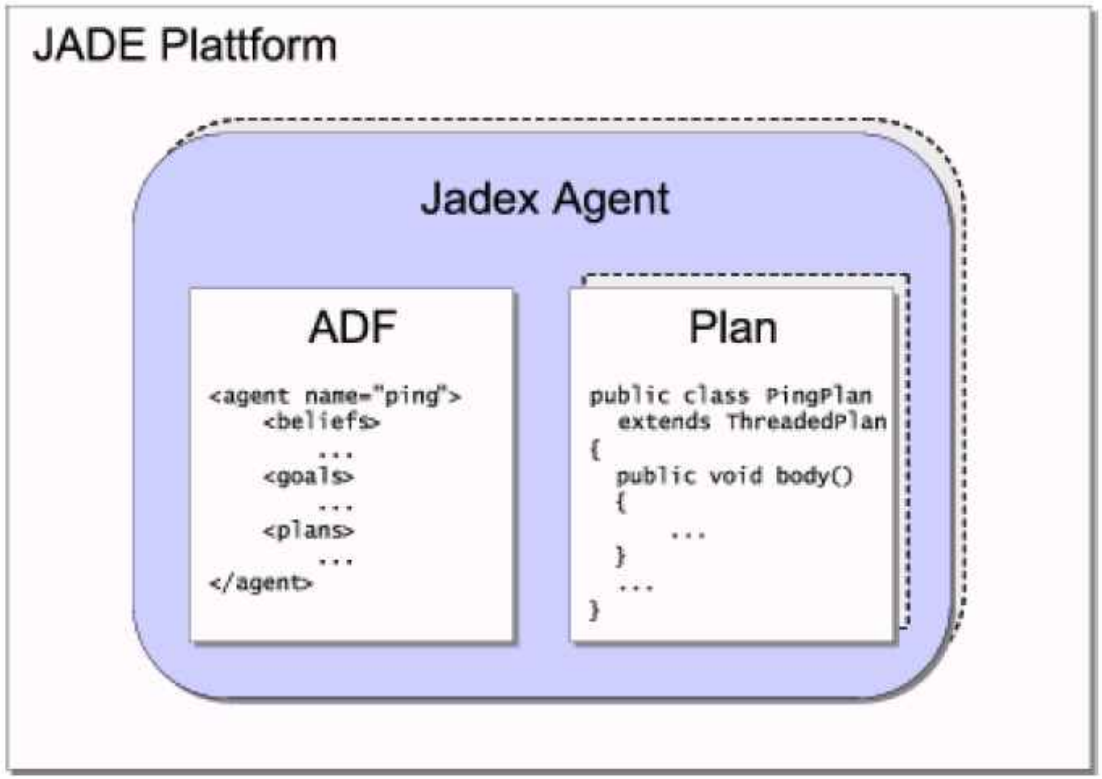


CleanerWorld Ontology



Jadex Agent

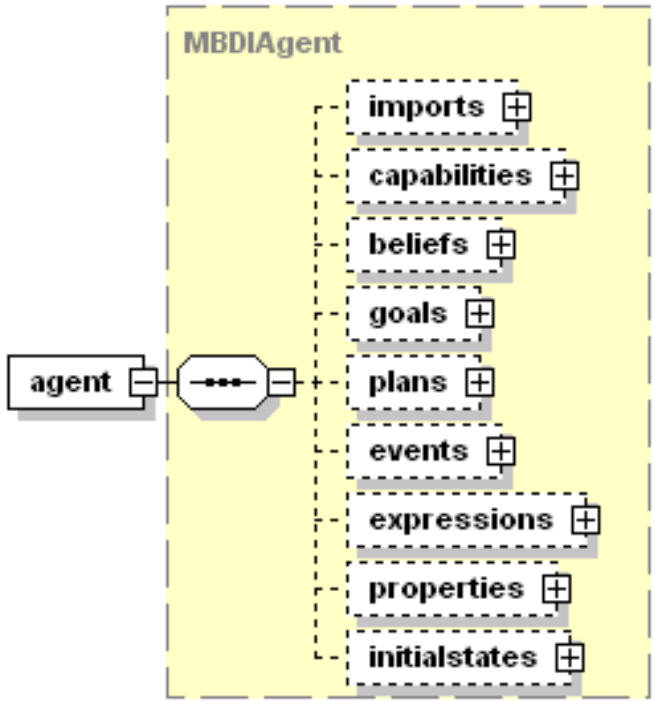
- A Jadex agent has two basic parts:
  - An **Agent Definition File** (ADF) written in XML
  - A set of Java classes, which specialize Jadex built-in classes, to specify how ***plans*** (*intentions*) are constructed out of ***beliefs*** and ***goals*** (*desires*)



