UNIVERSITE DE TECHNOLOGIE DE COMPIÈGNE
Département de Génie Informatique

MOSS 7 - User Interface (Windows)

Jean-Paul Barthès

BP 349 COMPIÈGNE
Tel +33 3 44 23 44 66
Fax +33 3 44 23 44 77

Email: barthes@utc.fr

N218
July 2008

Warning

This document describes the Windows user interface to the **MOSS v7.xx** programming environment that uses the PDM4 model for representing knowledge. It includes a description of the browser but not of the MOSS Editor. It is an upgrade of the first part of the previous document:

`N183 -MOSS 4-Editor-PC`

A current research version of MOSS 7 runs in a MacIntosh Common Lisp environment (MCL 5.1 or 5.2 for OSX) and in an Allegro Common Lisp environment (ACL 8.1 running under Windows XP).

Keywords

Object representation, object-oriented programming environment, ontology formalism, knowledge base, semantic queries.

Revisions

Version	Date	Author	Remarks
1.0	Jul 08	Barthès	Initial issue

MOSS documents

Related documents

- UTC/GI/DI/N196L - PDM4
- UTC/GI/DI/N206L - MOSS 7 : User Interface (Macintosh)
- UTC/GI/DI/N218L - MOSS 7 : User Interface (Windows)
- UTC/GI/DI/N219L - MOSS 7 : Primer
- UTC/GI/DI/N220L - MOSS 7 : Syntax
- UTC/GI/DI/N221L - MOSS 7 : Advanced Programming
- UTC/GI/DI/N222L - MOSS 7 : Query System
- UTC/GI/DI/N223L - MOSS 7 : Kernel Methods
- UTC/GI/DI/N224L - MOSS 7 : Low Level Functions
- UTC/GI/DI/N225L - MOSS 7 : Dialogs
- UTC/GI/DI/N228L - MOSS 7 : Paths to Queries

Readers unfamiliar with MOSS should read first N196 and N219 (Primer), then N220 (Syntax), N218 (User Interface). N223 (Kernel) gives a list of available methods of general use when programming. N222 (Query) presents the query system and gives its syntax. For advanced programming N224 (Low level Functions) describes some useful MOSS functions. N209 (Dialogs) describes the natural language dialog mechanism.

Contents

Warning MOSS has a *versioning mechanism*, i.e. all objects can have different versions in the same environment. The current document is intended for beginning users who want to explore a particular knowledge base that is assumed to be in *version 0* (also referred to as *context 0*).

1 Introduction

After having launched the Lisp environment, MOSS can be started as a stand alone application by loading the moss-load.fasl file (Fig.1).

Figure 1: File for loading MOSS as a stand alone application: moss-load.fasl

When started as a stand alone application, MOSS opens with an interface window shown Fig.2. Throughout the document we refer to this user's interface as the **MOSS window**.

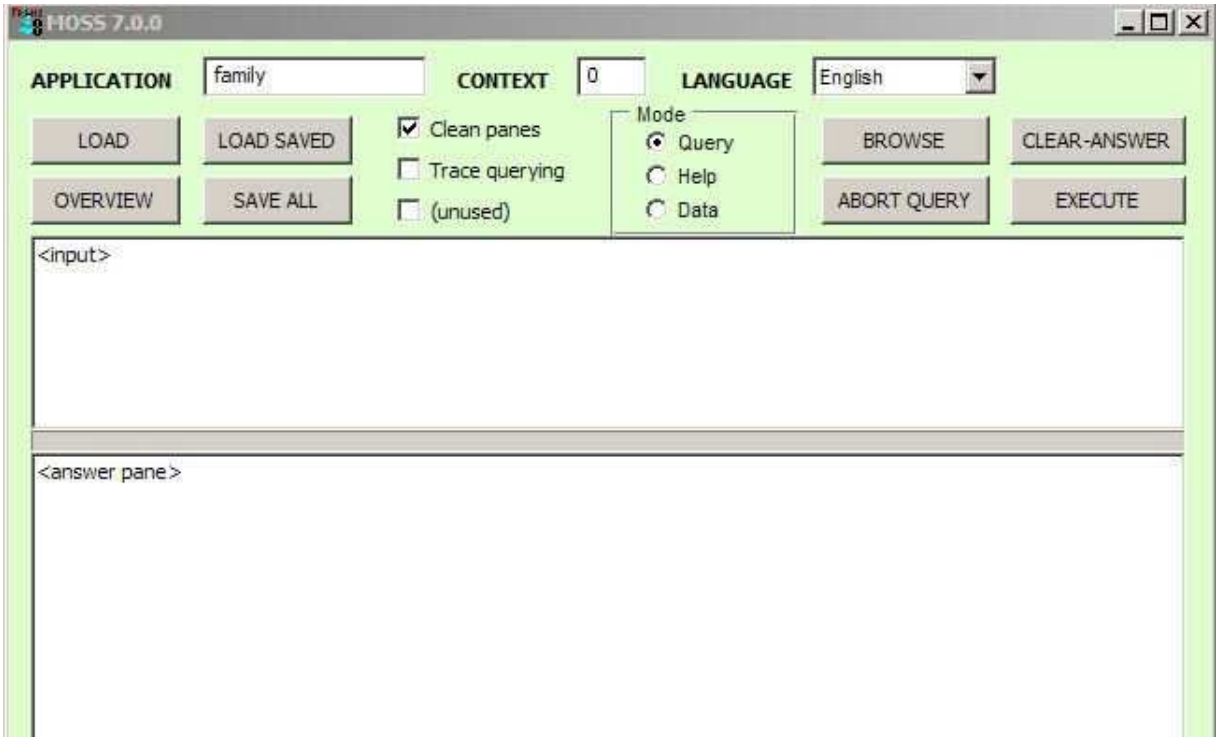

Figure 2: User Interface - MOSS window

In addition to the MOSS window, other more specialized windows can be activated for locating objects, editing objects, or browsing through the knowledge base. Section 2 describes the MOSS command window, Section 3 introduces the concept display, Section 4 the browser, Section 5 the experimental help dalogs. The interactive editor is described in a separate document:

<Editor reference>

2 User's Interface - MOSS Window

The MOSS Window has several areas: A top line, a row of buttons underneath, a first display area labeled <input>, a second display area labeled <answer pane>, and in between the two display areas a progress line (that shows in grey). This section introduces the various features of the MOSS window taking the view of a naïve user discovering MOSS.

