

JessTab & JessAgentTab Tutorial

Henrik Eriksson
Karl-Heinz Krempels

Outline

1. Introduction
2. Background
3. Installation
4. Interaction with JessTab
5. Introduction to Jess programming
 - Jess functions
 - Rule-based reasoning with Jess
6. Managing Protégé ontologies with Jess
7. Mapping Protégé ontologies to Jess
8. Metalevel mappings
9. JessTab and Protégé OWL
10. Example
11. JessTab Conclusion
12. JessAgentTab part (Karl-Heinz Krempels)

Tip: JessTab slides available at
<http://www.ida.liu.se/~her/JessTab/tutorial07/>

2. Background

Ontologies are nice to look at, but...
 ...they do not do anything.

Background (cont.)

- Protégé-related problems
 - Difficult to directly integrate problem solving and ontology development in Protégé
 - ⇒ Languages/shells need direct access to Protégé
 - Difficult manage large/complex ontologies
 - ⇒ Ontology editors should be programmable
- Protégé allows *alternative* problem-solving engines through the plug-in API
 - The Java API allows access to the internal ontology representation

Why Jess and JessTab?

- Jess
 - Popular language/shell
 - Active user community
 - Implemented in Java
- JessTab
 - A Protégé plug-in for running Jess under Protégé
 - Combines the strengths of Protégé and Jess

Practical uses of JessTab

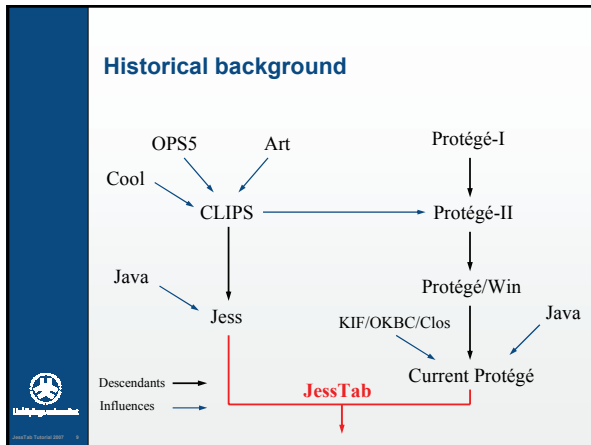
- Macro language
 - Creating lots of classes quickly
 - Making large changes to ontologies
- Rule engine
 - Information retrieval
 - Classification
 - Decision support
 - Planning

Examples of applications

- Ontology engineering and reengineering
 - Jess as macro/scripting for ontologies
- Importing ontologies
 - Jess as input filter
- Semantic Web
- Problem-solving methods
- Agent frameworks
 - Jade/JessProtege
 - JessAgentTab
- Classic expert-system development

What is Jess?

- Java Expert System Shell; based on CLIPS
- Forward chaining; production rules
- Fact-base and pattern matching
- Lisp-like syntax
- No support for object orientation
 - The Cool subsystem of CLIPS not implemented
- Developed by Sandia Laboratories
 - <http://herzberg.ca.sandia.gov/jess/>



Tool integration – Two possibilities

- Loose integration
 - No changes to each representation model
 - Translators between formats
 - Independent software
- Tight integration
 - Changes to representation models when needed
 - Integrated software (e.g., same Java VM)
 - Unified user interface

JessTab supports tight integration

Approach – JessTab plug-in for Protégé

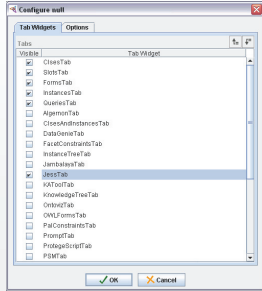
- Jess console window in Protégé
- Mapping instances to Jess facts
- Functions for knowledge-base operations
- Mirroring Jess definitions in Protégé knowledge bases
- Support for metalevel objects
- Support for methods and message handlers

3. Installation

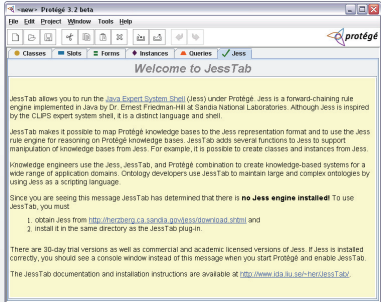
- JessTab is bundled with the Protégé distribution
- Latest JessTab version available from SourceForge
- It is necessary to download and install Jess separately
 - Because of licensing

Enabling JessTab

- Enable the tab



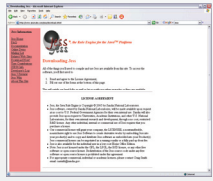
JessTab with no Jess engine



Jess installation

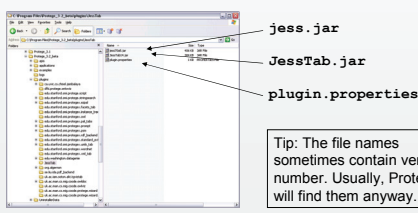
- Visit <http://www.jessrules.com/>
- Choose license type
 - Trial – expires after 30 days
 - Licensed – commercial or academic (includes source)
- Choose version
 - Stable (e.g., 6.1p8)
 - Development (e.g., 7.0b7)

Tip: Development versions of Jess are usually stable



Jess installation (cont.)