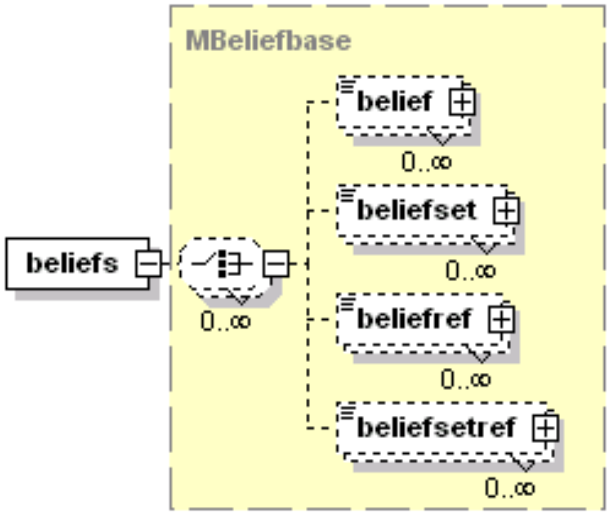
Agent Definition File

- XML file describing agent in respect to Jadex XML
  http://jadex.sourceforge.net/jadex-0.94.xsd
- Schema described in Jadex documentation

- Tool for editting: *XMLBuddy* plugin for Eclipse
  http://www.xmlbuddy.com/



Beliefs in the ADF



- **Belief**
  A single-valued piece of agent knowlegde.
- **BeliefSet**
  A multi-valued piece of agent knowlegde.
- **fact**
  An expression that evaluated to a default value.

```
<beliefs>
  <belief name="environment" class="IEnvironment">
    <!-- local environment (comment out for remote) -->
    <!--<fact>Environment.getInstance()</fact>-->
  </belief>
  ...
  <!-- The points used for patrolling at night. -->
  <beliefset name="patrolpoints" class="Location">
    <fact>new Location(0.1, 0.1)</fact>
    <fact>new Location(0.1, 0.9)</fact>
    <fact>new Location(0.3, 0.9)</fact>
    <fact>new Location(0.3, 0.1)</fact>
    <fact>new Location(0.5, 0.1)</fact>
    <fact>new Location(0.5, 0.9)</fact>
    <fact>new Location(0.7, 0.9)</fact>
    <fact>new Location(0.7, 0.1)</fact>
    <fact>new Location(0.9, 0.1)</fact>
    <fact>new Location(0.9, 0.9)</fact>
  </beliefset>
</beliefs>
```

Access to beliefs within Plan body

```
/**
 *  Patrol along the patrol points.
 */
public class PatrolPlan extends Plan {
  //-------- constructors --------
  /**
   *  Create a new plan.
   */
  public PatrolPlan() {
   getLogger().info("Created: "+this);
  }

  //-------- methods --------
  /**
   *  The plan body.
   */
  public void body() {
    Location[] loci =
     (Location[])getBeliefbase().getBeliefSet("patrolpoints").getFacts();

    for(int i=0; i<loci.length; i++) {
      IGoal moveto = createGoal("achievemoveto");
      moveto.getParameter("location").setValue(loci[i]);
      dispatchSubgoalAndWait(moveto);
    }
  }
}
```

