JESS
Java Expert System Shell
Framework

Integrated Master's in Informatics Engineering

Intelligent Agents

2018/2019

Synthetic Intelligence Lab

Filipe Gonçalves

César Analide

Useful Links

- http://www.jessrules.com/jess/download.shtml
- http://diuf.unifr.ch/drupal/sites/diuf.unifr.ch.drupal.softeng/files/teaching/studentprojects/vogt/download/J2J.pdf
- http://www.jessrules.com/jess/docs/
- http://jade.tilab.com/documentation/tutorials-guides/integrating-jade-and-jess/
- http://www.jessrules.com/docs/71/library.html
- http://www.jessrules.com/docs/71/embedding.html

Expert Systems

- An AI branch
- Simulation of human reasoning in a domain
- <u>Rule-Based Expert Systems are the most used to:</u>
  - Simulate human reasoning using heuristic knowledge
  - Problem data stored as facts
  - Reasoning based on rules of type ***IF … THEN …***

Rule-Based Systems

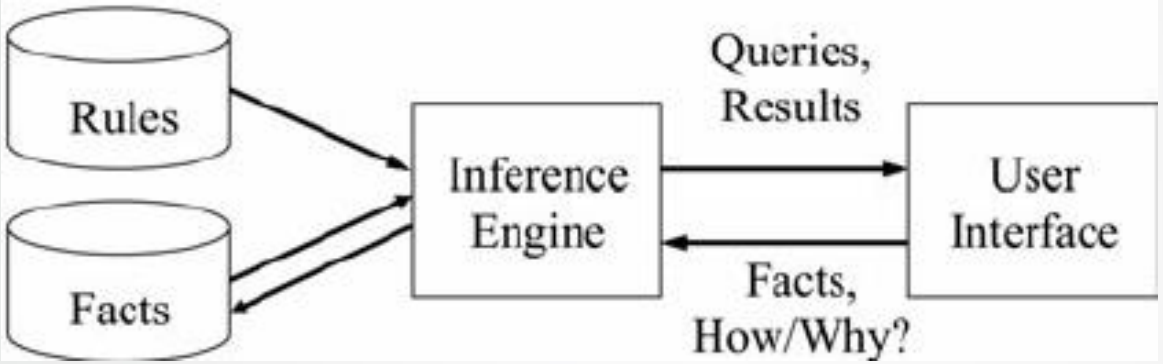
- Contain rules for a certain domain:
  - Knowledge not necessarily expert
  - Examples:
    - Definition of "business rules"
    - Decision components (e.g. in computational agents)

- Advantages:
  - Intuitive representation of knowledge
  - Division between knowledge and its application
  - Changes do not imply recompilation
- Paradigm of declarative programming:
  - Definition of independent rules
  - Non-sequential execution
  - Interpreter decides when to apply which rules

Rule-Based Systems

- Chaining (execution) of rules:
  - **Backward-chaining**
    - Goal-driven: how to prove a goal?
    - Logical programming languages (e.g. Prolog)
  - **Forward-chaining**
    - Data-driven: what to do when a fact arises?
    - Production systems (e.g., CLIPS, JESS)

Rule-Based Systems



- Inference Engine: decides when to apply what rules; controls the activation and selection of rules
- Knowledge Base: saves the set of rules; rules follow the pattern ***P1, ..., Pm →Q1, ..., Qn***, where if the premises/ conditions ***P1, ..., Pm*** are True, actions/conclusions ***Q1, ..., Qn*** will be executed
- Work Memory: saves facts and intermediate results that make up the current state of the problem; facts can be examined and modified by the rules