2.1 Loading an Application

One must first initialize the environment by loading an application, i.e. a knowledge base containing a set of concepts, individuals and methods implementing an application. The format of an application file and its location must obey some rules and examples are given in the Appendix. We assume here that we have an application ready to be loaded that models a family and is named "Family."

Loading an application consists in typing the name of the application in the APPLICATION slot and clicking the LOAD button. If every thing goes well, then some obscure messages are printed in the ANSWER PANE (Fig.3). When typing the file name the case is not important.

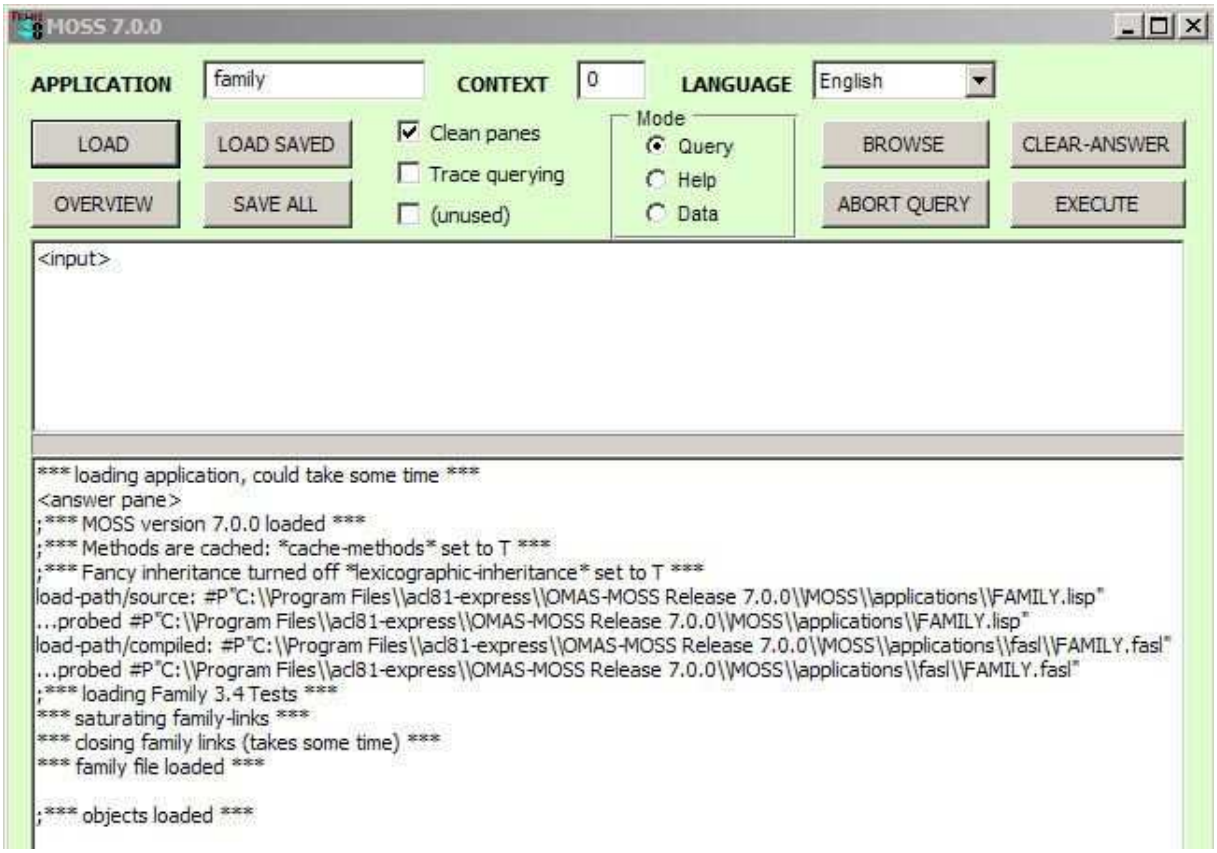

Figure 3: Loading the Family knowledge base

2.2 Exploring the Family World

Note The input pane is used to give commands. Ending the input depends on the mode we are in:

- in the query mode, inputs are queries and must obey the query format (refer to the document on MOSS queries). An important point is that there must be as many closing parentheses as opening ones. To execute the command, close enough parentheses and hit the return key.
- in the help mode, a command is a text that ends with a period or a question mark. Hitting the return key simply allows entering more text in continuation lines.

- in the data mode the user may type any Lisp expression, and hit the return key.

2.2.1 Simple Queries

Now, let us explore the family world. If we do not know what are the concepts of the FAMILY knowledge base, we can ask for them by using the formal request

```
(concept)
```

followed by carriage return. MOSS displays the available concepts in the ANSWER PANE (Fig.4). If we want to display results on a clean page each time, we check the clean panes box.

In the example, we obtain a list of four concepts: PERSON, COURSE, STUDENT, and ORGANISM. Note that during the access to the object base a progress line (dark blue in between the two large panes) shows the progression (this however is mostly for complex queries that take some time to execute).

