JADE Notes

ref: Developing Multi-agent Systems with JADE, misc. JADE .PDFs

FIPA: Foundation for Intelligent, Physical Agents – defines the external behavior of agent system components, leaving internal details to developers of individual platforms. JADE is one such platform.

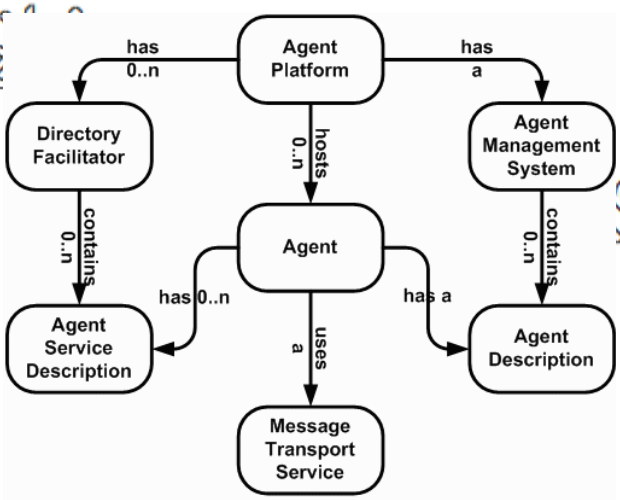


Figure 2.5 FIPA Agent Management Ontology

Agent: Autonomous software component that provides an interoperable interface to an arbitrary system and/or behaves like a human agent -- working for some clients in pursuit of its own agenda. Each agent has a unique agent identifier (AID) assigned by the AMS, and owns its own thread.

Agent Platform (AP): Physical infrastructure in which agents are deployed. Consists of physical devices (hosts), operating systems, agents and support software, and the components that follow. A single AP may be spread across multiple physical devices (hosts).

Agent Management System (AMS): Provides "white page" service to platform agents (where agents make themselves known and discover other agents). Is an agent. Responsible for managing the operation of an AP (e.g. agent creation/deletion and overseeing the migration of agents from/to the AP). Mandatory AP component.

Directory Facilitator (DF): Provides "yellow page" service to platform agents (where agents publish their and discover other agents' services). Is an agent. Maintains an accurate, complete and timely list of all agents and must provide the most current information about agents in its directory on a non-discriminatory basis to all authorized agents. An AP may support any number of DFs which may register with one another to form federations. Optional AP component.

Message Transport Service (MTS): Transports FIPA-ACL (agent communication language) messages between agents on any given AP and between agents on different APs.

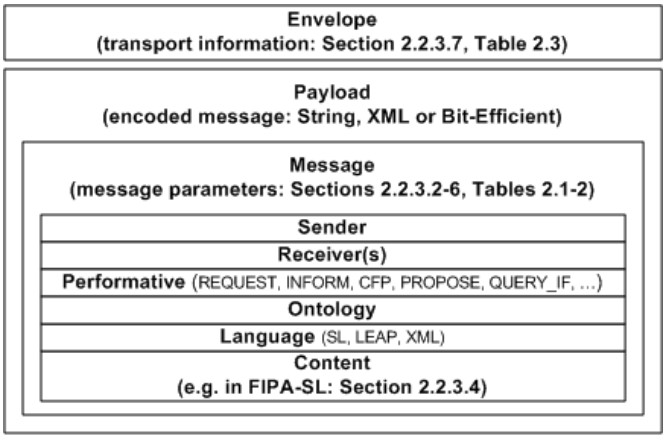


Figure 2.6 Typical FIPA-ACL Message Structure

JADE Notes

ref: Developing Multi-agent Systems with JADE, misc. JADE .PDFs

FIPA Agent Life-Cycle: from JADE programmersguide.pdf

3.2.1 Agent life cycle

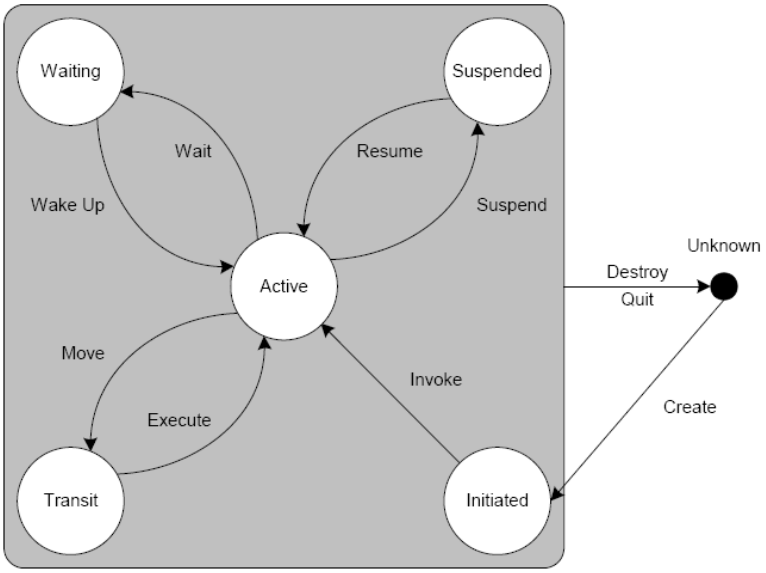


Figure 3 - Agent life-cycle as defined by FIPA.

1 **Initiated:** the Agent object is built, but hasn't registered itself yet with the AMS, has neither a name nor an address and cannot communicate with other agents.

2 **Active:** the Agent object is registered with the AMS, has a regular name and address and can access all the various (JADE) features.

3 **Suspended:** the Agent object is currently stopped. Its internal thread is suspended and no agent behavior is being executed.

4 **Waiting:** the Agent object is blocked, waiting for something. Its internal thread is sleeping on a Java monitor and will wake up when some condition is met (typically when a message arrives).

5 **Deleted:** the Agent is definitely dead. The internal thread has terminated its execution and the Agent is no more registered with the AMS.

6 **Transit:** a mobile agent enters this state while it is migrating to the new location. The system continues to buffer messages that will then be sent to its new location.