Rule-Based Systems

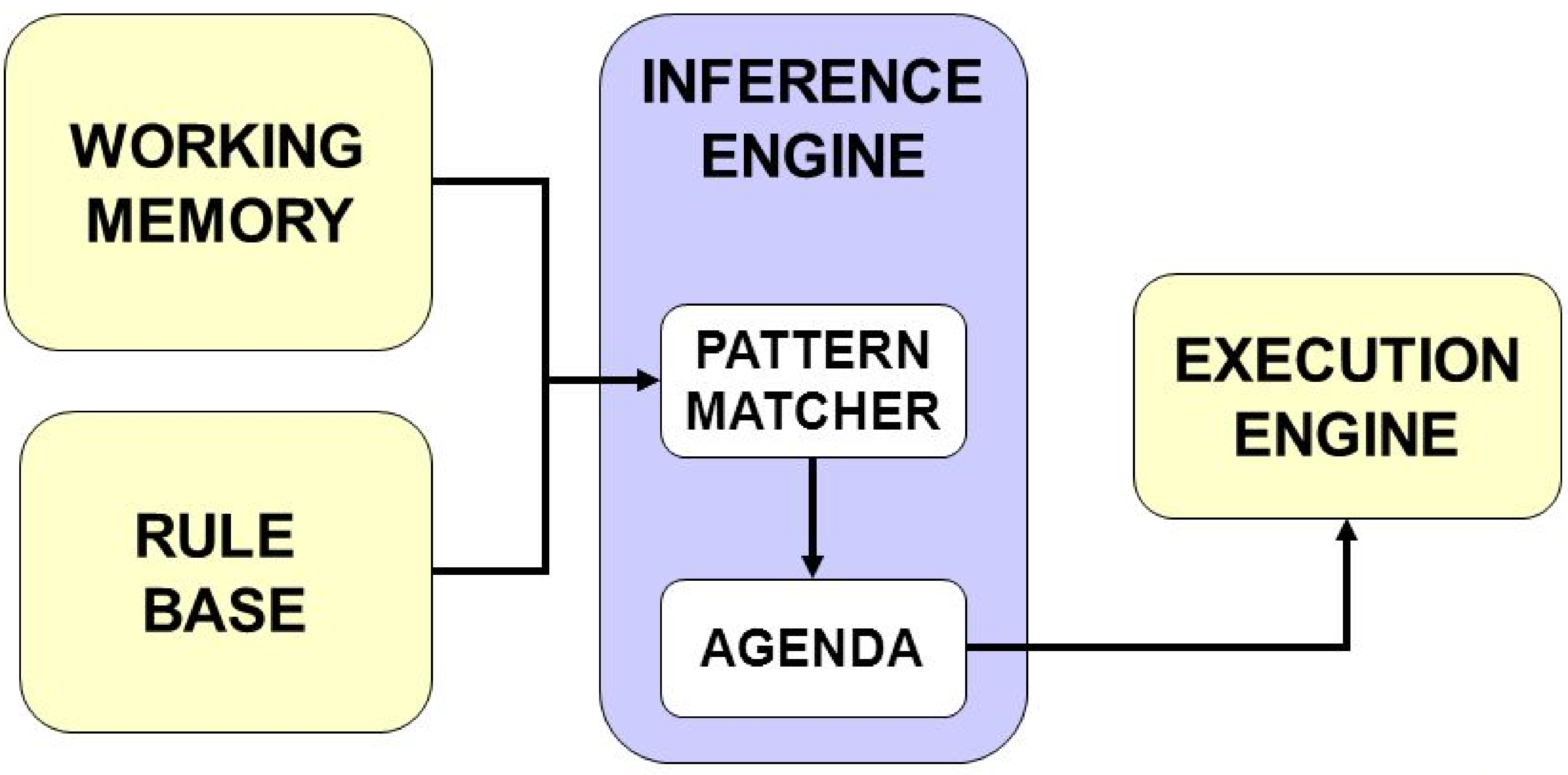
- The inference engine works in a cyclic way, decomposing into three phases:

1. Match Phase:
    - Groups the rules whose premises/conditions are satisfied by the work memory: the rules are instantiated with facts that make their premises true
    - Obtain the set of conflicts
2. Conflict Resolution Phase:
    - Selection of the rule to be performed, according to a strategy of conflict resolution (e.g. through priority)
3. Action Phase:
    - Sequential execution of the actions/conclusions presented in the selected rule
    - Actions / conclusions can modify Knowledge Base

JESS

- A rule engine that very efficiently applies rules to date
- Inspired by the AI production rule language CLIPS (LISP-like syntax)
- Fully developed Java API for creating rule-based expert systems
- How does Jess work?
  - Jess matches facts in the fact base to rules in the rule base
  - The rules contain function calls that manipulate the fact base and/or other Java code
  - Jess uses the Rete algorithm to match patterns

JESS Architecture



JESS Language

- LISP-like syntax
- Can be used to script Java API: JAVA →JESS
- Can be used to access JavaBeans: JESS → JAVA
- Can create Java objects and access its methods from within Jess!