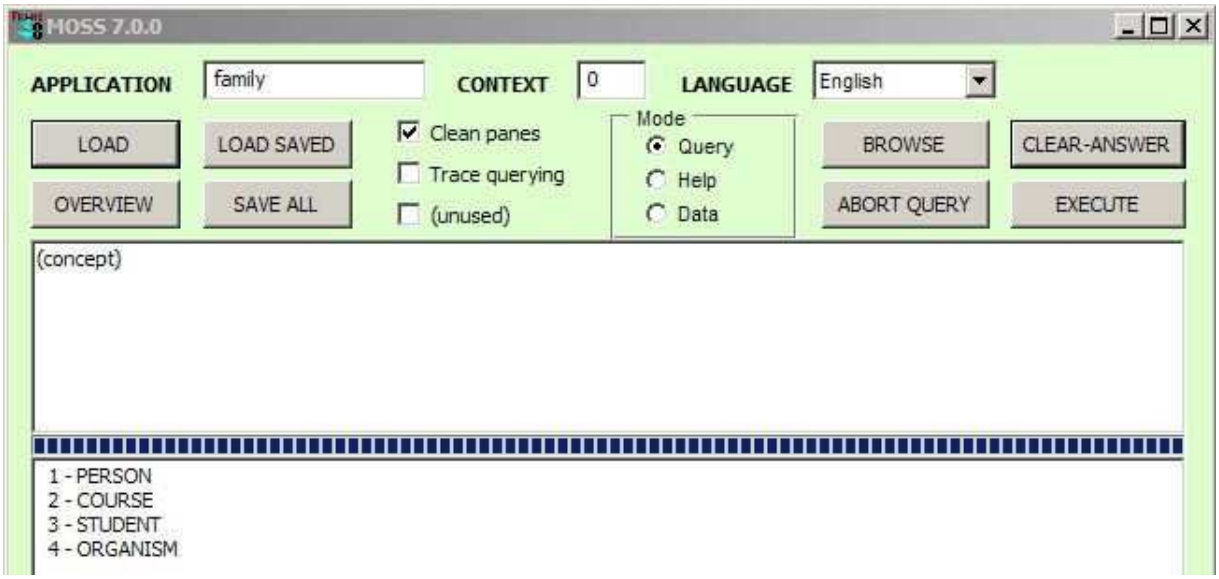

Figure 4: Concepts of the Family knowledge base

If we want now to obtain the list of individuals instances of person, we type

```
(person)
```

into the input pane. The result appears in the ANSWER PANE (Fig.5).

The objects are displayed using a =summary method attached to the concept of PERSON. In the example it prints the first name followed by the family name of a person.

2.2.2 More Complex Queries

More complex queries can be asked, reducing the list of results:

- Fig.6 for example gives the students whose name is "Barthès". The corresponding query is:

  ```
  ("student" ("name" :is "barthes"))
  ```

- Fig.7 gives the list of persons whose name is "Barthès" by using the entry point:

  ```
  "barthes"
  ```

  Note that the result could be obtained by using the query:

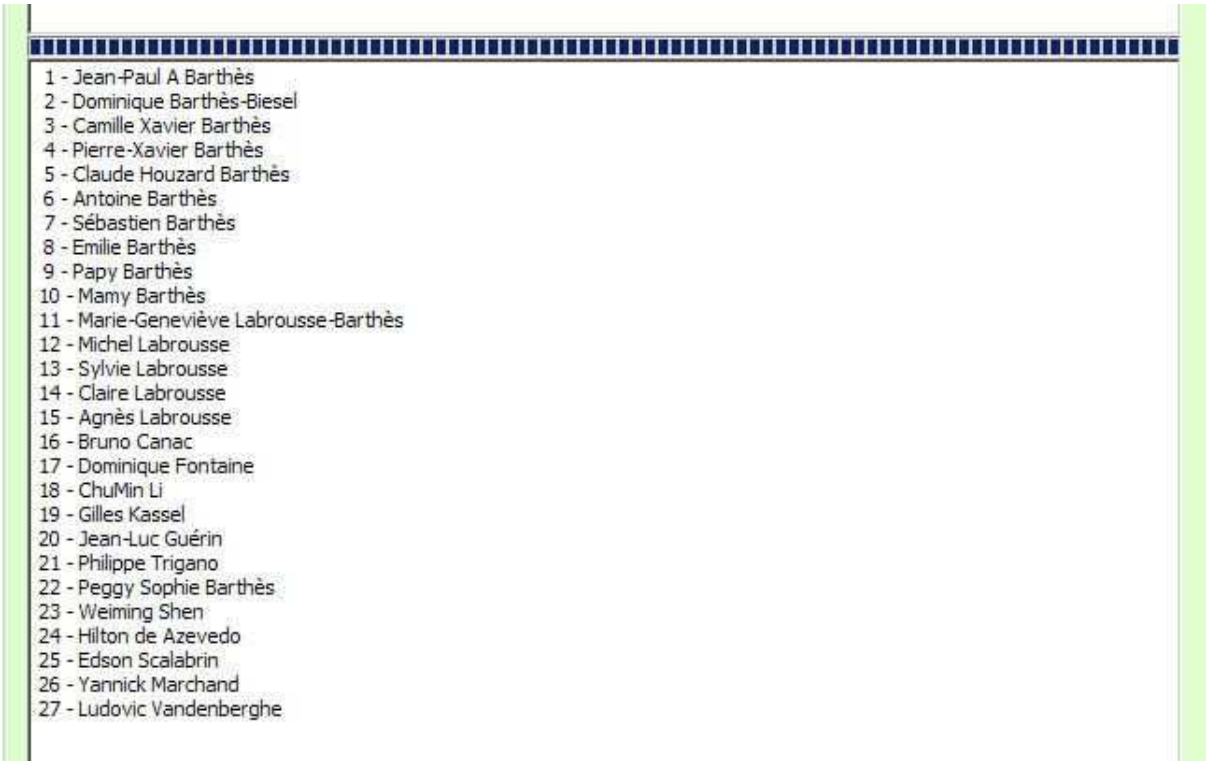

Figure 5: Person individuals in the Family knowledge base. Query: ("person")

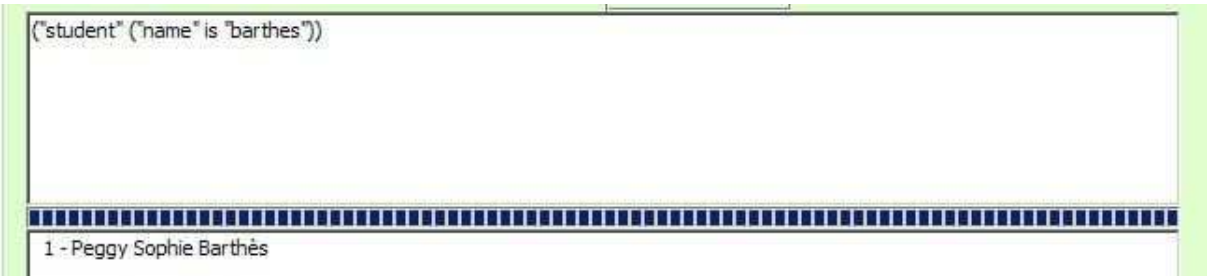

Figure 6: Query: ("student" ("name" is "barthes"))

```
("person"("name" :is "barthès"))
```

Using the entry point (index) directly returns the list of all objects in the knowledge base for which "barthes" is en entry point. It may include objects that are not persons.