- The distribution contains the file `jess.jar`
- Put the file `jess.jar` in the `Protégé/plugins/JessTab` directory in the Protégé installation



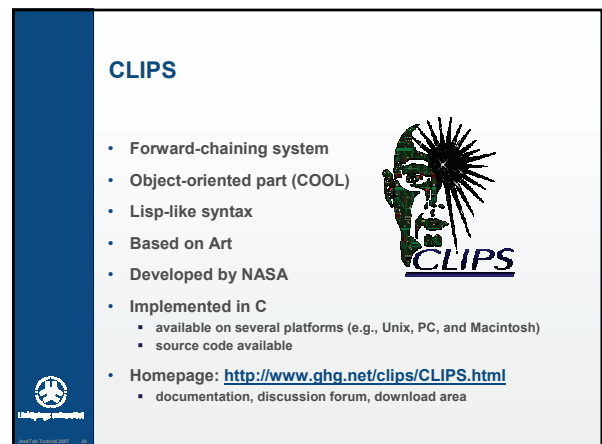
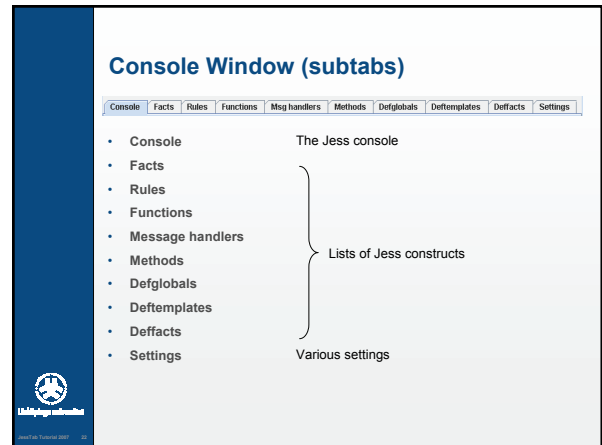
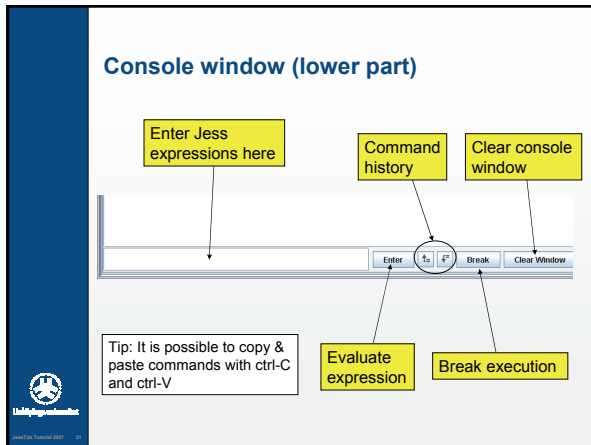
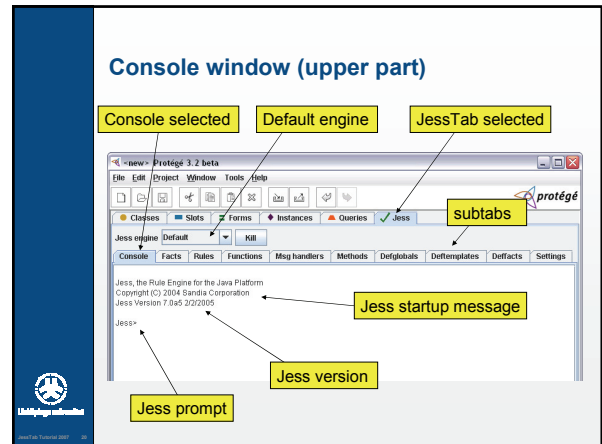
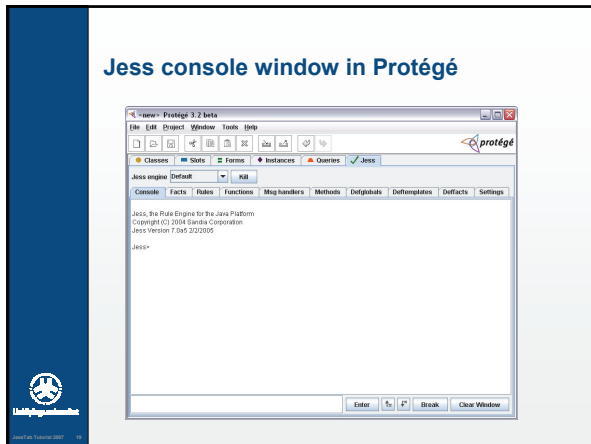
Tip: The file names sometimes contain version number. Usually, Protégé will find them anyway.