FIPA Agent Characteristics:

Autonomous: operates without direct intervention, and has control over its actions and internal state.
Social: co-operates with other agents or humans.
Reactive: perceives and responds to environmental changes.

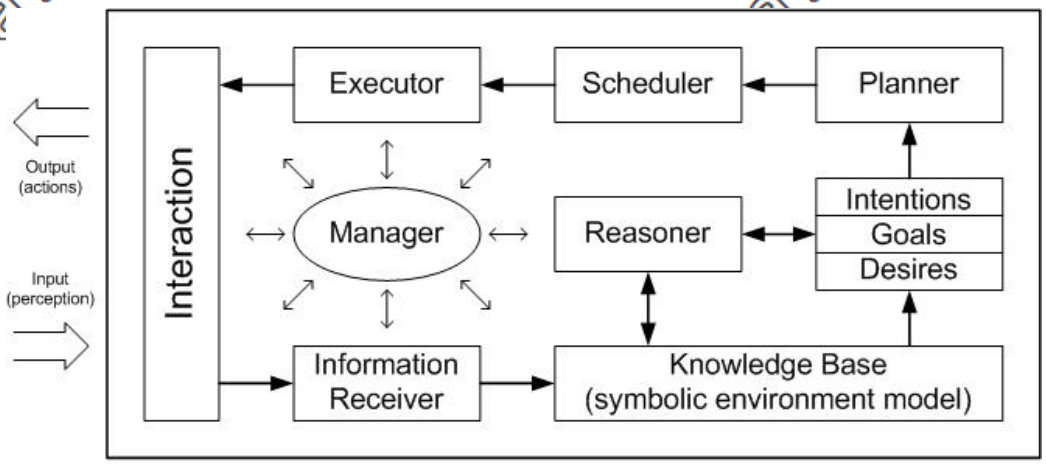
Proactive: can exhibit goal-directed behavior by taking the initiative.
Mobile: can have the ability to travel between network nodes.
Truthful: can (has the choice of) not deliberately communicating false information.
Benevolent: can always try to perform what's asked of it.
Rational: can (has the choice of) always trying to achieve its goals.
Learning: can have the ability of adapting itself to its environment and to its user's desires.

JADE Notes

ref: Developing Multi-agent Systems with JADE, misc. JADE .PDFs

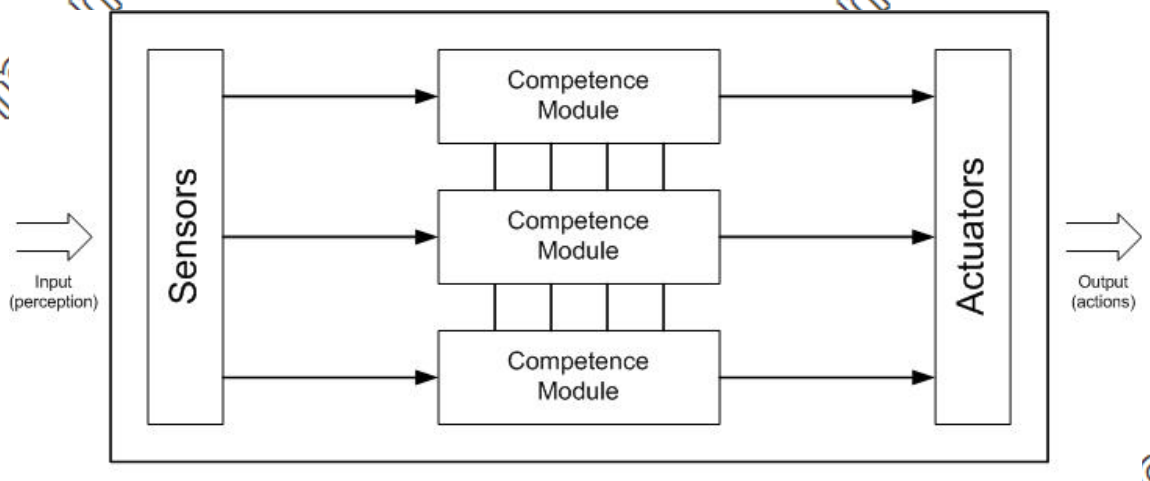
Agent Models: (ways of implementing Agent behavior)

Deliberative: Introduced by Rao and Georgeff. Based on the Belief-Desire-Intention (BDI) model. Has an internal representation of the environment (belief). The efforts undertaken by the agent to attain its goals result from its desires. Desires are typically generated in response to changes in the environment or interactions with other agents; a special process then selects specific desires that will become the agent's intentions for future endeavors. JADE integrates the *Java Expert System Shell* (JESS, see below), which is a rule engine and scripting environment written entirely in *Java*. With JESS, an agent can *reason* on its internal knowledge, use declarative rules and make appropriate decision plans. See also the JADEX (see below) reasoning (belief, goal, plan) engine (http://vsis-www.informatik.uni-hamburg.de/projects/jadex/).



Deliberative Agent Model (ref. Rossier-PhD02.pdf)

Reactive: Proposed by Brooks. Mainly inspired from robotics and reactive systems. Composed of different competence modules linked with sensors (input, perception of the environment) and actuators (output, actions in the environment). Information can be exchanged between the competence modules and the reaction to external changes occurs faster than in the deliberative agent model.