- Fig.8 gives the list of persons whose name is "Barthès" and who have a sister. The corresponding query is:

```
("person" ("name" :is "barthes")
                ("sister" ("person")))
```

- Fig.9 gives the list of persons whose name is "Barthès" and who have at least two brothers. The corresponding query is:

```
("person" ("name" is "barthes")("brother" (> 1) ("person")))
```

The syntax for queries can be quite complex and is detailed in the document "MOSS Query System."

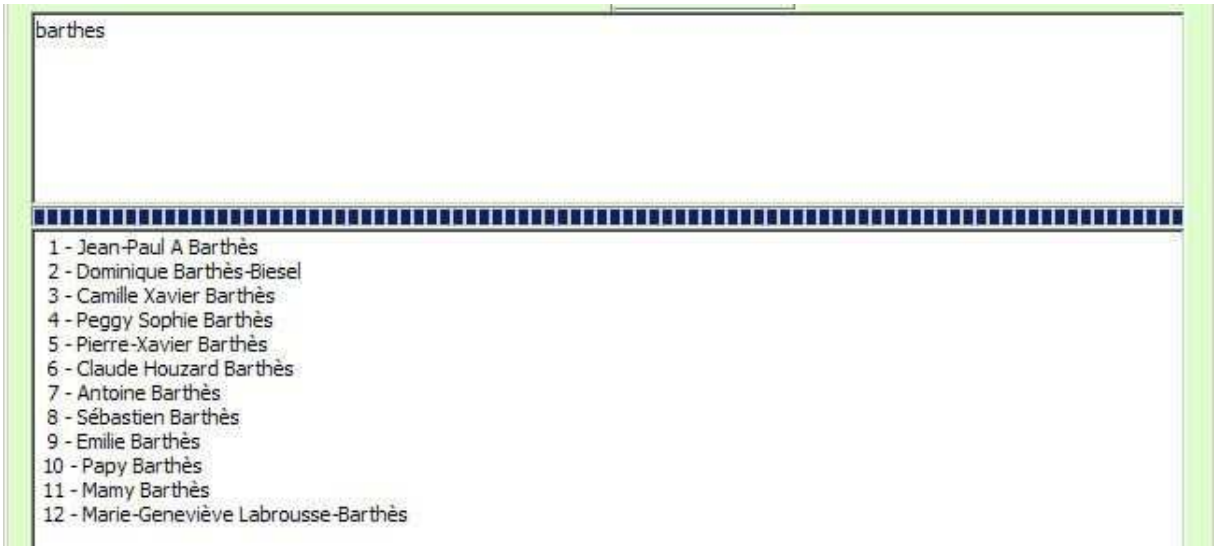

Figure 7: Query: "barthes"

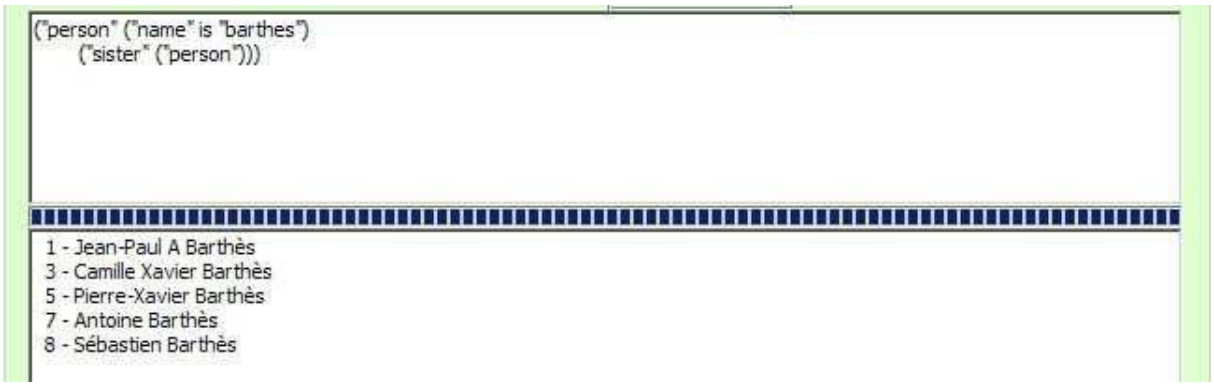

Figure 8: Query: ("person" ("name" is "barthes") ("sister" ("person")))

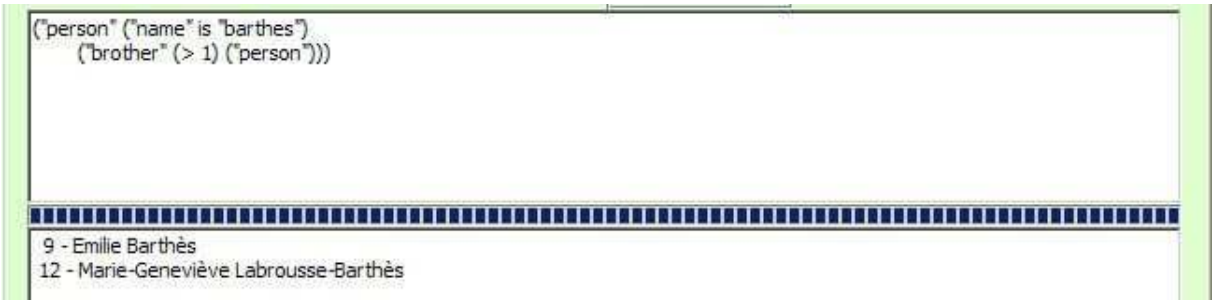

Figure 9: Query: ("person" ("name" is "barthes") ("brother" (> 1) ("person")))

Note If a query takes too long, it can be aborted using the ABORT QUERY button. The message "Query aborted at user request" appears in the ANSWER PANE.