Installation troubleshooting tips

- Make sure everything is in the correct directory
 - The files `jess.jar`, `JessTab.jar`, and `plugin.properties` should be in the `plugins/JessTab` directory
- Two or more versions of the same `.jar` file on the CLASSPATH means **trouble**. Double check this!
- Check Jess version number in startup message
- Check the `plugin.properties` file
- Try running Jess without Protégé/JessTab
- If all else fails, try a clean installation of Protégé

4. Interaction with JessTab


- Ready to go
 - Jess installation completed
 - JessTab enabled
- Interaction based on read-evaluate-print loop
 - Just like Lisp



Jess



- Based on CLIPS
 - Rules from CLIPS but no object-oriented part
- Developed by Dr. Ernest Friedman-Hill at Sandia National Laboratories
- Implemented in Java
 - source code license available
- Homepage: <http://www.jessrules.com/>
 - documentation, download area




Running Jess

```

> jess ↵
Jess, the Rule Engine for the Java Platform
Copyright (C) 2004 Sandia Corporation
Jess Version 7.0a5 2/2/2005

Jess> exit ↵
exit
Jess> (+ 3 4) ↵
7
Jess> (exit) ↵
>
    
```

Exits Protégé if running in JessTab




Jess Facts

Jess manages a list of known *facts*

Sample Jess fact:

```

(person (hair-color black)
        (name "John Smith")
        (eye-color blue)
        (age 23))
    
```



The Deftemplate Construct

Declaration of valid slots for a given relation name

The general format of *deftemplate* is


```

(deftemplate <relation-name> [<optional-comment>]
  <slot-definition>*)
    
```

where <slot-definition> is defined as

```

(slot <slot-name>) | (multislot <slot-name>)
    
```




Sample Deftemplate

Information about a person


```

(deftemplate person "A sample deftemplate"
  (slot name)
  (slot age)
  (slot eye-color)
  (slot hair-color))
    
```



Multifield slots


- Normal, single-field slots can hold only one value
- Multifield* slots can hold several values
- The values are ordered in a list



Sample Fact Revisited

```
(person (name John Smith)
        (age 23)
        (eye-color blue)
        (hair-color brown))
```

- Illegal if *name* is a single-field slot, but
- legal if *name* is a multifield slot




JessTab Tutorial 2007 31

Ordered Facts

- Implied deffemplate: Single implied multifield slot
- List of elements
- Order of elements important; slot names not required

- Example: list of numbers

```
(number-list 7 9 3 4 20)
```




JessTab Tutorial 2007 32

Adding and Removing Facts

- All facts known to Jess are stored in the fact list
- Add new facts with the assert command:



```
(assert <fact>+)
```
- More than one fact can be added with assert



JessTab Tutorial 2007 33

Adding Facts: Example

```
Jess>
(deftemplate person
  (slot name)
  (slot age)
  (slot eye-color)
  (slot hair-color))
Jess>
(assert (person (name "John Q. Public")
               (age 23)
               (eye-color blue)
               (hair-color black)))
<Fact-0>
Jess>
```




JessTab Tutorial 2007 34

Displaying Facts

- The *facts* command: (facts)
- Example:

```
Jess> (facts)
f-0 (MAIN::person (name "John Smith") (age 23)
      (eye-color blue) (hair-color black))
For a total of 1 fact.
Jess>
```




JessTab Tutorial 2007 35

Adding another Fact

```
Jess>
(assert (person (name "Jane Smith")
               (age 36)
               (eye-color green)
               (hair-color red)))
<Fact-1>
Jess> (facts)
f-0 (person (name "John Smith") (age 23)
      (eye-color blue) (hair-color black))
f-1 (person (name "Jane Smith") (age 36)
      (eye-color green) (hair-color red))
For a total of 2 facts.
Jess>
```

Normally, Jess does not accept duplicate fact entries



JessTab Tutorial 2007 36

Jess Rules

Sample rule:

```
IF the emergency is a fire
THEN the response is to activate the sprinkler system
```

Step 1 — Define the relevant *deftemplates*:

```
(deftemplate emergency (slot type))
(deftemplate response (slot action))
```

Jess Rules (cont.)

Step 2 — Define the rule

```
(defrule fire-emergency "A sample rule"
(emergency (type fire))
=>
(assert (response
(action activate-sprinkler-system))))
```

Labels: rule header, patterns, actions

The Run Command

Run the forward-chaining system

```
Jess> (run) ↵
Jess> (facts) ↵
f-0 (emergency (type fire))
f-1 (response (action activate-sprinkler-system))
For a total of 2 facts in module MAIN.
Jess>
```

Printing the Result

```
(defrule fire-emergency
(emergency (type fire))
=>
(printout t "Activate the sprinkler system"
crLf))
```

Label: output stream (t = stdout)

Multiple Rules

```
(defrule fire-emergency
(emergency (type fire))
=>
(printout t "Activate the sprinkler system"
crLf))

(defrule flood-emergency
(emergency (type flood))
=>
(printout t "Shut down electrical equipment"
crLf))
```