JESS Language - Basics

- Symbols:
  - identifiers: letters, digits and $*=+-/<>_?#.
  - Case-sensitive
  - Special Symbols: nil TRUE FALSE
- Numbers
- Strings: delimited by “”

- Lists:
  - Delimited by (e)
  - Contain zero or more symbols, numbers, strings, or other lists
  - Ex: *(+ 3 2) (a b c) ("Hello, World") ()*

    (deftemplate foo (slot bar))
  - The first element of the list is called the head of the list
- Comments:
  - All text after following a “;”
  - Comments code: / * … * /

JESS Language - Functions

- Function calls (whether predefined or defined by the user) are lists
  - Notation prefix: the head of the list is the name of the function
  - Example:

```
Jess> (+ 2 3)
5
Jess> (+ (+ 2 3) (* 3 3))
14
Jess> (printout t "Answer is " 42 "!" crlf)
Answer is 42!
Jess> (batch examples/hello.clp)
```

JESS Language - Variables

- Identifiers that start with “?”
  - Can contain a symbol, number or string, or a list
  - Can be assigned a value through the function bind

```
Jess> (bind? V "The value")
Jess> (bind? Grocery-list (list eggs bread milk))
```

- Check the value of a variable

```
Jess> (bind? A 123)
Jess>? A
123
```

- Variables are not declared before they are first used
  - Exception: you can create global variables, which are not destroyed during a *reset - defglobal*

JESS Language – Function Definition

(*deffunction* <func-name> [<doc-comment>] (<parameter>*)
<expr>* [<return-specifier>])

- Example: *(deffunction max (?a ?b) (if (> ?a ?b) then (return ?a)*

  else (return ?b)))

- Flow control Funtions: *foreach*, *if*, *while*, ...
- Call:

  Jess> (printout t "Greater of 3 and 5 is " (max 3 5) "." crlf)

  Greater of 3 and 5 is 5.

JESS Language – Ordered Facts

- <u>Ordered Facts:</u>
  - lists in Jess
  - the head of the list serves as a sort of category
- Examples:

  (shopping-list eggs milk bread)

  (person "Bob Smith" Male 35)

  (father-of danielle ejfried)
- Assertion / retraction of facts: *assert / retract*
- Visualization of existing facts: *facts*
- Clear all facts: *clear*

JESS Language – Ordered Facts (Examples)

```
Jess> (reset)
TRUE
Jess> (assert (father-of danielle ejfried))
<Fact-1>
Jess> (facts)
f-0 (MAIN::initial-fact)
f-1 (MAIN::father-of danielle ejfried)
For a total of 2 facts in module MAIN.
Jess> (retract (fact-id 1))
TRUE
```

```
Jess> (facts)
f-0 (MAIN::initial-fact)
For a total of 1 facts in module MAIN.
```

Note: the fact (initial-fact) is created by the *reset* command

JESS Language – Unordered Facts

- Unordered Facts:
  - Allows to structure information

- Examples:

(person (name "Bob Smith") (age 34) (gender Male))

(automobile (make Ford) (model Explorer) (year 1999))

- Each fact has an associated *template* that defines its *slots*

```
(deftemplate <template-name> [extends <template-name>]
    [<doc-comment>]
    [(declare ...)]
    [(slot | multislot <slot-name>
        [(type <typespec>)
        [(default <value>)]
        ...
    )]*
)
```

JESS Language – Unordered Facts (Examples)

```
Jess> (deftemplate automobile
        "A specific car."
        (slot make)
        (slot model)
        (slot year (type INTEGER))
        (slot color (default white)))
Jess> (assert (automobile (make Chrysler)
(model LeBaron) (year 1997)))
<Fact-0>
```

```
Jess> (facts)
f-0 (MAIN::automobile (make Chrysler) (model
LeBaron) (year 1997) (color white))
For a total of 1 facts in module MAIN.
```