Reactive Agent Model (ref. Rossier-PhD02.pdf)

JADE Message Content Reference Model (message semantics):

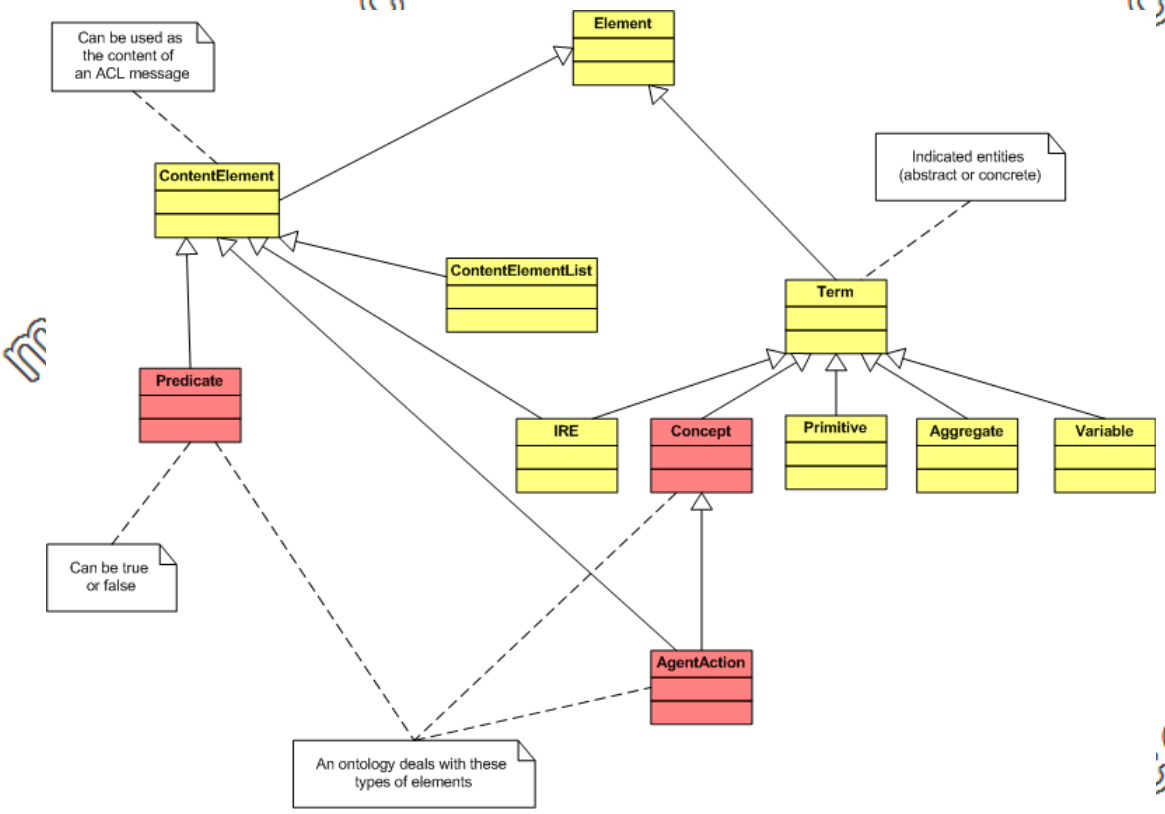


Figure 5.2 JADE ACL Message Content Reference Model

(elements that can appear as the content of an ACL message, within a valid sentence sent by an agent)

ContentElement: Abstract.

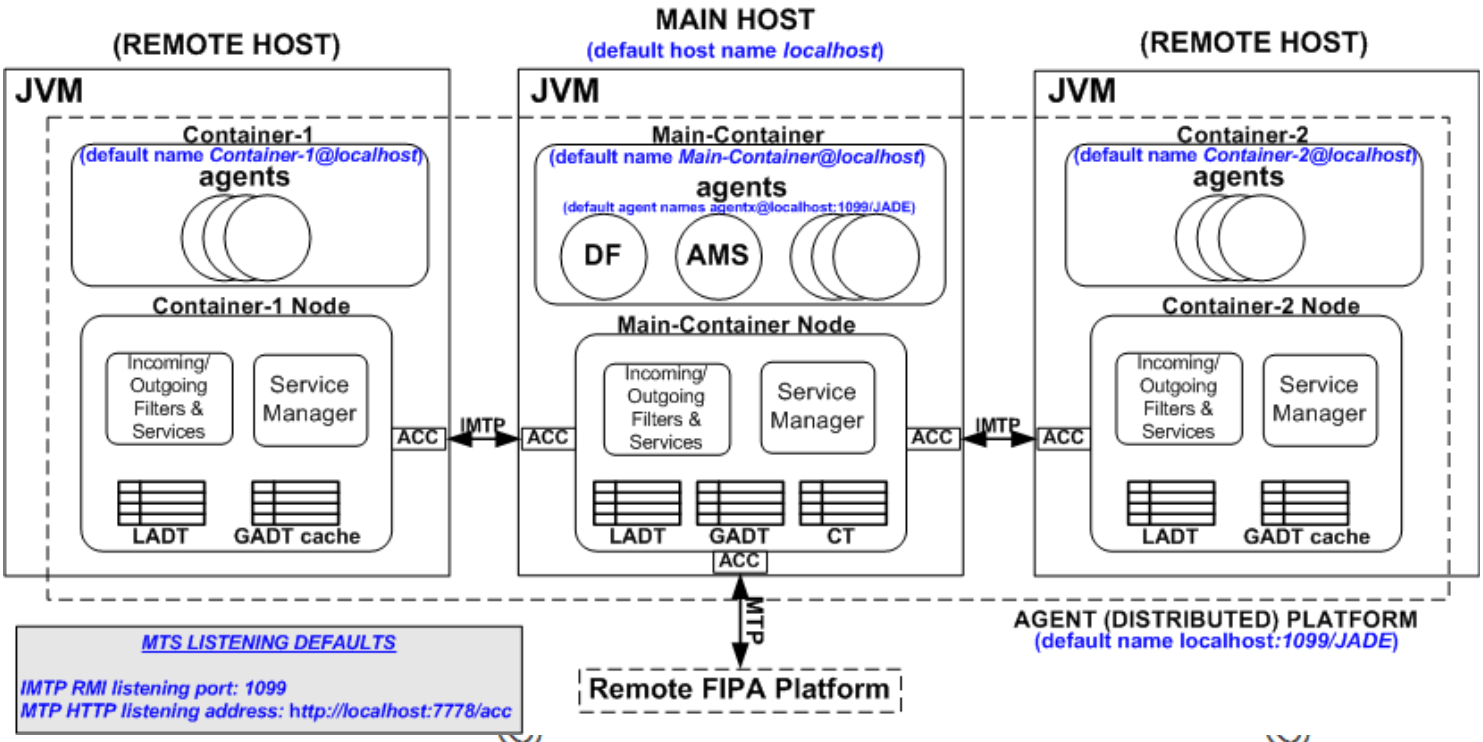
ContentElementList: Class for lists of type Predicate, IRE or AgentAction.