2.3 Obtaining More Details about an Object

Objects listed in the answer area can be selected. For example, selecting the Claire Labrousse entry in the list of answers and clicking the BROWSE button makes a browsing window appear as shown Fig.10.

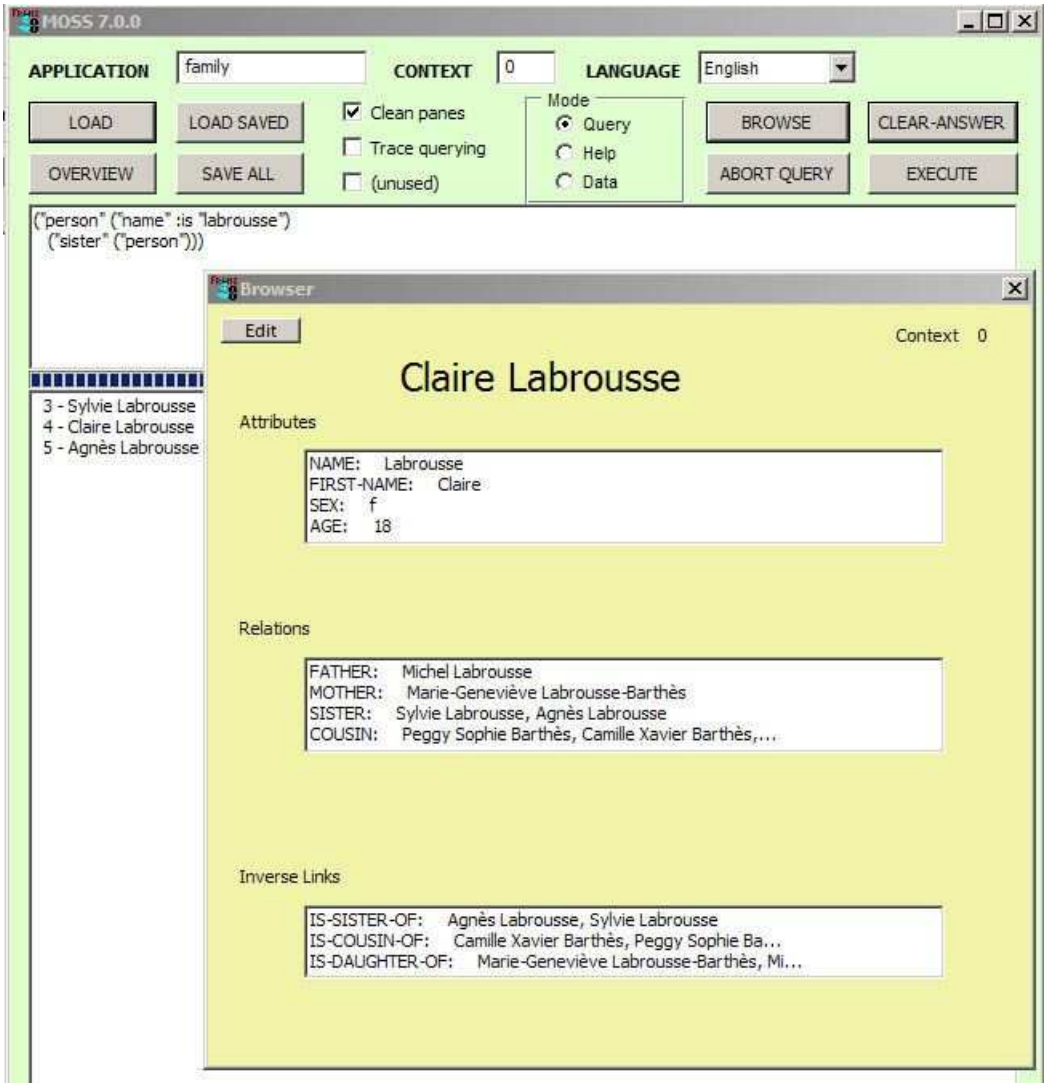

Figure 10: Detailing an object

Then it is possible to browse the database from there (see Section 4), or to edit the object [1] by pushing the Edit button (Fig.11), or to discard the browser window by closing it.

2.4 Browsing Classes and Instances "Ontology" Style

Clicking the OVERVIEW button triggers the CONCEPT DISPLAY, which is described in Section 3.

2.5 Other Buttons

The meaning of the other buttons of the MOSS window is the following:

- CLEAR ANSWER obvious role is to clear the ANSWER PANE.
- EXECUTE is mostly intended for debugging and executes any Lisp expression typed into the INPUT PANE, displaying the result into the ANSWER PANE.
- SAVE ALL builds a flat file of the application containing all classes, instances, methods, special variables, and functions of the application, stripping the environment of the MOSS system data (so that the system can be updated independently from applications). Using this feature allow saving the objects that have been edited with the MOSS editor, however the link to the original

[1] Note that the object will be edited in memory and not saved back into the original file