JESS Language – Unordered Facts (Examples)

- *slot* that can hold multiple values: *multislot*

```
Jess> (deftemplate box (slot location) (multislot contents))
TRUE
Jess> (bind ?id (assert (box (location kitchen) (contents spatula sponge frying-pan))))
<Fact-1>
```

Note: the variable ?id was associated with the identifier of the fact

- Change the values of a slot: *modify*

- Extension of *deftemplate*:

```
Jess> (deftemplate used-auto extends automobile (slot mileage)
            (slot blue-book-value)
            (multislot owners))
TRUE
```

JESS Language – Deffacts

- Deffacts:
  - Allows to define grouped facts that are created when invoking the reset command
  - Example:

```
Jess> (deffacts my-facts "The
documentation string" (foo bar)
(box (location garage) (contents scissors
paper rock))
(used-auto (year 1992) (make Saturn)
(model SL1)
(mileage 120000) (blue-book-value 3500)
(owners ejfried)))
TRUE
```

```
Jess> (reset)
TRUE
Jess> (facts)
f-0 (MAIN::initial-fact)
f-1 (MAIN::foo bar)
f-2 (MAIN::box (location garage) (contents
scissors paper rock))
f-3 (MAIN::used-auto (make Saturn) (model
SL1) (year 1992)
(color white) (mileage 120000)
(blue-book-value 3500) (owners ejfried))
For a total of 4 facts in module MAIN.
```

JESS Language – Shadow Facts

- Shadow Facts:
  - Unordered facts that map Java objects
  - A Java object can be placed in memory of work
- Shadow Facts templates:

  (deftemplate <template-name>

  (declare (from-class <class-name>)))
- Alternative:

  (defclass <template-name> <class-name>)

The created template has slots corresponding to the JavaBeans properties of the class:

```
public class ExampleBean {
    private String name = "Bob";
    public String getName()
        { return name; }
    public void setName(String s)
        { name = s; }
}
```

JESS Language – Shadow Facts

- Java Object creation:

  (bind <var> (new <class-name>))

- Shadow Fact creation:

  (add <Java object>)

  - if it does not already exist, the template is created automatically

- Alternative:

  (definstance <template-name> <Java object>)

JESS Language – Shadow Facts (Examples)

- Template creation:

```
Jess> (defclass ExampleBean ExampleBean)
ExampleBean
Jess> (ppdeftemplate ExampleBean)
        "(deftemplate MAIN::ExampleBean
        \"$JAVA-OBJECT$ ExampleBean\"
        (declare (from-class ExampleBean)))"
```

- Java object creation in Work Memory (shadow fact):

```
Jess> (bind ?x (new ExampleBean))
(Java-Object::ExampleBean
Jess> (add ?x)
<Fact-0>
Jess> (facts)
f-0 (MAIN::ExampleBean
(class <Java-Object:java.lang.Class>)
(name "Bob")
(OBJECT <Java-Object:ExampleBean>))
For a total of 1 facts in module MAIN.
```

JESS Language – Java Objects (Examples)

- Criation and use example of Hashtable:

```
Jess> (bind ?ht (new java.util.Hashtable))

<Java-Object:java.util.Hashtable>

Jess> (call ?ht put "key1" "element1")

Jess> (call ?ht put "key2" "element2")

Jess> (call ?ht get "key1")

"element1"
```

- Manipulation of member variables:

```
Jess> (bind ?pt (new java.awt.Point))

<Java-Object:java.awt.Point>

Jess> (set-member ?pt x 37)

Jess> (set-member ?pt y 42)

Jess> (get-member ?pt x)

37
```

JESS Language – Rule definition

```
(defrule <rule-name> [<doc-comment>]
      [<fact-pattern>]* => [<function-call>]*)
```