Object Query Language-like queries

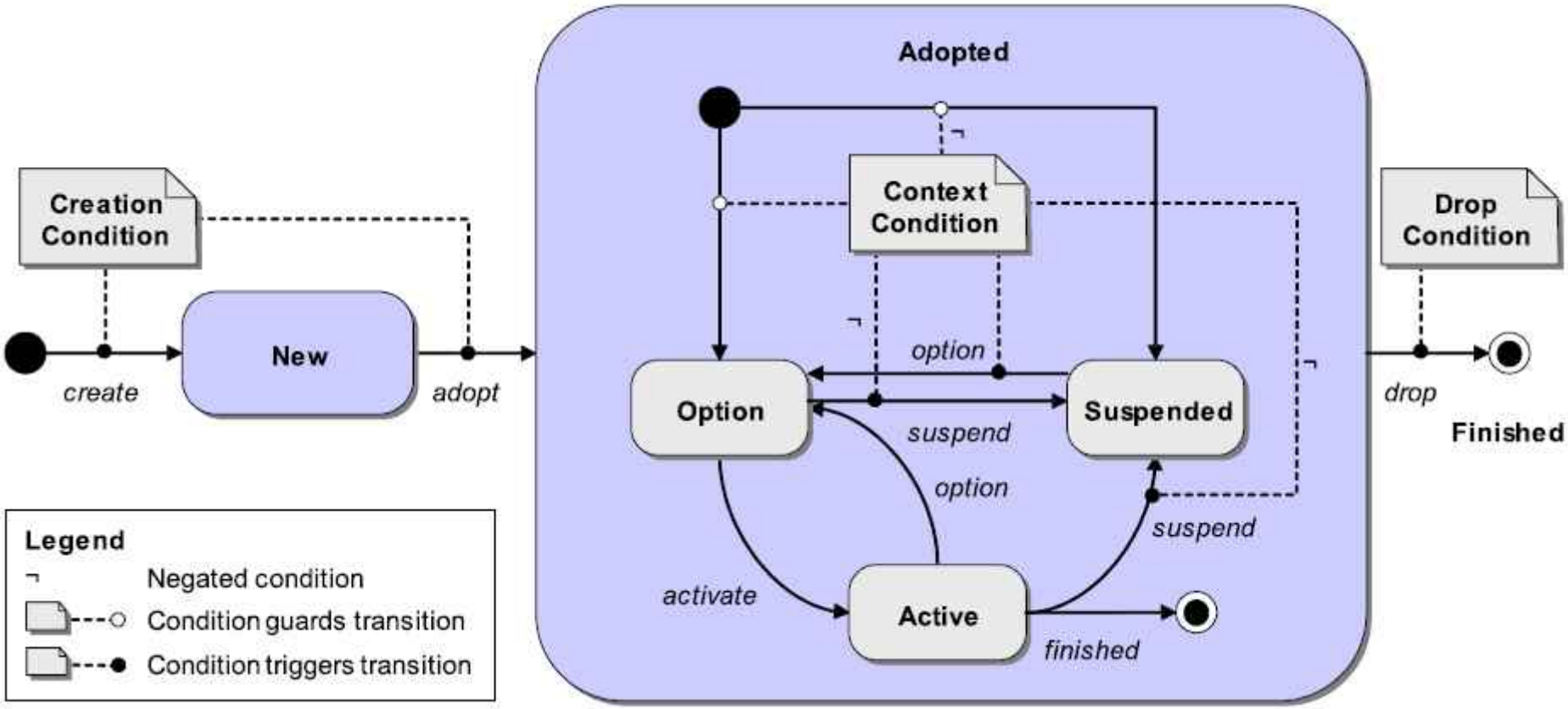
OQL syntax in EBNF:

```
select expression ::= "SELECT"("ALL" | "ANY" | "IOTA")?
(
  (expression "FROM"("$" identifier "IN" expression) (","$" identifier "IN" expression)* )
 | ("$" identifier "FROM"expression)
 )
 ("WHERE" expression)?
 ("ORDER""BY"expression ("ASC" | "DESC")? )?
```

Query example inside of Query Goal:

```
<!-- Try to find a not full waste bin that
     is as near as possible to the agent. -->
<querygoal name="querywastebin" exclude="never">
  <parameter name="result" class="Wastebin" direction="out">
    <value evaluationmode="dynamic">
       select one Wastebin $wastebin
       from $beliefbase.wastebins
       where !$wastebin.isFull()
       order by
         $beliefbase.my_location.getDistance($wastebin.getLocation())
    </value>
  </parameter>
</querygoal>
```

Goal lifecycle



Situation of conflicting goals

- Goal-oriented agent is capable of purisying **multiple goals simultaneously**
- Some goals could be **conflicted**
  - Example:
    - Agent cannot both *Maintain Battery Loaded* and Perform *Look For Waster*, or *Perform Patrol*
    - Agent cannot look for a new waste, if the old one has not been cleaned up
- Some goals require limitation in **number of activated instances**:
  - Example
    - For improved *performance*, the cleaner should alsway clean up the nearest piece of waste first

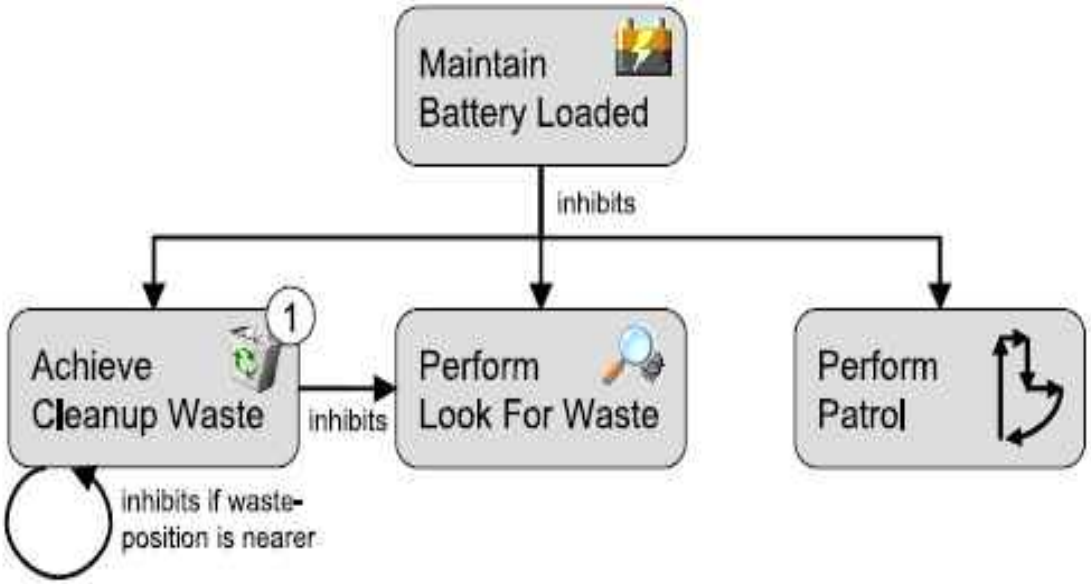
Goals deliberation strategy (1)