Single-Field Patterns

```
(deftemplate person
(multislot name)
(slot social-security-number))

(deffacts some-people
(person (name John Smith)
(social-security-number 483-98-9083))
(person (name Jack Smith)
(social-security-number 483-98-9084)))

(defrule print-social-security-numbers
(print-ss-numbers-for ?last-name)
(person (name ?first-name ?middle-name ?last-name)
(social-security-number ?ss-number))
=>
(printout t ?ss-number crLf))
```

Single-Field Wildcards

```
(defrule print-social-security-numbers
  (print-ss-numbers-for ?last-name)
  (person (name ? ? ?last-name)
          (social-security-number ?ss-number))
  =>
  (printout t ?ss-number crlf))
```

Conditional Patterns

```
(defrule black-or-brown-hair
  (person (name ?name) (hair brown | black))
  =>
  (printout t ?name " has dark hair" crlf))

(defrule black-or-brown-hair
  (person (name ?name) (hair ?color&brown |
                           black))
  =>
  (printout t ?name " has " ?color " hair"
            crlf))
```

Conditional Patterns (cont.)

```
(defrule not-black-or-brown-hair
  (person (name ?name)
          (hair ?color&-brown&-black))
  =>
  (printout t ?name " has " ?color " hair" crlf))
```

Conditional Patterns (cont.)

```
(defrule complex-eye-hair-match
  (person (name ?name1)
          (eyes ?eyes1&blue | green)
          (hair ?hair1&-black))
  (person (name ?name2&-?name1)
          (eyes ?eyes1&-?eyes2)
          (hair ?hair2&-red | ?hair1))
  =>
  (printout t ?name1 " has " ?eyes1 " eyes and " ?hair1 " hair"
            crlf)
  (printout t ?name2 " has " ?eyes2 " eyes and " ?hair2 " hair"
            crlf))
```

Conditional Patterns (cont.)

```
(defrule no-identical-birthdays
  (not (and (person (name ?name)
                   (birthday ?date))
            (person (name ~?name)
                   (birthday ?date))))
  =>
  (printout t "No two people have the same birthday"
            crlf))
```

Loading Jess Constructs from a File

- Create the Jess source file with a text editor
- Use the *batch* command:


```
(batch <file>)
```
- Example:


```
Jess> (batch "fire.clp") ↵
      Defining deftemplate emergency
      Defining deftemplate response
      Defining defrule: fire-emergency +j
```


6. Managing Protégé ontologies with Jess

- The Protégé GUI is nice but sometimes scripting or programmatically modifying an ontology is better
- JessTab adds several functions and constructs for managing ontologies
- Naturally, these functions and constructs are available from the Jess command prompt and from Jess programs

Defining classes and instantiating them

```
Jess> (defclass Person (is-a :THING)
(slot name (type string))
(slot age (type integer))) ↵
TRUE
Jess> (make-instance john of Person (name "John")
(age 20)) ↵
<External-Address:SimpleInstance>
Jess>
```

Modifying slots

```
Jess> (slot-set john age 21) ↵
Jess>
Jess> (slot-get john age) ↵
21
Jess>
```

Creating a second instance

```
Jess> (make-instance sue of Person (name "Sue")
(age 22)) ↵
<External-Address:SimpleInstance>
Jess>
Jess> (slot-get sue age) ↵
22
Jess>
```

Functions for knowledge-base operations


mapclass	slot-range	instance
mapinstance	slot-allowed-values	instance-existp
ummapinstance	slot-allowed-classes	instance-name
defclass	slot-allowed-parents	instance-address
make-instance	slot-documentation	instance-addressp
initialize-instance	slot-sources	instance-namep
modify-instance	facet-get	slot-existp
duplicate-instance	facet-set	slot-default-value
definstances	class	set-lb-save
ummake-instance	class-existp	get-lb-save
slot-get	class-abstractp	load-kb-definitions
slot-set	class-reactivep	load-project
slot-replace\$	superclass	include-project
slot-insert\$	subclass	save-project
slot-delete\$	class-superclasses	jesstab-version-number
slot-facets	class-subclasses	jesstab-version-string
slot-types	get-defclass-list	get-knowledge-base
slot-cardinality	class-slots	get-tabs

7. Mapping Protégé ontologies to Jess

- Transferring the Protégé representation to the Jess representation

Mapping classes

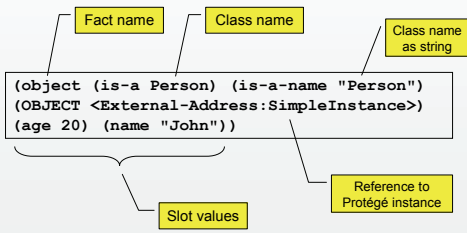
```
Jess> (defclass Person (is-a :THING)
  (slot name (type string))
  (slot age (type integer))) ↵
TRUE
Jess> (make-instance john of Person (name "John")
  (age 20)) ↵
<External-Address:SimpleInstance>
Jess> (mapclass Person) ↵
Person
Jess> (facts) ↵
f-0 (object (is-a Person) (is-a-name "Person")
  (OBJECT <External-Address:SimpleInstance>)
  (age 20) (name "John"))
For a total of 1 facts.
```