- Example:

```
Jess> (deftemplate person
      (slot firstName) (slot lastName) (slot age))
Jess> (defrule welcome-toddlers
            "Give a special greeting to young children" (person {age < 3}) => (printout t "Hello, little
one!" crlf) )
Jess> (assert (person (age 2)))
Jess> (run)
Hello, little one!
```

JESS Language – Rules use Standards

- Boolean expressions to evaluate slot content - within {}

  < <= > >= == != <> && ||

- Variable for reference after slot value - within ()

- Examples:

```
Jess> (defrule teenager ?p <- (person {age > 12
&& age < 20} (firstName ?name))

=> (printout t ?name " is " ?p.age " years old."
crlf) )

Jess> (assert (person (age 15) (firstName
Maria)))

Jess> (assert (person (age 18) (firstName Paul)))

Jess> (run)

Paul is 18 years old.

Maria is 15 years old.
```

JESS Language – Rules use Standards

- Testing slots:
  - Literals
  - Variables (possibly not free)
  - & (and) | (or) ~ (not)
  - : (happens if the next function returns *TRUE*)
    - *(coord ?X&:(> ?X 10) ?)*
  - = equality between slot value and function return
    - *(coord ?X =(+ ?X 1))*
  - regular expressions surrounded by /.. /
    - *(person (firstName /A.*/))*
- *Other Examples:*
  - *(coord ?X ?X)*
  - *(coord ?X ?Y&~?X)*
  - *(coord ?X&~10 ?)*
  - *(coord ? 10|20)*
  - *(coord $?both) ; multislot*

JESS & Java

JESS API:

- Classes
  - jess.Context
  - jess.Jesp
  - jess.JessException
  - **jess.Rete**
  - jess.Value
  - jess.ValueVector
  - …
- Interfaces
  - jess.Userfunction
  - …

JESS & Java

Class *jess.Rete*:

- To access the Inference Engine, the *jess.Rete* object of this class must be called

Functions:

- *run()*, *reset()*, *clear()*, *assertFact()*, *retract()*, …
- Execution of Jess command: *eval()*
- Add Jess's invoked functions into Java: *addUserfunction()*

jess.Userfunction Interface:

- Definition of Java functions invoked in Jess
- getName() and call() methods

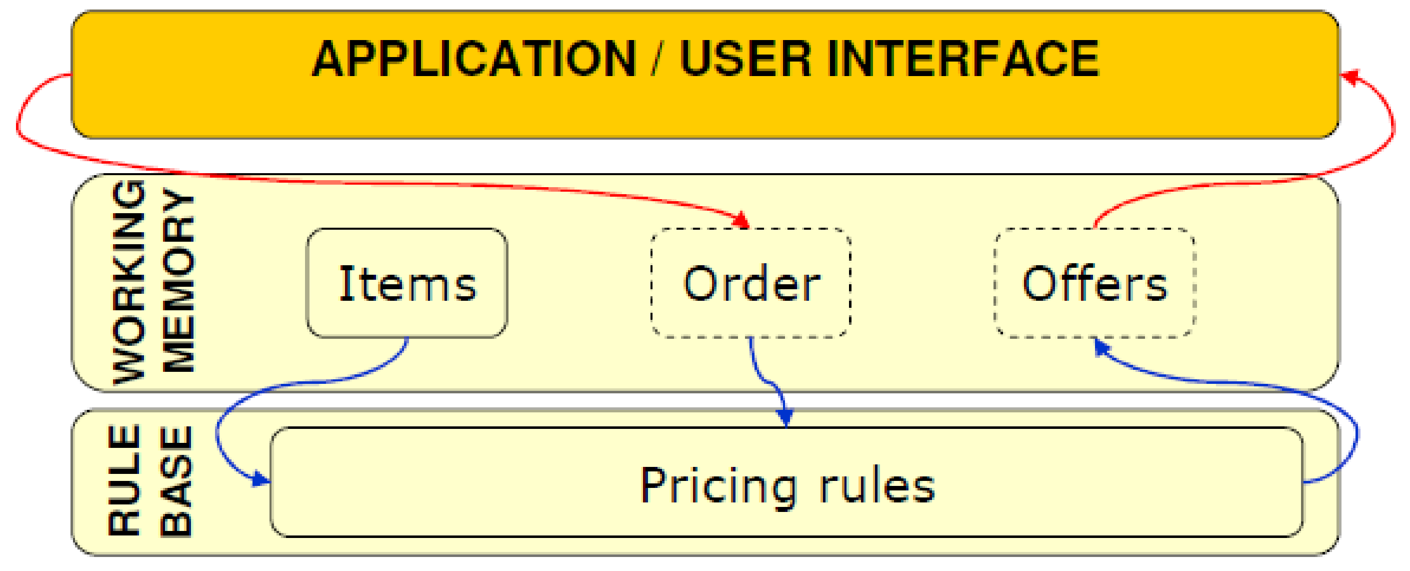
JESS & Java

```
import jess.*;
public class ExSquare {
        public static void main(String[] str) {
                try {
                        Rete r = new Rete();
                        r.eval(" (deffunction square (?n) (return (* ?n ?n)) ) ");
                        Value v = r.eval("(square 3)");
                        System.out.println(v);
                } catch (JessException ex) {
                        System.err.println(ex);
}}}
```