- Goals deliberation allows for avoiding activatation of conflicted goals
- Jadex uses ***Easy Deliberation*** strategy for this purpose

- Driving factors:
  - **Cardinalities** for goal instances
    Only *x* instances of a certain type of goal is allowed to be active simultanously
    - ➔ Example:
      - *Achieve Cleanup Waste* goal with cardininality of 1
  - **Inhibition links**:
    Goals which has been activated should suspend goals inhibited by it
    - ➔ Example:
      - If an agent *Maintains Battery Loaded* then it inbits realization of all other goals

Goals deliberation strategy (2)

- Graph consitisting of inhibiting arc should be acyclic to avoid cycles in deliberations



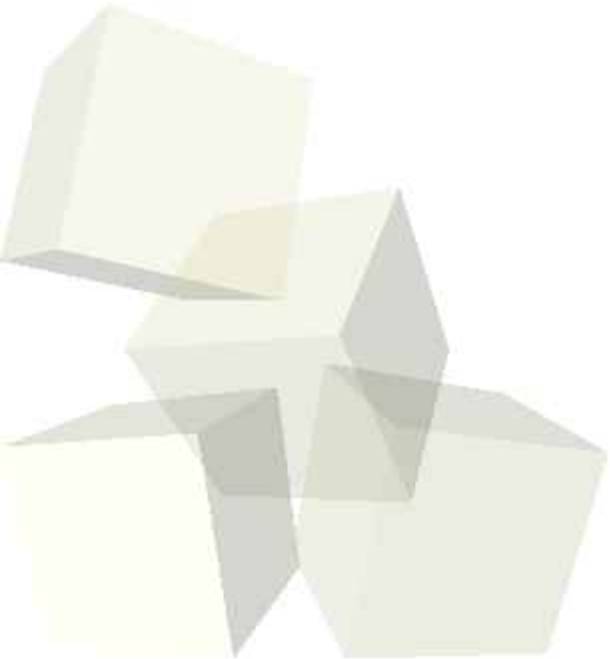

```
<!-- Observe the battery state. -->
<maintaingoal name="maintainbatteryloaded">
  <deliberation>
    <inhibits ref="performlookforwaste" inhibit="when_in_process"/>
    <inhibits ref="achievecleanup" inhibit="when_in_process"/>
    <inhibits ref="performpatrol" inhibit="when_in_process"/>
  </deliberation>
  <!-- Engage in actions when the state is below 0.2. -->
  <maintaincondition>
    $beliefbase.my_chargestate > 0.2
  </maintaincondition>
  <!-- The goal is satisfied when the charge state is 1.0. -->
  <targetcondition>
    $beliefbase.my_chargestate >= 1.0
   </targetcondition>
</maintaingoal>
```

When often should be delibarate ?

- Only **on demand**
  - 1: *Deliberate a new option*
    Check which inhibited goals should be suspended
  - 2: *Deliberate a deactive goals*
    Check which inhibited goals should be reactivated.