The "object" fact

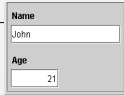
- Pattern for facts corresponding to Protégé instances

```
(object (is-a Person) (is-a-name "Person")
  (OBJECT <External-Address:SimpleInstance>)
  (age 20) (name "John"))
```



Modifying slots

```
Jess> (slot-set john age 21) ↵
Jess> (facts) ↵
f-1 (object (is-a Person) (is-a-name "Person")
  (OBJECT <External-Address:SimpleInstance>)
  (age 21) (name "John"))
For a total of 1 facts.
```



Creating Instances

```
Jess> (make-instance sue of Person (name "Sue") (age 22)) ↵
<External-Address:SimpleInstance>
Jess> (facts) ↵
f-1 (object (is-a Person) (is-a-name "Person")
  (OBJECT <External-Address:SimpleInstance>)
  (age 21) (name "John"))
f-4 (object (is-a Person) (is-a-name "Person")
  (OBJECT <External-Address:SimpleInstance>)
  (age 22) (name "Sue"))
For a total of 2 facts.
```

Adding a Jess rule

```
Jess> (defrule twenty_one
  (object (is-a Person)
    (name ?n) (age ?a&:(>= ?a 21)))
=>
  (printout t "The person " ?n
    " is 21 or older" crlf) ↵
TRUE
Jess> (run) ↵
The person John is 21 or older
The person Sue is 21 or older
2
Jess>
```

More rule examples

```
(defrule twenty_two
  (object (name ?n)
    (age 22))
=>
  (printout t "The object " ?n " is 22" crlf))

(defrule print_age
  (object (is-a Person)
    (name ?n)
    (age ?a))
=>
  (printout t "The person " ?n " is " ?a crlf))
```

Accessing instances from rules

```
(defrule print_age_2
  (object (is-a Person)
    (OBJECT ?obj)
    (name ?n))
=>
  (printout t "The person " ?n
    " is " (slot-get ?obj age) crlf))

(defrule print_age_3
  ?f <- (object (is-a Person)
    (name ?n))
=>
  (printout t "The person " ?n
    " is " (slot-get ?f age) crlf))
```

Modifying slot values from rules

```
(defrule upgrade_my_age
  ?f <- (object (is-a Person)
    (name "Joe Hacker")
    (age 20))
=>
  (slot-set ?f age 21))

(defrule upgrade_my_age_2
  ?f <- (object (is-a Person)
    (name "Joe Hacker")
    (age ?a))
=>
  (slot-set ?f age (+ ?a 1)))
```

Tip: Avoid.
Infinite loop!

Consistency rules

```
(defrule set_john_doe_if_no_name
  ?f <- (object (is-a Person)
    (name nil)
    (age ?a&:(numberp ?a)))
=>
  (slot-set ?f name "John Doe")
)

(defrule remove_if_no_name_and_no_age
  ?f <- (object (is-a Person)
    (name nil)
    (age nil))
=>
  (unmake-instance ?f)
)
```

Matching instances in rules

- Remove duplicates

```
(defrule remove_if_duplicate_name_and_age
  (object (is-a Person)
    (OBJECT ?p)
    (name ?n)
    (age ?a))
  (object (is-a Person)
    (OBJECT ~?p)
    (name ?n)
    (age ?a))
=>
  (unmake-instance ?p)
)
```

Matching instances in rules (cont.)

- Find pet owners

```
(defrule print_owner_and_pet
  (object (is-a Person)
    (OBJECT ?p)
    (name ?n))
  (object (is-a Dog|Cat|Bird)
    (is-a-name ?type)
    (owner ?p)
    (name ?an))
=>
  (printout t "The person " ?n " has a " ?type
    " called " ?an crlf))
```

Mappings and rules – Summary

- Use (mapclass <class-name>) or (mapinstance <instance>) to map instances to facts
- Use (facts) to check what facts you got
- Define the rules matching the object facts (defrule ...)
- Use (run) to invoke the rules
- Tip #1: Learn as much Jess as possible (e.g., rule patterns)
- Tip #2: Start out with simple examples (when learning and troubleshooting)

Question #1: Why mappings?

- Separates the Protégé model from the Jess model
- Allows selected parts of the Protégé kb to be visible to Jess
- Allows for large ontologies and/or fact bases with a small common part
- Different Jess engines can map in different part of a Protégé kb
- Why not have Jess index the Protégé frames directly?
 - Implementation complexity
 - Maintenance problem: The Jess indexing scheme changes often (due to code optimizations)
 - Several Jess versions supported simultaneously
 - However, mappings increase memory consumption

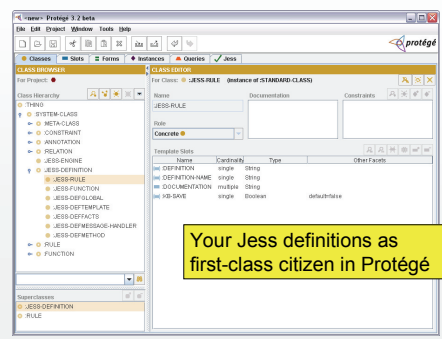
Question #2: Why does mapclass map to these “object” facts?