JESS & Java (Example)

Approach to determining discounts and offers in a store:

1. Create a Jess engine with the rules to apply and with the product data in catalogue
2. When an order arrives:
   - Add the order data to the working memory
   - Run the inference engine
   - Obtain from the Work Memory facts added by the rules



JESS & Java (Example)

Rules define Examples:

```
(defrule 10%-volume-discount
    "Give 10% discount to everybody who spends more than €100."
    (Order {total > 100})
        =>
    (add (new Offer "10% volume discount"
    (/ ?total 10))) )
```

```
(defrule 25%-multi-item-discount
    "Give 25% discount on items customer buys 3 or more of."
    (OrderItem {quantity >= 3} (price ?price))
        =>
    (add (new Offer "25% multi-item discount"
    (/ ?price 4))) )
```

JESS & Java (Example)

```
public class PricingEngine {
    private Rete engine;
    private WorkingMemoryMarker marker;

    // Constructor
    public PricingEngine(Database database)
throws JessException {
        // Create a Jess rule engine
        engine = new Rete();
        engine.reset();

    // Load the pricing rules
    engine.batch("pricing.clp");

// Load the catalog data into working memory
    engine.addAll(database.getCatalogItems());
    // Mark end of catalog data for later
    marker = engine.mark();
    }
```