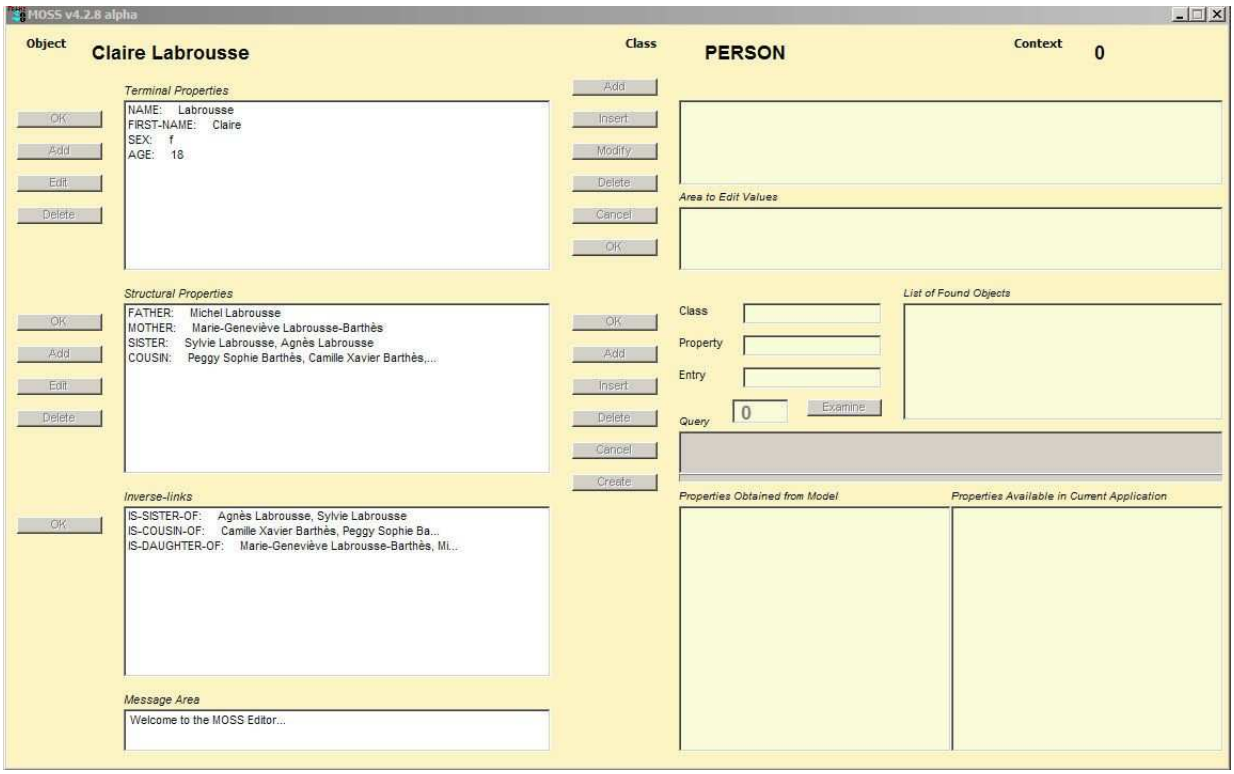

Figure 11: MOSS Editor

text file is lost. Thus, in practice the changes must be done to the original text file rather than to the structured objects directly in memory, which reduces the role of the editor near to nothing.

- LOAD SAVED loads an application from its saved file.
- BROWSE displays a selected object among the objects returned as the result of a query.
- CONTEXT indicates the context in which objects are displayed (a context is similar to a version). Here the test example only uses context 0. Other contexts are illegal.
- TRACE QUERYING is used mainly for debugging and can be safely ignored.
- LANGUAGE specifies the language to be chosen for displaying a knowledge base (provided it is a multilingual knowledge base).

3 MOSS Concept Display

The Concept Display as shown Fig.12 is used to browse application concepts and individuals. It has three main areas: a left concept area, a central instance area, and a right detail area.

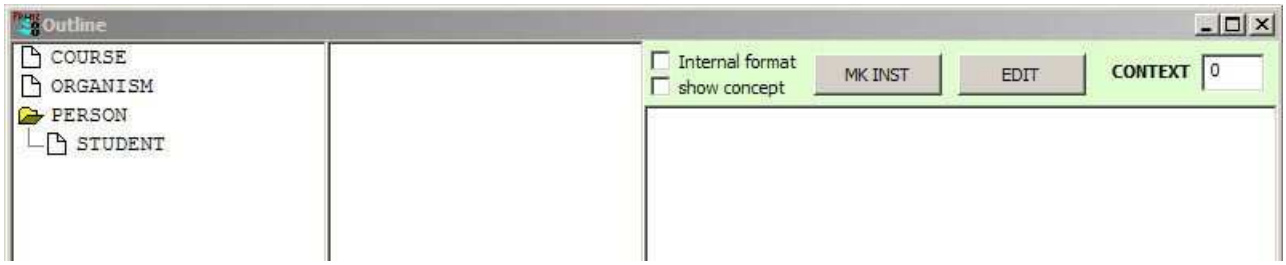

Figure 12: MOSS concept display

3.1 Concept Display Concept Area

The left part contains a tree representation of the application concepts/subconcepts, actually it is a forest. Top level concepts are represented as files, and concepts with subconcepts as folders.

3.2 Concept Display Central Area

When one selects a concept, the list of corresponding individuals is displayed in the central area (Fig.13).

Figure 13: The PERSON concept and corresponding individuals

Individuals corresponding to a concept include individuals corresponding to subconcepts. Individuals are displayed using the =summary method attached to the concept, or else as a list of internal identifiers when the method has not been defined (Fig.14).

3.3 Concept Display Detail Area

When an individual is selected the details are shown on the right hand side detail area (Fig.15).

It is possible to view the internal format of an object by checking the INTERNAL FORMAT checkbox. However, this is not intended for the casual user (Fig.16).

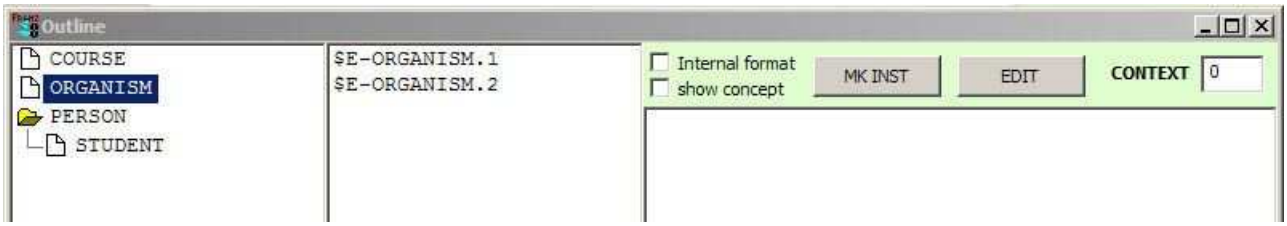

Figure 14: The ORGANISM concept and corresponding individuals (internal identifiers)