- Backward compatibility with CLIPS rules
 - The Jess rules have (almost) the same syntax as the CLIPS rules for objects
- Support for more general patterns in rules
 - Example: Find an instance of any class with a certain slot value
- No name conflicts with other types of facts
 - The name `object` is reserved for facts coming from Protégé
- Why can't I change the mappings to something else?
 - In fact, there is an API for this
 - Using Java, you can write arbitrary mappings

Mirroring Jess definitions in Protégé knowledge bases

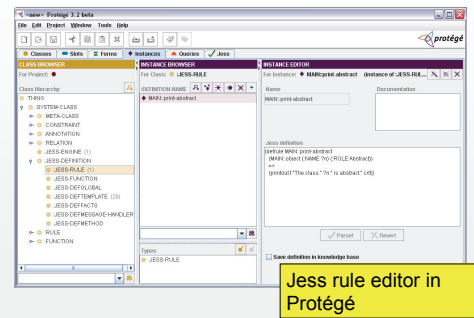
- Reverse mapping
- Jess definitions become instances in the Protégé kb
- Graphical browsing of definition instances in Protégé
- Definition instances visible to other tabs
- Introspection because JessTab can access the definition instances
- Limited support for editing and customization of definition editors in Protégé
- Limited support for saving these Jess definitions with the Protégé kb
 - Not recommended unless you know its limitations

Mirroring Jess definitions in Protégé knowledge bases (cont.)



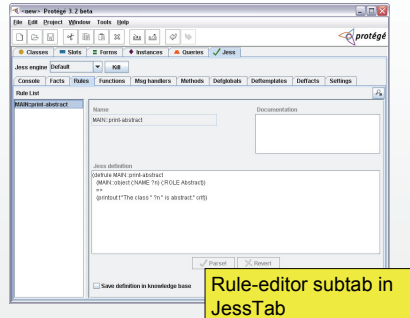
Your Jess definitions as first-class citizen in Protégé

Editing Jess definitions in Protégé



Jess rule editor in Protégé

Editing Jess definitions in Protégé (cont.)



Rule-editor subtab in JessTab

8. Metalevel mappings

- The Next Level!
- Mapping classes to facts
- Good news: Classes are instances!

```

graph TD
    Instance[Instance] -- instance of --> Class[Class]
    Class -- instance of --> Metaclass[Metaclass]
    
```

Support for Protégé metalevel objects

- JessTab support for metaclasses, metaslots, and metafacts
- Functions for instances work for classes too
 - and for slots and facets
- Defining classes by instantiating metaclasses:

```
(make-instance Person of :STANDARD-CLASS (:DIRECT-SUPERCLASSES :THING))
```

Class definition by instantiation

```
(make-instance of :STANDARD-CLASS
  (:NAME LivingThing)
  (:ROLE Abstract)
  (:DIRECT-SUPERCLASSES :THING))

```

(defclass LivingThing)

```
(make-instance of :STANDARD-CLASS
  (:NAME Person)
  (:DIRECT-SUPERCLASSES LivingThing)
  (:DIRECT-TEMPLATE-SLOTS
    (make-instance of :STANDARD-SLOT
      (:NAME name)
      (:SLOT-VALUE-TYPE String))
    (make-instance of :STANDARD-SLOT
      (:NAME age)
      (:SLOT-VALUE-TYPE Integer))))

```

(defclass Person)

Class definition by instantiation (cont.)

- Resulting classes: LivingThing and Person

Tip: Classes can also be created programmatically with calls to defclass (which is a construct implemented as a function)

Class definition with custom metaclass

```
(make-instance of :STANDARD-CLASS
  (:NAME MyMetaClass)
  (:DIRECT-SUPERCLASSES :STANDARD-CLASS)
  (:DIRECT-TEMPLATE-SLOTS
    (make-instance of :STANDARD-SLOT
      (:NAME AUTHOR)
      (:SLOT-VALUE-TYPE String))))

```

```
(make-instance of MyMetaClass
  (:NAME Person2)
  (:DIRECT-SUPERCLASSES Person)
  (:DIRECT-TEMPLATE-SLOTS
    (make-instance of :STANDARD-SLOT
      (:NAME income)
      (:SLOT-VALUE-TYPE Integer))))

```

; Set the AUTHOR value for Person2
(slot-set Person2 AUTHOR "The meta man")

Add AUTHOR property to classes

Resulting class with AUTHOR property

(instance of MyMetaClass)

Class changes

- Change the metaclass of an existing class


```
(slot-set Person :DIRECT-TYPE MyMetaClass)
```
- BTW, you can change the class of an existing ordinary instance


```
(slot-set john :DIRECT-TYPE Person2)
```