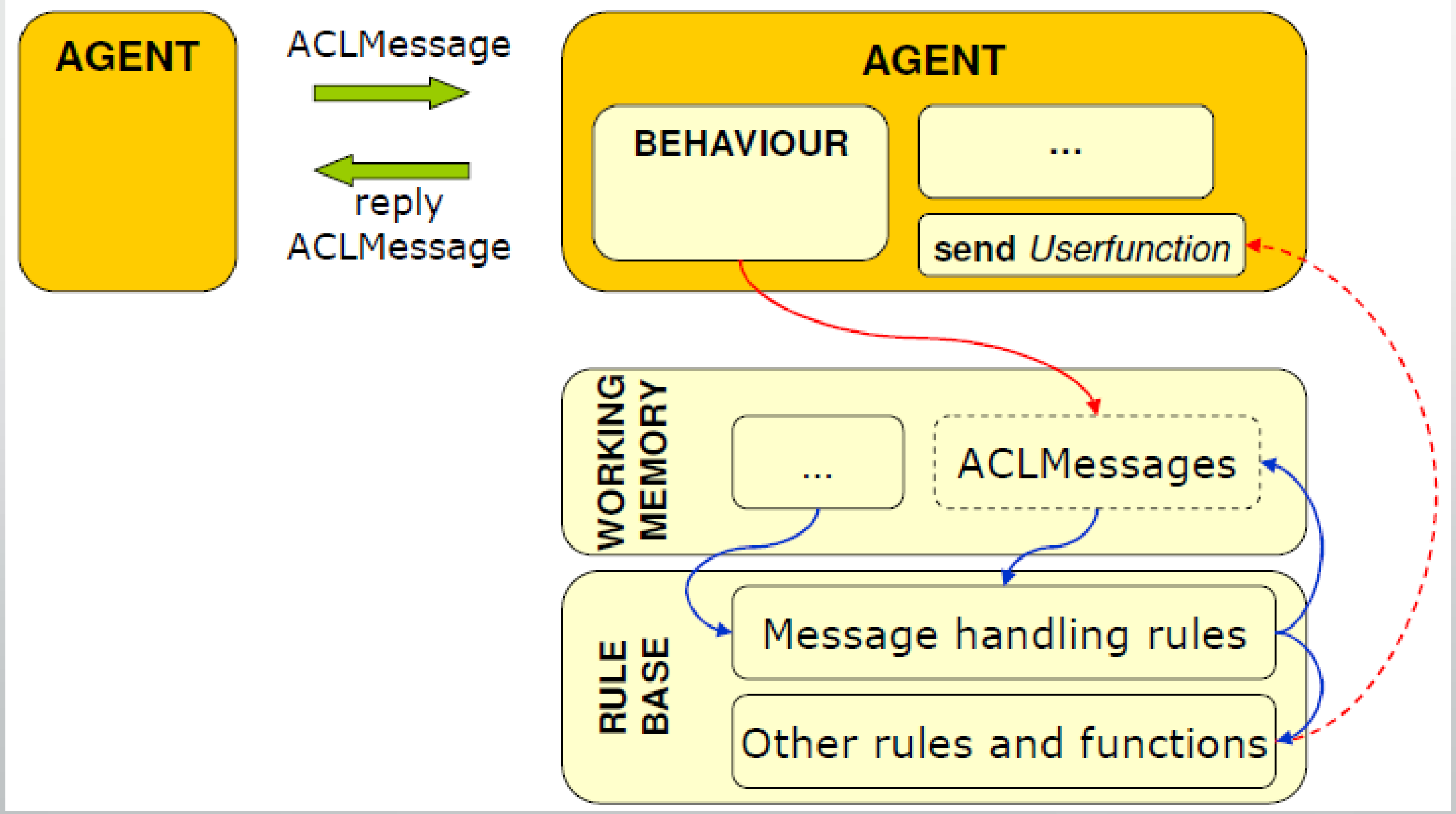
JESS & Java (Example)

```
// Method for handling a new order
public Iterator run(Order orderNumber) throws JessException {
// Remove any previous order data, leaving
// catalog data
    engine.resetToMark(marker);
// Add the order and its contents to working
// memory
    engine.add(order);
    engine.add(order.getCustomer());
    engine.addAll(order.getItems());
// Fire the rules that apply to this order
    engine.run();

// Return the list of offers created by the rules
    return engine.getObjects(new Filter.ByClass(Offer.class));
}}
```

JESS & Java (Example)

- Using Jess Rules to reply ACL Messages
  - Use Jess engine in an agent's *behavior*



JESS & Java (Example) - JessBehaviour

```
public class BasicJessBehaviour extends
CyclicBehaviour {

        Rete jess;

        // Constructor

        BasicJessBehaviour(Agent agent) throws
JessException {

        // create a Jess rule engine

        jess = new Rete();

// create a fact with the agent's name: (i-am X)

        jess.eval(" (deffacts Me " +

        " (i-am " + myAgent.getName() + ") )");

// define Userfunction "send" to send

// ACLMessages

jess.addUserfunction(new JessSend(myAgent));

// load rules and functions into working memory

jess.batch("JadeAgent.clp");

jess.reset();

}
```

JESS & Java (Example) – ACLMessage receive

```
// Action method
public void action() {
    // Receive a message
    MessageTemplate mt = … // some template
    ACLMessage msg = myAgent.receive(mt);
    if (msg != null) {
        try {
// Convert ACLMessage into Jess fact and assert
// it
    jess.assertString(ACL2JessString(msg));
    // run Jess engine
    jess.run();
} catch (JessException je) { … }
} else block();
}
```