Predicate: Expression that say something about the status of the world and can be true or false. Predicates can be meaningfully used for instance as the content of an INFORM or QUERY-IF message, and would make no sense if used as the content of a REQUEST message. E.g. (Costs :price 100).

Term: Expression identifying entities (abstract or concrete) that "exist" in the world and that agents talk and reason about. Sub-classed into:

- **IRE** (Identifying Referential Expression): Expression identifying the entity (entities) for which a given predicate is true. E.g. (all ?x (Works-for ?x (Company :name "MCS"))). Most typically used in queries, e.g. the content of a QUERY_REF message.
- **Primitive**: Atomic entity, e.g. strings and integers.
- **Concept**: An entity with a complex structure that can be defined in stems of slots, e.g. (Book :title "Document"). Referenced inside predicates and other concepts.
- **AgentAction**: Special Concept indicating an action that can be performed by an agent, e.g. (Sell (Book :title "Document") (Costs :price 100)). Meaningful content of messages such as REQUEST.
- **Aggregate**: An entity that aggregates other entities, e.g. (sequence (Person :name John) (Person :name Bill)).
- **Variable**: Generic entity that is not known a-priori, typically used in a query.

JADE Architecture:



Internal Message Transport Protocol (IMTP): Exchanges messages between agents living in different containers of the same platform, and internal distributed platform commands between platform containers. Java RMI (JADE default) or TCP sockets (JADE LEAP default).

Message Transport Protocol (MTP): Exchanges messages between different FIPA platforms. HTTP, HTTPS, IIOP, JMS and XMPP are the available transport protocols. Includes a standard encoding of the message envelope.

Agent Communication Channel (ACC): Provides a MTP/IMTP and is responsible for sending and receiving messages on an Agent Platform ("**Command Dispatcher**" in JADE-LEAP, implementing JICP(S) (JADE-InterContainer Protocol(SSL))).

Container Table (CT): Registry of the object references and transport addresses of all platform container nodes.

Local Agent Descriptor Table (LADT): Registry of all agents present in this container (incl. current status).

Global Agent Descriptor Table (GADT): Registry of all agents present in the platform (incl. current status and location).

Global Agent Descriptor Table cache(GADT cache): Platform operations involve just the LADT and the sender and receiver agent. If the LADT search fails, the Main Container GADT is searched and cached locally.

Agent ID (AID): A JADE AID object includes of a globally unique ID (GUID) and zero-or-more MTP addresses. AIDs can be generated at startup (in the AgentSpecifier part of the command line, see Appendix A and below), by means of the administration RMA GUI (section 3.7.1), via a code request to the AMS (section 5.5), or by using the JADE in-process interface (section 5.6). The GUID takes the form <local-name>@<platform-name> (NOT AN ADDRESS), where <local-name> may be assigned at startup and must be unique within the platform and <platform-name> may (discouraged) be assigned at startup (-name parameter) or defaulted to the concatenation of the host name (-local-host parameter, defaults to "localhost") and the port number (-port parameter, defaults to "1099") as <host name>:<port number>/JADE. MTP preferred incoming listening addresses (for communicating outside of the AP, with remote FIPA platforms) can be generated at startup (-mtps parameter, see Appendix A.5 and section 3.6), by means of the administration RMA GUI, or passed programmatically to MTPs (section 3.6). By default, the HTTP MTP address is activated only on the main container (http://manncrk:7778/acc).

JADE Notes

ref: Developing Multi-agent Systems with JADE, misc. JADE .PDFs

JADE Administration and Debugging Tools:

Remote Monitoring Agent (RMA): Launched from the command line (-gui). Multiple instances can be launched (different agent name must be associated with each instance). Monitors and administers a distributed JADE platform (one or more hosts and/or containers). Can launch other tools. Left pane represents platform topology as nodes: Agent Platform nodes, Container nodes, Agent nodes. With "Agent Platforms" selected, other (FIPA compliant) APs can be added to the view (access to remote APs is restricted to FIPA-defined actions and management messages). With a specific AP selected (e.g. "manncrk:1099/JADE"), information about that AP can be viewed and MTPs to specific AP containers can be added/removed. With a specific Container selected, new agents can be created and created agents can be added to that Container; MTPs to that Container can be added/removed; the Container can be terminated (terminating the AP). With a specific Agent selected, the Agent can be suspended, resumed, killed, cloned, saved, frozen or migrated to another container; custom ad-hoc messages can be composed and sent to the Agent. See http://jade.tilab.com/doc/tools/rma/html/starting.htm.

Dummy Agent: Launched from the RMA or from a command line. Can send/receive custom ACL messages to/from other agents. See http://jade.tilab.com/doc/tools/DummyAgent/html/intro.htm.

Sniffer Agent: Launched from the RMA or the command line. Multiple instances can be launched. Can take directives from a "sniffer.inf" file (p. 47). For debugging/documenting conversations between agents. Is notified of all platform events and all message exchanges between specified agents. The left pane is the same as the left pane of the RMA, but is used to select the agents to be sniffed. The right pane is a graphical representation of the messages exchanged between sniffed agents. See http://jade.tilab.com/doc/tools/sniffer/html/starting.htm and figure 7.2 (p. 133).

Rockwell Java Sniffer Agent: A stand alone Java application that can also remotely connect to running JADE systems. Is intended as a more powerful alternative to JADE's built-in sniffer. Includes the following agent/message views: (1) Dynamic UML sequential diagram of (filtered) messages between agents; (2) Detail of selected agent messages; (3) High-level plan list (list = set of messages which were sent as a part of CFP (Call For Proposal) protocol or for JDL (Job Description Language) type of messaging to prepare, commit and execute a plan); (4) Workflow detail of selected workflow agent or selected connection (communication) between agents; (5) High-level agent workflow diagram showing, for a selected high-level plan, agents and communication paths between those agents. Also includes a statistical display that summarizes system communications between selected agents and the ability to compose and send messages to agents. Usage: Java Sniffer icon within "JADE-related tools" desktop folder. See figure 7.2, p. 133.