Rules and metalevel objects

- Use `mapclass` on metaclasses
 - Maps classes (as instances) to facts
 - Check result with `(facts)`
- Define rules matching the facts representing the classes
 - Rules for searching ontologies and identifying patterns
 - Rules for modifying ontologies
- Useful for ontology development and maintenance
 - Rules for ontology-wide changes
 - Rules for identifying inconsistencies

Printing abstract classes in Protégé

```
(mapclass :THING)
(defrule print-all-abstract-classes
  ?c <- (object )
  (test (class-abstractp ?c))
  =>
  (printout t "The class "
    (instance-name ?c)
    " is abstract." crlf))
```

Map every class to Jess

Match every object

Test for abstract classes

Print matches

Modifying ontologies

Change the role to *abstract* for classes that have subclasses, but do not have any instances:

```
(defrule make-classes-abstract
  ?c <- (object (:NAME ?n)
    (:ROLE Concrete)
    (:DIRECT-INSTANCES ))
  (not (object (:NAME ?n) (:DIRECT-SUBCLASSES)))
  =>
  (slot-set ?c :ROLE Abstract))
```

Concrete classes & no instances

Change role

Not no subclasses = subclasses exist

Methods and Message Handlers

- Object-oriented programming for Protégé/Jess
 - Complements rule-based modeling
- Methods
 - Respond to generic function calls
 - Match on parameter types
- Message handlers
 - Handle messages sent to objects
 - Allow parameters, but not pattern matching
 - Before, after, and around handlers
- Implementation of methods and message-handlers supported by CLIPS

Defining methods

- Syntax


```
(defmethod <name> [<index>] [<comment>]
  [<parameter-restriction>* [<wildcard-parameter-restriction>])
  <action>* )
```
- Examples


```
(defmethod add ((?a STRING) (?b STRING))
  (str-cat ?a ?b))

(defmethod add ((?a MyClass) (?b MyClass))
  ...)
```

Defining message handlers

- Syntax

```
(defmessage-handler <class-name> <message-name>
  [<handler-type>] [<comment>]
  [<parameter>* [<wildcard-parameter>]]
  <action>* )
```

- Examples

```
(defmessage-handler MyClass get-foo ()
  ?self:foo)

(defmessage-handler rectangle find-area ()
  (* ?self:side-a ?self:side-b))
```

Invoking message handlers

- Sending messages
- Syntax: (send <instance> <message> <param>*)
- Example

```
Jess> (defclass MyClass (is-a :THING)
      (slot foo (type integer))) ↵
TRUE
Jess> (make-instance x of MyClass (foo 42)) ↵
<External-Address:DefaultSimpleInstance>
Jess> (slot-get x foo) ↵
42
Jess> (defmessage-handler MyClass get-foo () ?self:foo) ↵
TRUE
Jess> (send x get-foo) ↵
42
Jess>
```

Methods and Message Handlers – Summary

- Object-oriented modeling
- Methods
 - Advanced parameter matching (similar to CLOS)
 - Both classes and datatypes in parameter patterns
 - Overloading of existing functions
- Message handlers
 - Message passing
 - Invoked with (send <inst> <msg> <param>*)
 - Easy access to slot values (through ?self)
 - Before, after, and around methods

9. JessTab and Protégé OWL

- Basic support for OWL
- JessTab uses the Protégé frames API
 - which provides a best-effort implementation of OWL functionality
- Metaclasses not supported
- OWL constraints/expressions not supported
- Jess definition mirroring not supported

10. Example

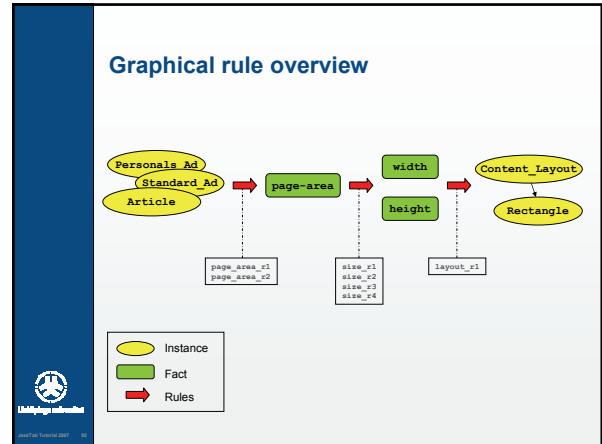
- Based on the Protégé newspaper example
- Goal: Layout rules in Jess
 - Reuse ontology
 - Add new rules
- Step-wise development
- Download from <http://www.ida.liu.se/~her/JessTab/tutorial07/>

Steps

- Prerequisite
 - Jess and JessTab installed
- Open newspaper example
 - Available in the standard Protégé installation
- Enable JessTab
 - Project -> Configure -> select JessTab
- Cut-and-paste from source file (newspaper.jess)
- Alternatively, load with (batch "newspaper.jess")
 - Tip: You probably need the full pathname for this file