Goal types in Jadex

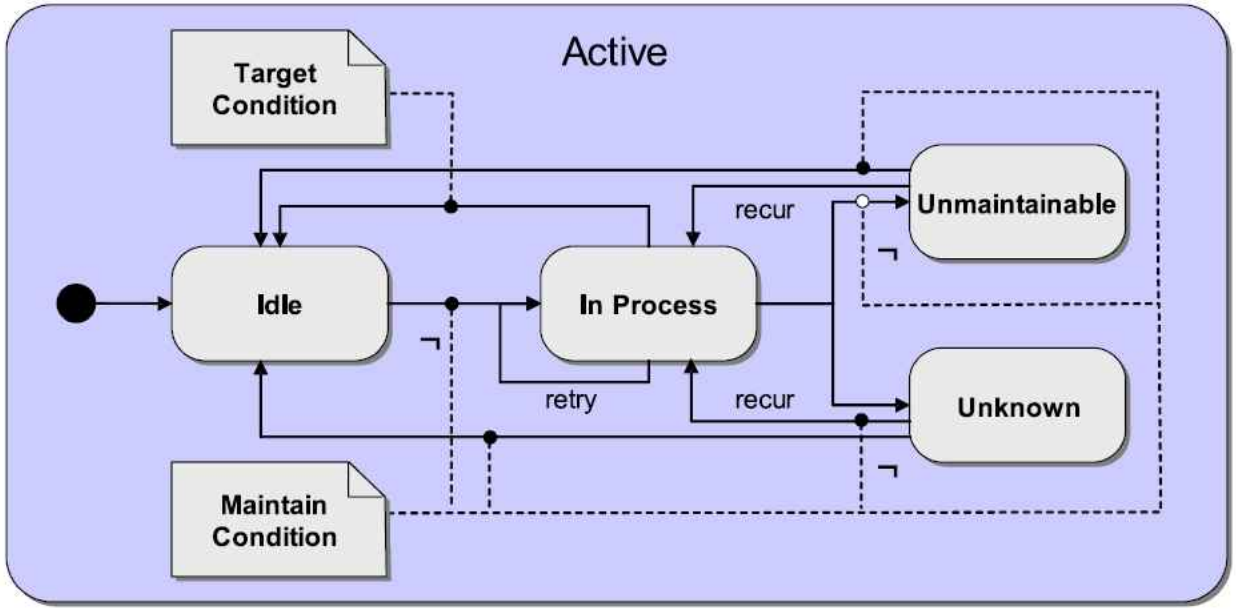
- Perform
- Achieve
- Query
- Maintain

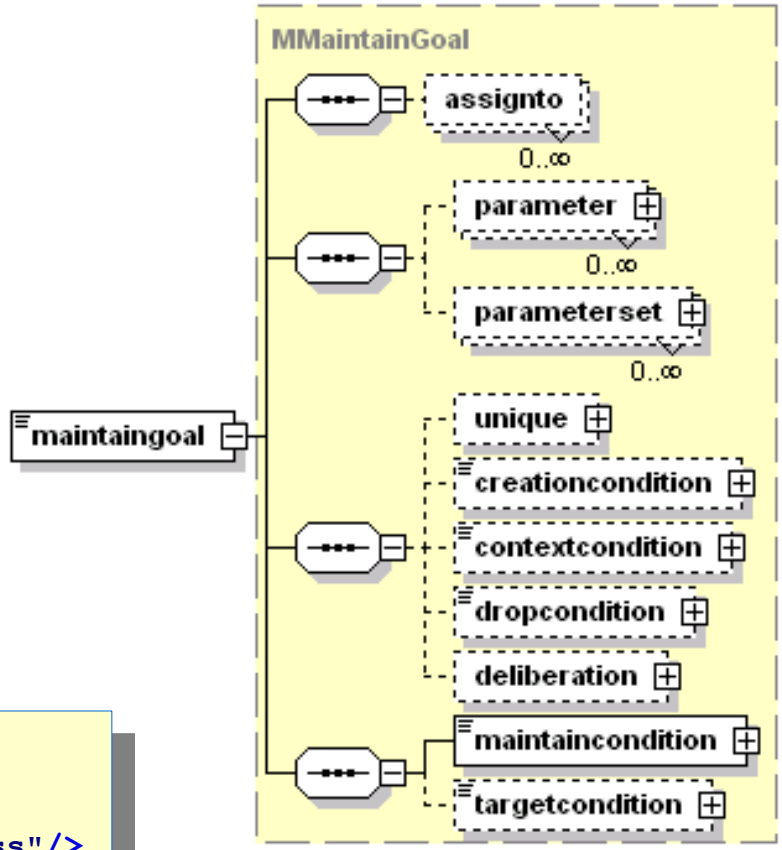


Goal elements: conditions

- **CreationCondition**
  A condition that creates a new goal of the given type when triggered.
- **DropCondition**
  If the dropcondition triggers the goal instance is dropped.
- **Deliberation**
  The goal deliberation setting for the *easy deliberation* strategy.
- **ContextCondition**
  The context condition is checked during the whole execution time of a goal. If it becomes invalid the goal will become suspended and is not actively pursued until reactivation.
- **MaintainCondition**
  The mandatory maintain condition represents a world state that should be monitored and re-established whenever it gets violated.
- **TargetCondition**
  A specalisation of the maintain condition taht should be re-established when the maintain condition is violated.
- **FailureCondition**
  Can be used to explicitly state when a goal cannot be pursued any longer and is failed.