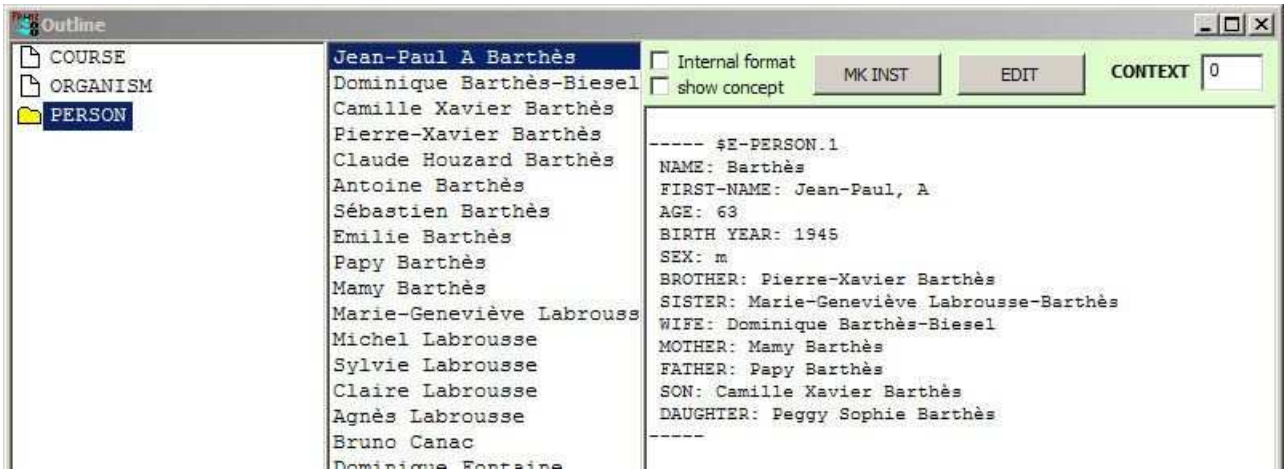

Figure 15: Details of the $E-PERSON.1 object

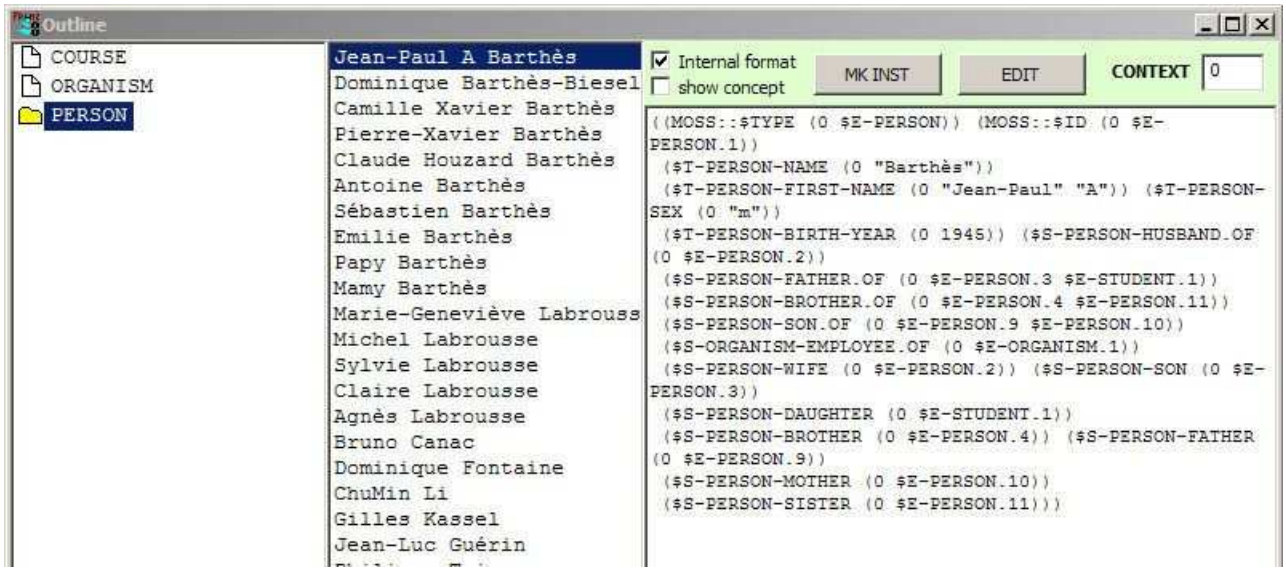

Figure 16: Internal representation of the $E-PERSON.1 object

It is also possible to view the concept itself rather than the selected individual by checking the show concept check box (Fig.17).

It is also possible by checking both check boxes to view the internal format of the concept PERSON.

3.4 Editing or Creating Objects

This can be done by clicking the EDIT button. The EDITOR window is then called.

It is possible to create a new individual by clicking the "MK INST" button. The EDITOR opens