Rule overview

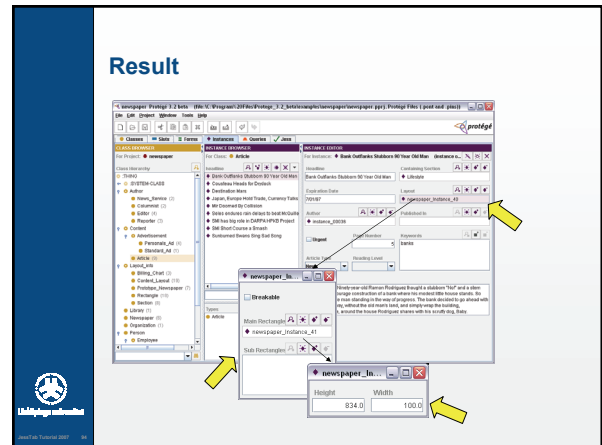
- Mapclass for newspaper articles and ads
 - (mapclass Content)
- Rules for mapping instance facts to page-area facts
 - page_area_r1, page_area_r2
- Rules for creating size facts from the page-area facts
 - size_r1, size_r2, size_r3, size_r4
- Rules for creating layout instances from the size facts
 - layout_r1



Rules – Code

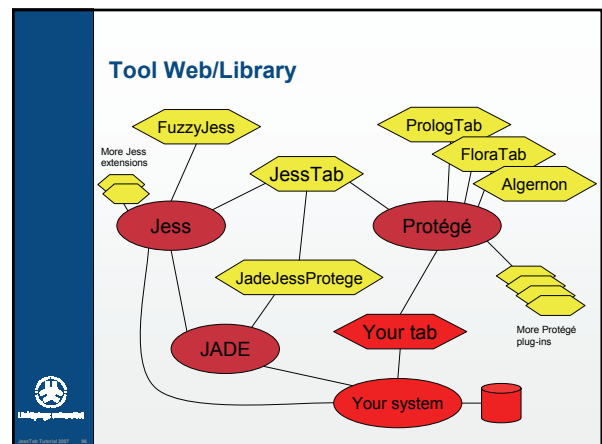
```

(mapclass Content)
(defrule area_r1
  (object (is-a Personals_Ad|Standard_Ad)
   (OBJECT ?a)
   (name ?n)
   (page_number ?p))
  =>
  (assert (page-area ?p Ad ?n ** ?a))
)
(defrule area_r2
  (object (is-a Article)
   (OBJECT ?a)
   (headline ?h)
   (text ?t)
   (page_number ?p))
  =>
  (assert (page-area ?p Article ?h ?t ?a))
)
(defrule size_r1
  (page-area ?p (< ?p 10) Article ? ? ?a)
  =>
  (assert (width ?a 100.0))
)
(defrule size_r2
  (page-area ?p (>= ?p 10) Article ? ? ?a)
  =>
  (assert (width ?a 50.0))
)
(defrule size_r3
  (page-area ? Article ? ? ?a)
  =>
  (assert (height ?a (* (str-length ?t) 2.0)))
)
(defrule size_r4
  (page-area ?p Ad ? ? ?a)
  =>
  (assert (width ?a 50.0) (height ?a 50.0))
)
(defrule layout_r1
  (width ?a ?w) (height ?a ?h)
  =>
  (slot-set ?a layout
   (make-instance of Content_Layout
    (main_rectangle (make-instance of Rectangle
     (width ?w) (height ?h))))
  )
)
    
```



11. Conclusion

- Web of tools
- Future work
- Trying JessTab
- Learning more
- Summary
- Questions



Ideas for future work

- Improved OWL support
- Custom mappings (defined in Jess)
- Support for managing Protégé forms
- Improved GUI
- Aspect-oriented functionality (e.g., pointcut for message-handlers)
- Find grained access to facet information
- ???



JessTab Tutorial 2007 97

Trying JessTab

- Obtain Protégé
 - Download from <http://protege.stanford.edu/>
 - License: MPL 1.1
- Obtain Jess
 - Download from <http://www.jessrules.com/>
 - License: Special Jess license (commercial or free academic)
 - Compilation sometimes required
- Get JessTab
 - Download from <http://www.ida.liu.se/~her/JessTab/>
 - License: MPL 1.1



JessTab Tutorial 2007 98

Learning more about Jess and JessTab

- Jess manual
 - See <http://www.jessrules.com/>
- Jess book
 - Ernest Friedman-Hill. *Jess in Action: Java Rule-based Systems*. Manning Press, 2003. ISBN: 1930110898
 - See <http://manning.com/friedman-hill/>
- Jess mailing list
 - See http://www.jessrules.com/jess/mailling_list.shtml
- JessTab manual
 - See <http://www.ida.liu.se/~her/JessTab/>
- Jess publication
 - Henrik Eriksson. *Using JessTab to integrate Protégé and Jess*. *IEEE Intelligent Systems*, 18(2):43–50, 2003.



JessTab Tutorial 2007 99

Summary

- JessTab: Protégé – Jess integration
- Manage Protégé ontologies and knowledge bases from Jess
- Rule-based reasoning in Protégé
- Protégé as graphical, object-oriented extension to Jess



JessTab Tutorial 2007 100