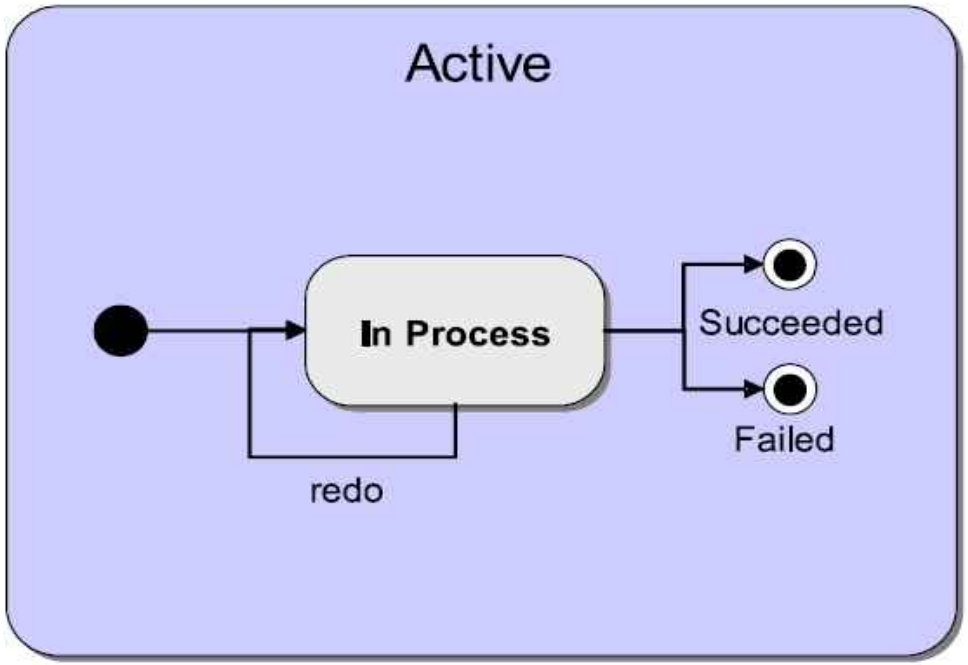
Maintain goal states

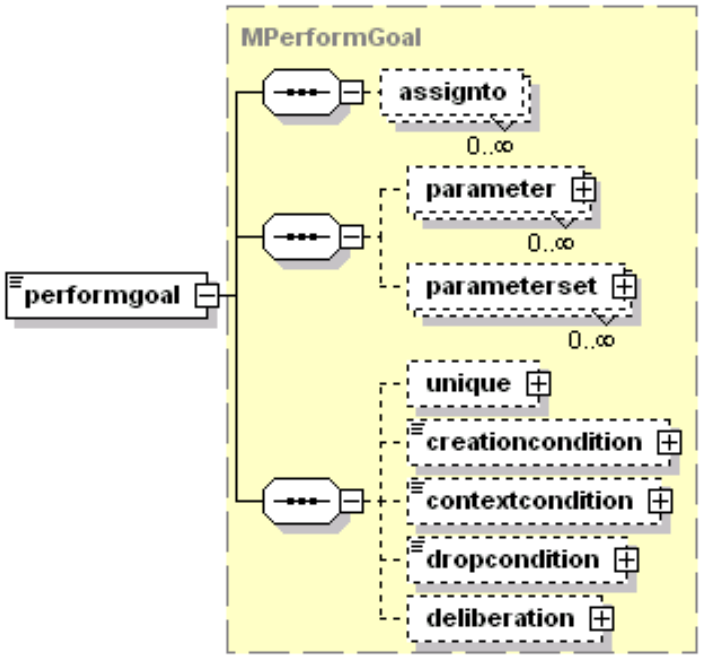





```
<!-- Observe the battery state. -->
<maintaingoal name="maintainbatteryloaded">
  <deliberation>
    <inhibits ref="performlookforwaste" inhibit="when_in_process"/>
    <inhibits ref="achievecleanup" inhibit="when_in_process"/>
    <inhibits ref="performpatrol" inhibit="when_in_process"/>
  </deliberation>
  <!-- Engage in actions when the state is below 0.2. -->
  <maintaincondition>
    $beliefbase.my_chargestate > 0.2
  </maintaincondition>
  <!-- The goal is satisfied when the charge state is 1.0. -->
  <targetcondition>
    $beliefbase.my_chargestate >= 1.0
   </targetcondition>
</maintaingoal>
```