JESS & Java (Example) – *JessSend*

```
public class JessSend implements Userfunction {
    Agent myAgent;
    public JessSend(Agent a) { myAgent = a; }
    public String getName() { return "send"; }

// JESS calls (send ?m) where ?m is an
// ACLMessage Jess fact
    public Value call(ValueVector vv, Context
ctx) throws JessException {
```

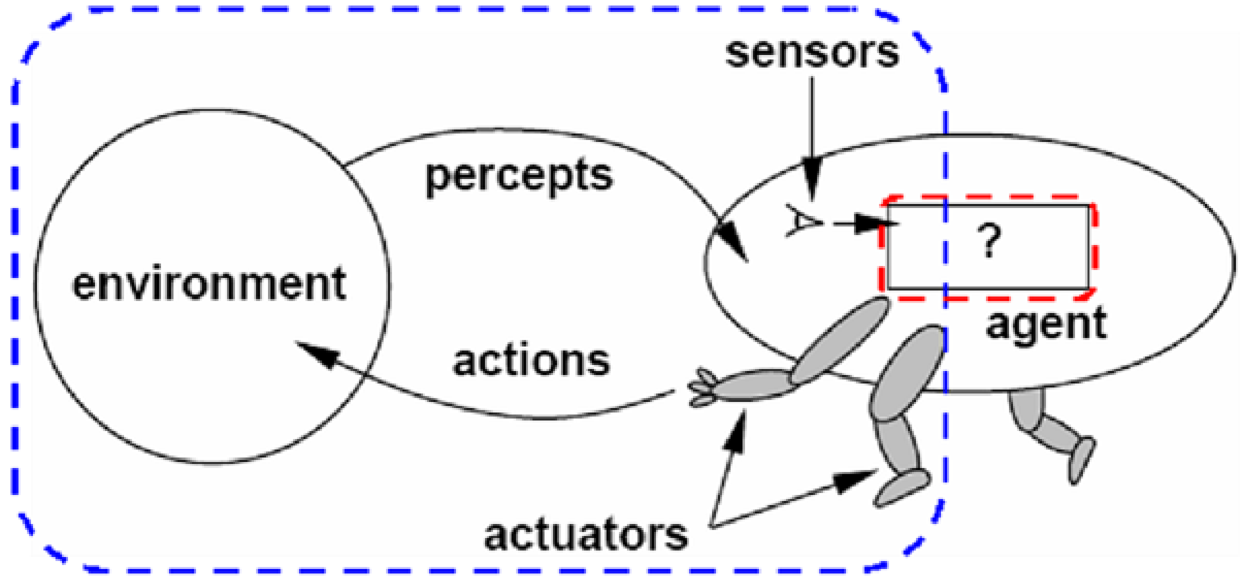
```
// Get the Fact
Fact f = vv.get(1).factValue(ctx);
// Convert fact into ACLMessage
ACLMessage msg = JessFact2ACL(ctx, f);
// Send the ACLMessage
myAgent.send(msg);
return Funcall.TRUE;
}
}
```

JESS & Java (Example) – JadeAgent.clp

```
; (limited) template of an ACLMessage
(deftemplate ACLMessage
    (slot communicative-act)
(slot sender) (multislot receiver) (slot content))
; rule for handling CFP
(defrule proposal
?m <- (ACLMessage (communicative-act CFP)
        (sender ?s) (content ?c) (receiver ?r))
(i-am ?r)
=>
(bind ?p (gen-proposal ?c))
(assert (ACLMessage (communicative-act PROPOSE)
    (sender ?r) (receiver ?s) (content ?p)) )
(retract ?m) )
; rule for sending a message
(defrule send-a-message
    ?m <- (ACLMessage (sender ?s))
    (i-am ?s)
=>
    (send ?m)
    (retract ?m) )
```

JESS & Java – Integration

JESS used to implement the reasoning module of a JADE agent



JADE provides environment and facilitates the sending / receiving of messages

JESS enables the implementation of the agent's decision module in a declarative way

- JESS can be used in one of the many behaviors of an agent

JESS
Java Expert System Shell
Framework

Integrated Master's in Informatics Engineering

Intelligent Agents

2018/2019

Synthetic Intelligence Lab

Filipe Gonçalves

César Analide