Introspector Agent: Launched from the RMA. Allows the introspection of an agent's execution (i.e. which behaviors are executed and which behaviors pass into the sleeping queue, and the monitoring of reactions to incoming messages). See figure 7.2, p. 133.

Log Manager Agent: Launched from the RMA or the command line. Can take directives from a logging.properties file. Allows the logging levels of each component of the JADE platform to be changed at run-time. See http://jade.tilab.com/doc/tutorials/logging/JADELoggingService.html and file:///C:\JADE-3.6(WADE)\wade\examples\booktrading\cfg\ReadMeLogging.txt.

Event Notification Service: There are four major types of events:

(1) Life-cycle related events (platform-type events): Events related to agent life-cycle changes (born, dead, moved, suspended, resumed, frozen, thawed) and to container life-cycle changes (added, removed).

(2) MTP-type events: Events generated by the platform when an MTP is (de)activated and when a message is sent/received by/from an MTP (i.e. when there is inter-platform communication).

(3) Message-passing-type events: Events generated when an ACL message is sent, received, routed, or posted into the agent message queue. These are the events that the sniffer typically monitors. Only fired on the container where the agent that generates the events. See p. 48 for listening procedure.

(4) Agent-internal-type events: Events which relate to changes in the state of agent behaviors. These are the events that the Introspector typically monitors. Only fired on the container where the agent that generates the events. See p. 48 for listening procedure.

manncrek.com

JConsole plugin: Shows CPU usage per thread. Usage: "jconsole -pluginpath C:\Program %20Files\java\jdk1.6.0_01\lib\toptheads.jar" from DOS window, then click "Local Process", "sum.tools…", "Connect".

Process Explorer: Extends Task Manager – detailed process information. Usage: from desktop "JADE-related tools" folder. Appears in task bar.

JADE Agent Execution Thread: Each agent has its own thread, behaviors may be on separate threads (not default).

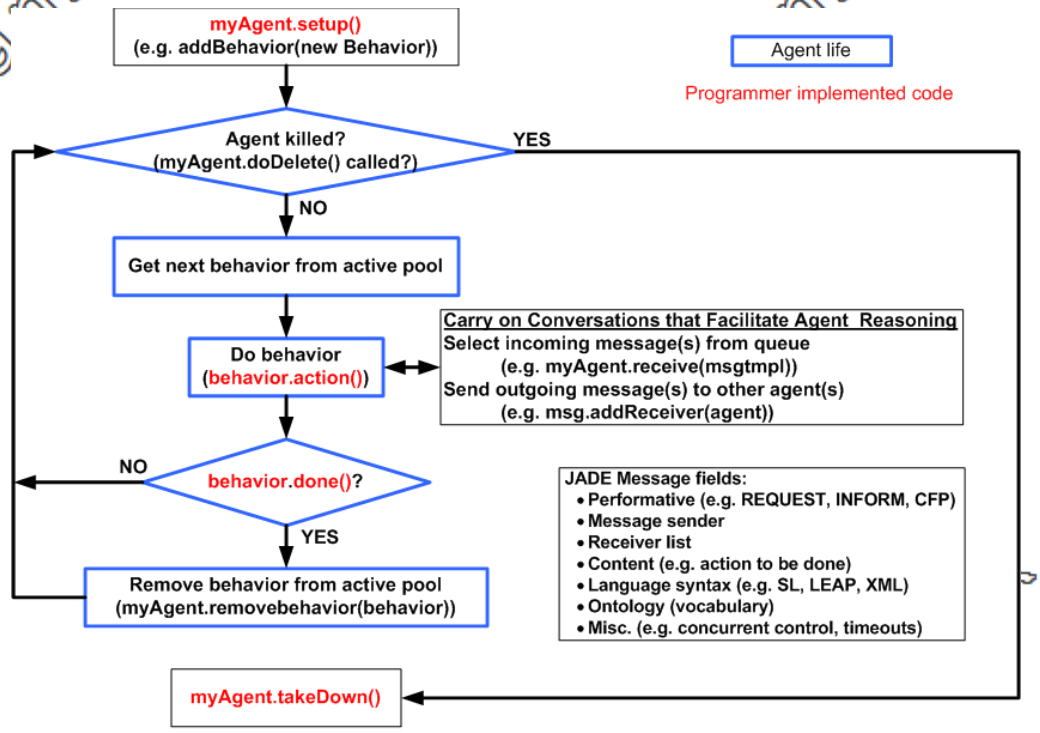
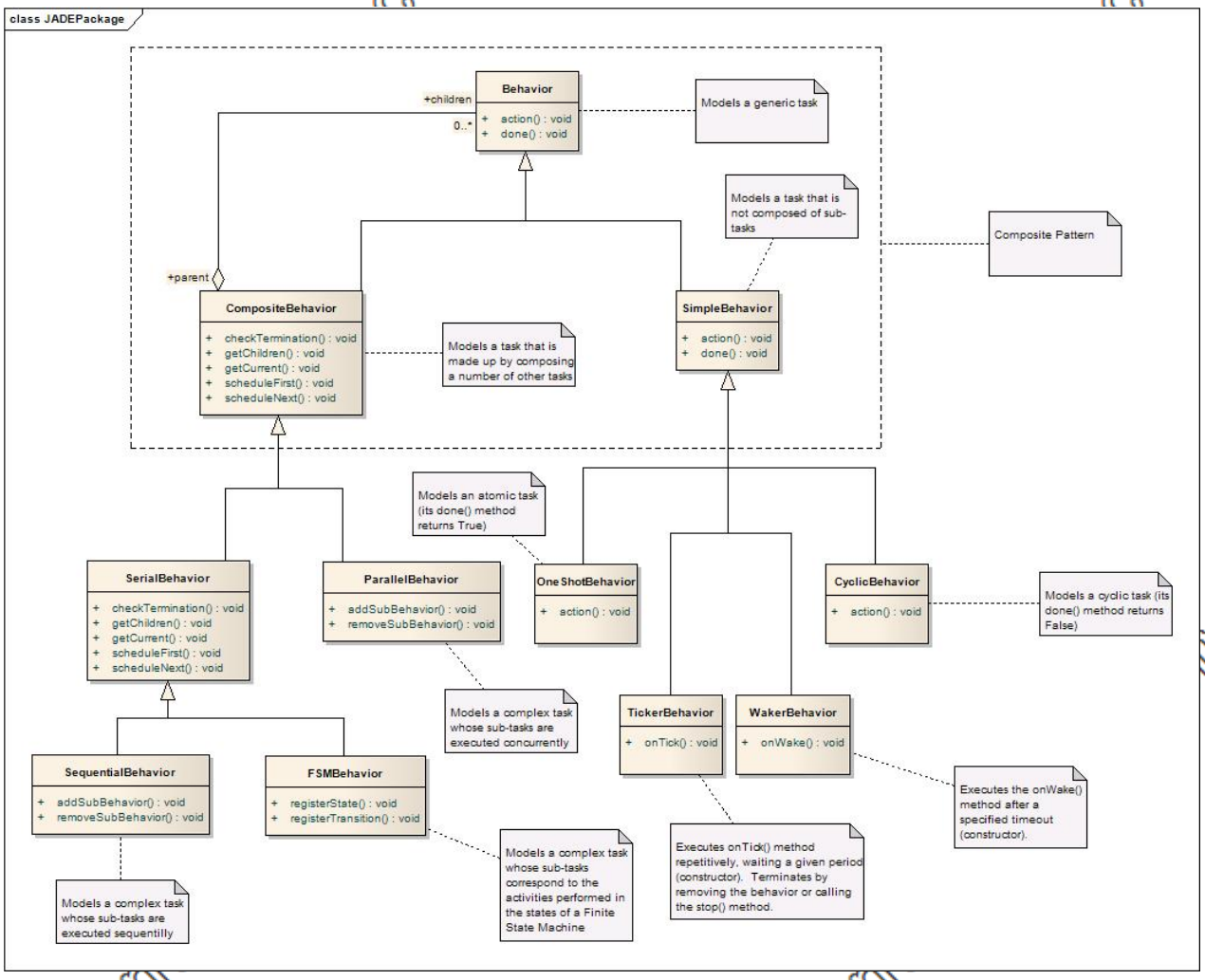