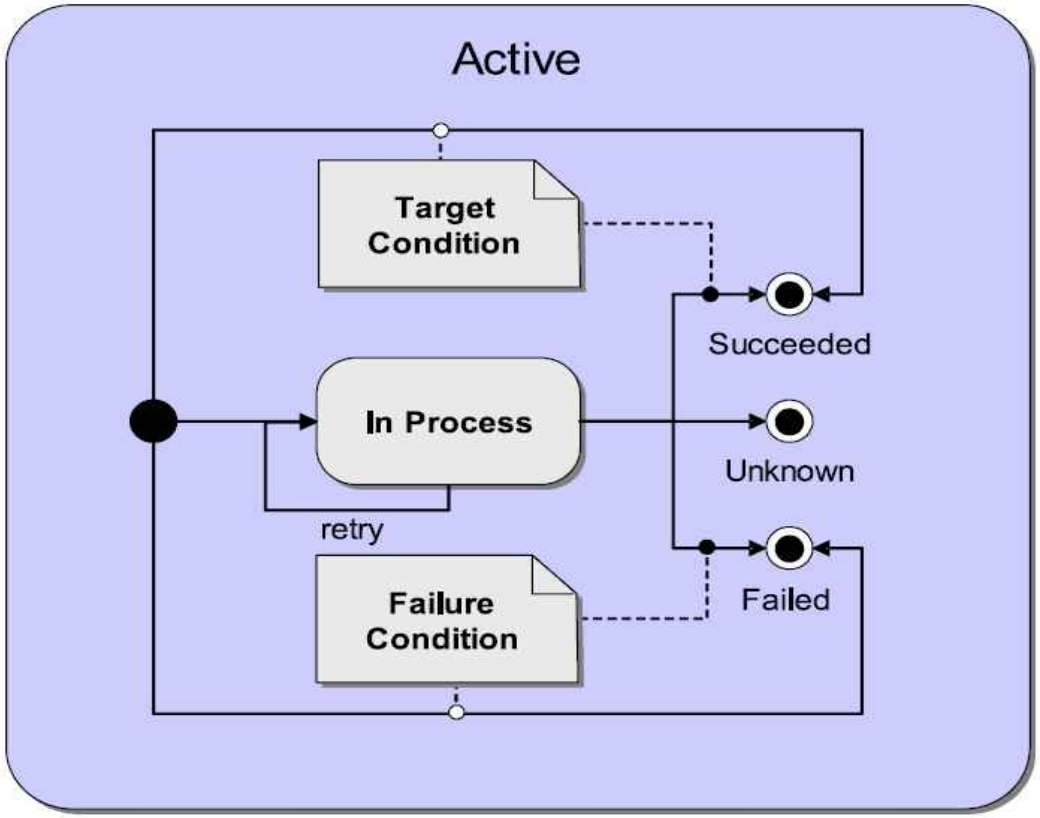
Perform goal

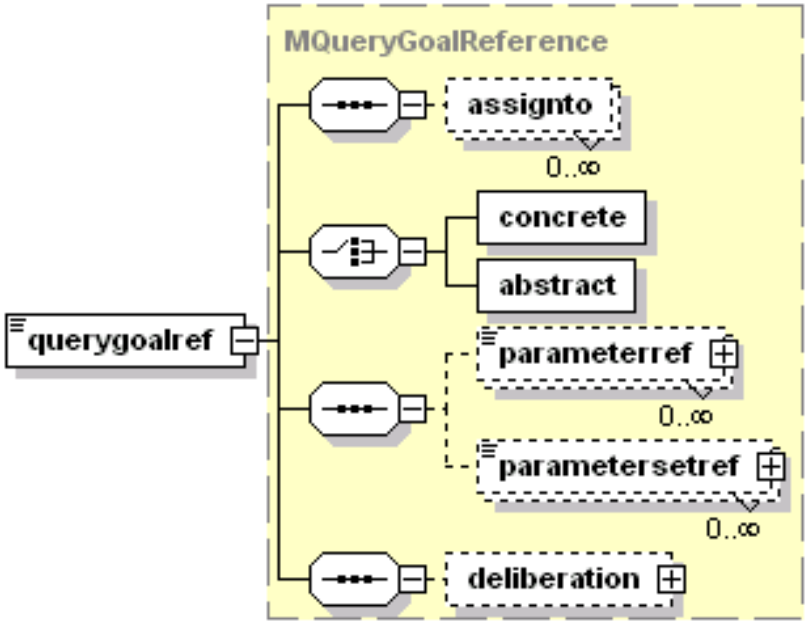





```
<!-- Look out for waste when nothing better to do, what means that
     the agent is not cleaning, not loading and it is daytime. -->
<performgoal name="performlookforwaste" retry="true" exclude="never">
  <contextcondition>
    $beliefbase.daytime
  </contextcondition>
</performgoal>
```