Figure 4.3 JADE Agent Execution Thread

JADE Task Behaviors: (UML class diagram)



JADE Content / Language / Ontology Development Process (Section 5.1.3):

(1) Define an ontology that includes schemas for the types of **predicates**, **agent actions** and **concepts** that are pertinent to the addressed domain (carried out manually or with Protégé).
(2) Develop proper Java **classes** for all types of **predicates**, **agent actions** and **concepts** in the ontology (carried out manually or with Protégé / Bean Generator).
(3) Select a suitable content language from among those directly supported by JADE (SL, LEAP, XML).
(4) Register the defined ontology (classes) and the selected content language with the agent.
(5) Create and handle content expressions as Java objects that are instances of the classes developed in step 2, and let JADE translate these Java objects to/from strings or sequences of bytes that fit the content slot of ACLMessages.

Mobile Agent Characteristics (Chapter 6 sections):

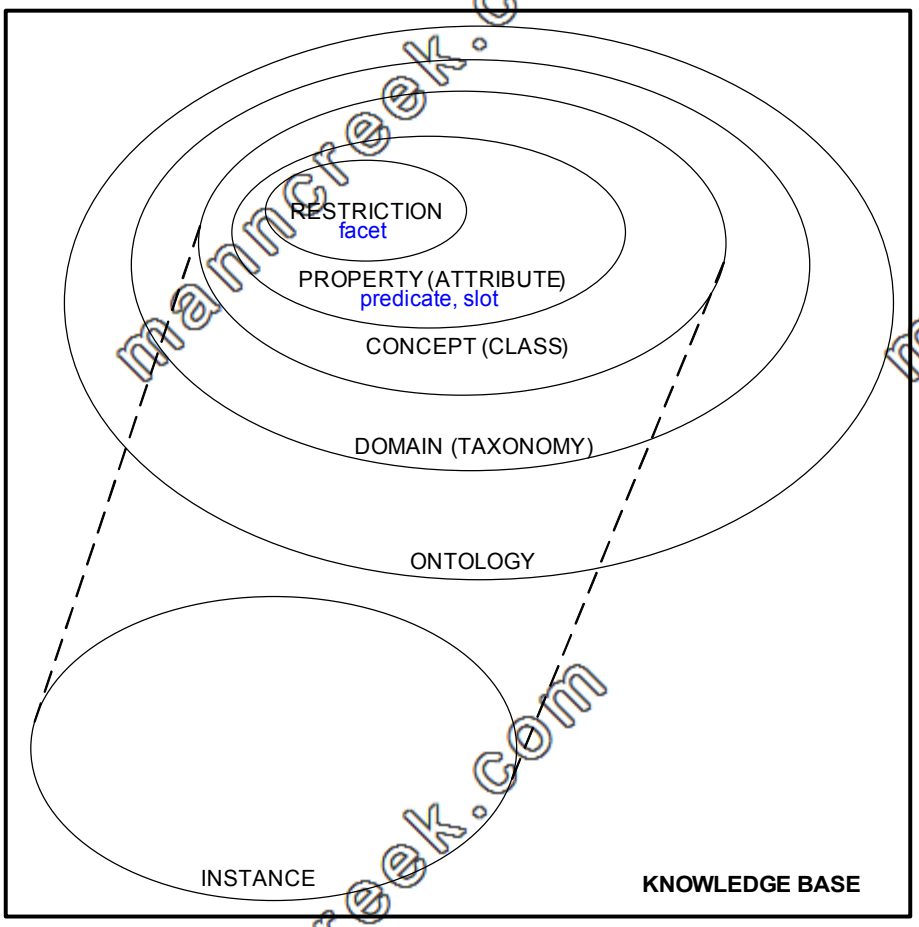
Includes the Agent characteristics listed on page 2 (above), plus the ability to migrate its code, data and state from one platform to another (intra-platform mobility) or between containers on the same platform (inter-platform mobility). Agent code is that part of the agent that executes at the new or old location. Agent data represents the knowledge the agent has accumulated, collected in agent variables. Agent state is the agent execution environment within which the agent executes, including any CPU registers, counters and stacks. "Strong migration" implies capturing an agent's data, code and state at some instant in time (representing one of an infinite number of agent states), migrating that capture to a new place and then continuing agent execution at the new location. "Weak migration" implies capturing an agent's data at some state (one state of a possible finite number of agent states), migrating that capture to a new place and continuing agent execution at the new location. Agent capture may be static (determined at agent creation time) or dynamic (determined during agent execution). Advantages of mobility include: (1) asynchronous / independent behavior, (2) fault tolerance and (3) ability to move to data. Disadvantages of mobility include: (1) increased complexity through interoperability / portability, (2) decreased security and (3) lack of JADE-LEAP support. Intra- and inter-platform mobility are supported by JADE.