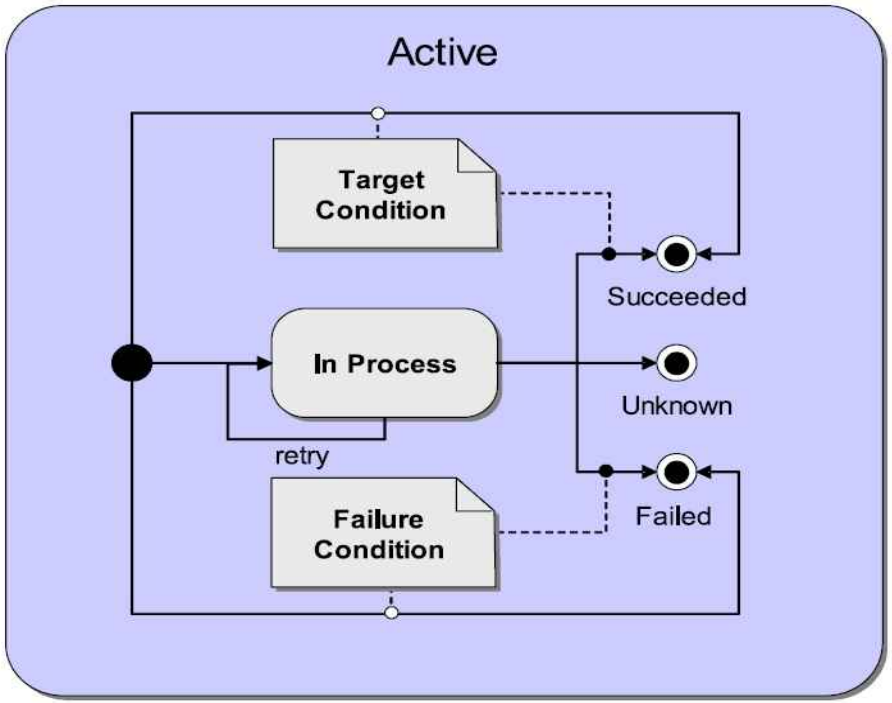
Query goal

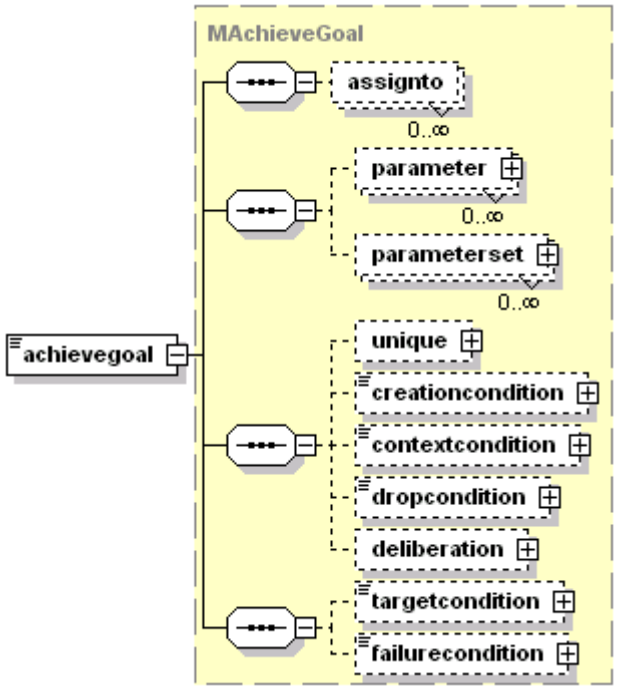





```
<!-- Try to find a not full waste bin that
     is as near as possible to the agent. -->
<querygoal name="querywastebin" exclude="never">
  <parameter name="result" class="Wastebin" direction="out">
    <value evaluationmode="dynamic">
      select one Wastebin $wastebin
      from $beliefbase.wastebins
      where !$wastebin.isFull()
      order by
        $beliefbase.my_location.getDistance($wastebin.getLocation())
    </value>
  </parameter>
</querygoal>
```

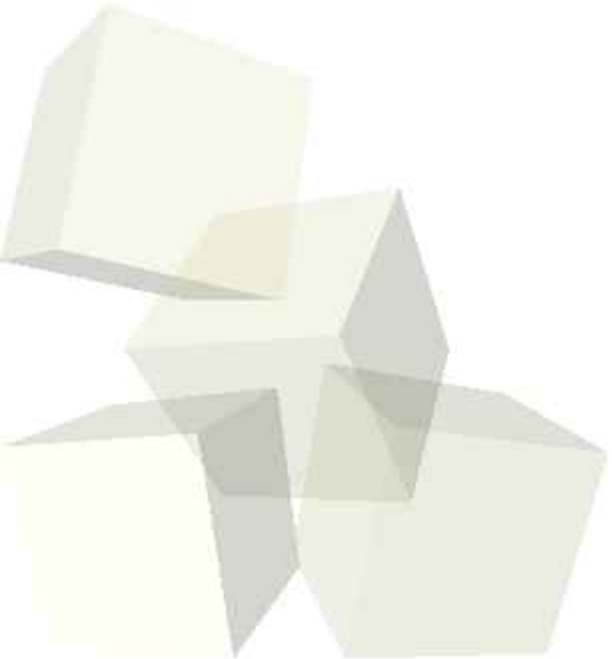
Achieve goal






```
<!-- Drop a piece of waste into a wastebin. -->
<achievegoal name="achievedropwaste" retry="true" exclude="never">
  <parameter name="wastebin" class="Wastebin"/>
  <!-- The goal has failed when the aimed wastebin is full. -->
  <failurecondition>
    (select one Wastebin $wastebin
    from $beliefbase.wastebins
    where $goal.wastebin.getId().equals($wastebin.getId())).isFull()
  </failurecondition>
</achievegoal>
```

```
<!-- Try to move to the specified location. -->
<achievegoal name="achievemoveto">
  <parameter name="location" class="Location"/>
  <!-- The goal has been reached when the agent's location is
       near the target position as specified in the parameter. -->
  <targetcondition>
    $beliefbase.my_location.isNear($goal.location)
  </targetcondition>
</achievegoal>
```



Jadex: Getting Started

Goal structure in ADF