JADE with Eclipse/MyEclipse:

Add a Java project through the Package Explorer in the "Java" (or "MyEclipse J2EE") perspective.
Add appropriate libraries into project (right-click project>Properties>Java Build Path>Libraries>Add External JARs)
JRE System Library (may already be there, depending on Workspace settings)
crimson.jar (from jade-x.x.x\jade\lib)
http.jar (from jade-x.x.x\jade\lib)
iiop.jar (from jade-x.x.x\jade\lib)
jade.jar (from jade-x.x.x\jade\lib)
jadeTools.jar (from jade-x.x.x\jade\lib)
Add source files to source folder.
Change project properties if necessary (right-click project>Properties).
Setup project run characteristics (right-click project>Run As>Run…):
Main TAB:
Fill in Launch Name.
Set Main class to "jade.Boot".
Check "Include libraries when searching for a main class".
Arguments TAB:
Enter program arguments -- e.g. for the Book Trading Demo:
-nomtp -gui
bookSeller1:bookTrading.seller.BookSellerAgent
bookSeller2:bookTrading.seller.BookSellerAgent
bookBuyer1:bookTrading.buyer.BookBuyerAgent
bookBuyer2:bookTrading.buyer.BookBuyerAgent
sniffer:jade.tools.sniffer.Sniffer(df;b*) <or the Rockwell sniffer>
Specify the working directory. (If Eclipse compiles the project to a directory different from the Jade source tree, you have to specify where the compiled (object) files are.)
Run or Debug the program either from the Run/Debug icons at the top of the window or by right-clicking the project and selecting Run As>Run…>Run or Debug As>Debug…>Debug.

mannCreek.com

ONTOLOGY CONCEPTS



Knowledge Base: An ontology plus a set of instances.

Ontology: A formal and explicit description of concepts in a domain of discourse (problem domain). A theory about what types of things exist. E.g. library.

Taxonomy (Domain): A particular area of knowledge. A structured (subclass-superclass, "is-a" relationship) way to organize knowledge. E.g. Dewey decimal system.

Class (Concept): Objects that interact in a Domain. E.g. Book.

Property: Attributes assigned to a class. E.g. language, author.

Restriction: Property constraints (e.g. cardinality, range). E.g. English.

Instance: Class object realization (e.g. *Developing Multi-Agent Systems with Jade* is an instance of the class Book).

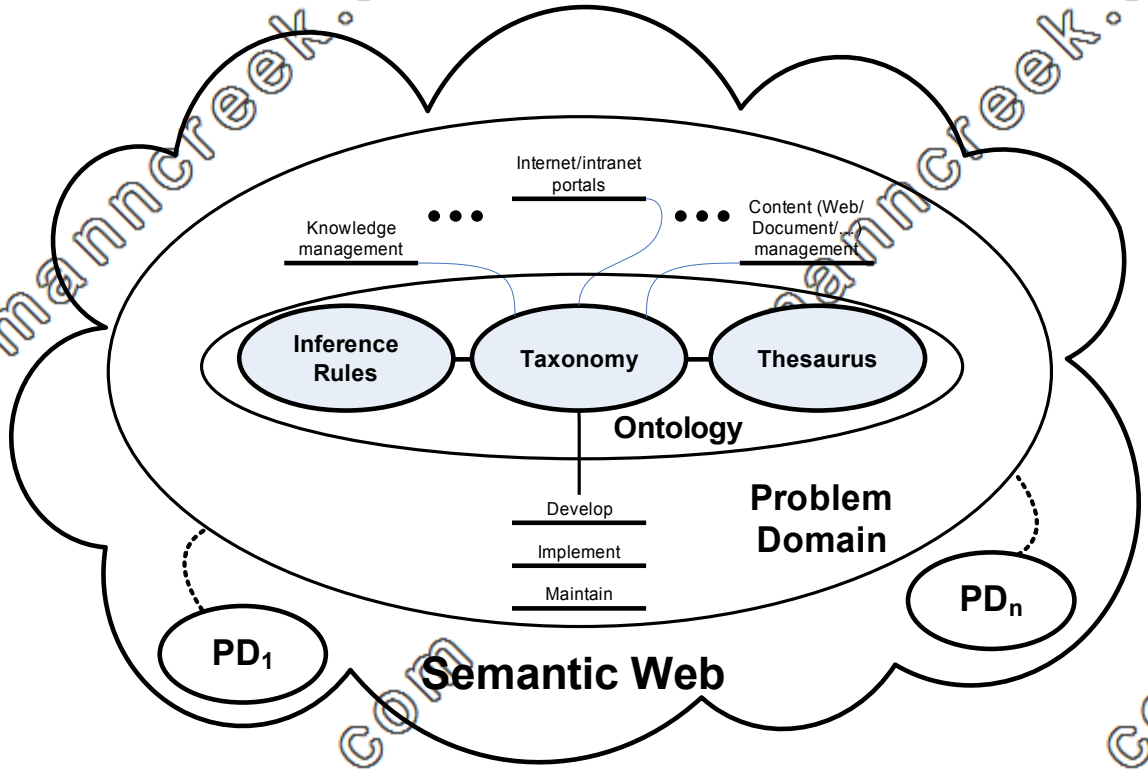
OWL (Web Ontology Language): A language for defining ontologies. Defines the types of relationships that can be expressed in RDF.

RDF (Resource Description Framework): A language for describing instances of concepts (classes) such that information exchange is standardized.

RDF Statement: A triplet composed of subject(instance name)-predicate(property name)-object(value) (e.g. rdf.html (subject), author (predicate), value John Doe (object)).

manncreek.com

SEMANTIC WEB ONTOLOGY CONCEPTS:



Semantic Web: Intended to become the next generation of the Internet, where Internet components understand the meanings of words and relationships. The Semantic Web is a kind of Semantic Network. Semantic Networks use nodes and arcs to represent structured knowledge, where the nodes are concepts or entities and the arcs are relationships and associations.
Problem Domain: In the partitioning of a problem space into components to be considered and not to be considered, the problem domain is those components to be considered. The problem domain also considers any connections to external entities that might influence it.
Ontology: Theory about what types of things exist within a problem domain. An ontology has a taxonomy, a thesaurus and a set of inference rules. WRT Agents: to make agents understand each other they have to not only speak the same language, but also have a common ontology. An ontology is a part of the agent's knowledge base that describes what kind of things an agent can deal with and how they are related to each other. See http://en.wikipedia.org/wiki/Software_agent.
Taxonomy: Structured ways to organize domain knowledge (e.g. life ⇒ fauna ⇒ mammal ⇒ canine ⇒ dog ⇒ spaniel ⇒ cocker ; Dewey Decimal system) so it can more easily be understood and shared.
Inference Rules: Premises known or assumed to be true from which logical domain conclusions are derived. The act of reasoning from factual domain knowledge or evidence.
Thesaurus: A detailed vocabulary. Information on preferred and related domain terms, and synonyms.
Taxonomy development: Choose some combination of software (automatic) and human classification (usually statistical, but my include natural language processing based on thesaurus / dictionary); choose existing off-the-shelf or new in-house taxonomy; balance taxonomy width and depth to maximize simplicity and understanding (juggle number/distribution of nodes); use skills of (corporate) librarians.
Taxonomy implementation: Prototype with major go/no-go decision points; involve stake holders at all levels; increment and iterate, including repeated verification and validation.
Taxonomy maintenance: Include user feedback; update topic maps constantly based on usage and changing terminology; use automation as much as possible but allow for human override.

JADE Notes

ref: Developing Multi-agent Systems with JADE, misc. JADE .PDFs

AGENT/JADE ONTOLOGY CONCEPTS:

Ontology: A formal explicit description of concepts in a domain of discourse (**classes** (sometimes called **concepts**)), properties of each concept describing various features and attributes of the concept (**slots** (sometimes called **roles** or **properties**)), and restrictions on slots (**facets** (sometimes called **role restrictions**)). An ontology together with a set of individual **instances** of classes constitutes a **knowledge base**. In reality, there is a fine line where the ontology ends and the knowledge base begins. In practical terms, developing an ontology includes:

- defining classes in the ontology,
- arranging the classes in a taxonomic (subclass–superclass = "is-a" relationship) hierarchy,
- defining slots and describing allowed values for these slots.

A knowledge base is then created by defining individual instances of these classes, filling in specific slot value information and additional slot restrictions. An ontology creation methodology (an iterative process):

(1) Determine the ontology domain and scope.
(2) Seek out existing ontologies to use.
(3) Enumerate important terms.
(4) Define classes (unary predicates) and class ("is-a") hierarchy (top-down, bottom-up, combination).
(5) Define class properties (slots (binary predicates)).
(6) Define slot facets (features of the slot values). E.g. NAME could be a property (slot) with features cardinality (one) and type (String) (there also are domain and range features).
(7) Create ontology instances.

WRT Agents: to make agents understand each other they have to not only speak the same language, but also have a common ontology. In this context, an ontology is a part of the agent's knowledge base that describes what kind of things an agent can deal with and how they are related to each other. See http://en.wikipedia.org/wiki/Software_agent. An ontology for a given domain is a set of schemas (see below) defining the structure of the predicates, agent actions and concepts (basically their names and their slots) that are pertinent to that domain.

Schema: A minimal and specialized ontology (e.g. a library card catalogue as opposed to a physical library).

Protégé: A knowledge acquisition and ontology development tool (http://protege.stanford.edu/). There is a Protégé plug-in (Bean Generator, ref. *Developing Multi-Agent Systems with Jade*, chapter 13) for creating message content ontologies compliant with JADE. Reference JADE *Tutorial Application-Defined Content Languages and Ontologies* for the JADE/Protégé procedure and an example application (Section 6 and Appendix D). Protégé usage: from desktop "JADE-related tools" folder.

JESS: Java Expert System Shell; a rule engine and scripting environment with forward chaining and production rules (http://herzberg.ca.sandia.gov/jess/) that can be integrated into JADE. With JESS, an agent can reason on its internal knowledge, use declarative rules, and make appropriate decision plans. There is a JESS Protégé plug-in (JessTab, ref. JessTab(JADE_JESS_PROTEGE).ppt). JESS usage: not yet downloaded from free site (see book *JESS in Action*).

JADEX: Allows JADE agent developers to design and implement JADE agents which exhibit goal-oriented (belief, goal, plan) behavior as opposed to task-oriented behavior (JADEX requires an explicit representation of goals).

manncreek.com