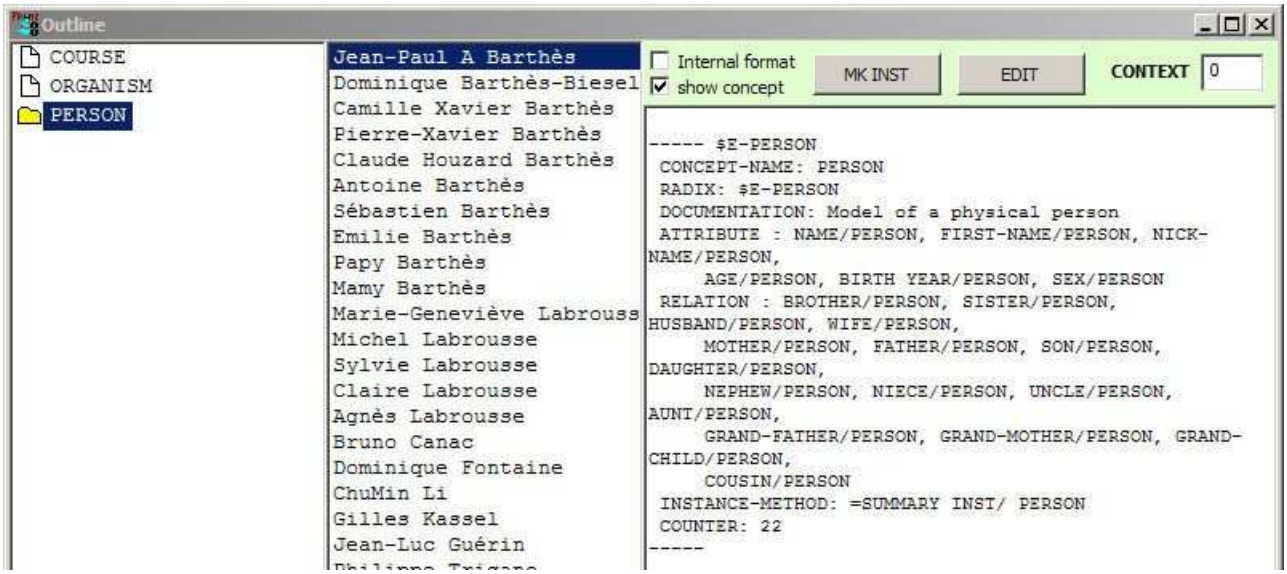

Figure 17: Detail of the PERSON concept

with a new empty individual of the selected concept.

However, in both cases the changes will occur in memory and will not be reported back to the original text file defining the knowledge base.

4 The MOSS Browser

The MOSS browser is called when clicking the BROWSE button in the MOSS window (Fig.18).

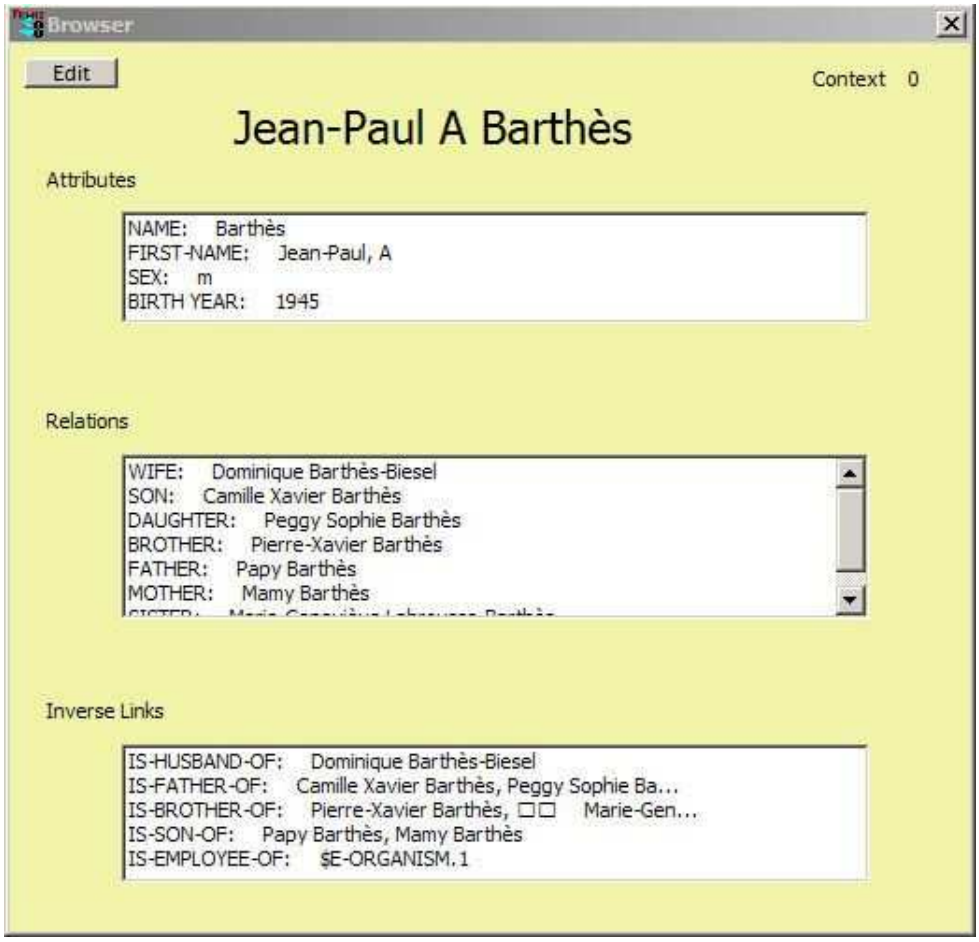

Figure 18: The MOSS browser window

The MOSS browser window displays the title of the object (concept or individual) as the result of applying the =summary method to the object. It contains three areas: one for attributes, one for relations, and one for inverse relations (inverse links). The context in which the object is displayed is shown at the top right corner, and an Edit button is available at the top left corner.

4.1 Browsing the Knowledge Base

MOSS allows navigation as follows:

Clicking on a relation or on an inverse link opens a small temporary window containing information about the linked objects (Fig.19). The first value is the internal key of the object (here $E-PERSON.9).

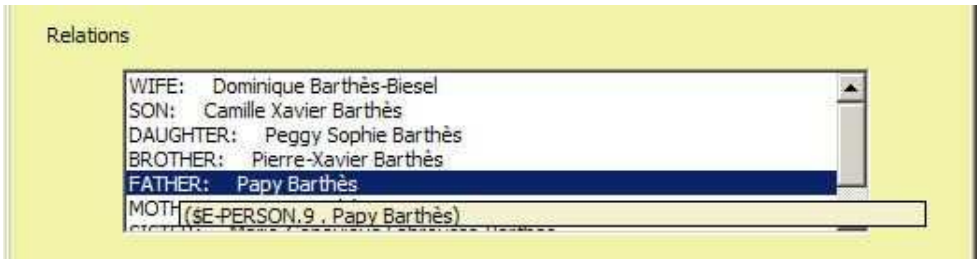

Figure 19: Objects linked to the current object

Double clicking on one of the objects of the temporary window opens a new window with the details of the new object. Windows are tiled.

Coming back to the previous window is done by closing the browser window.

Thus the user can navigate from object to object using direct or inverse links and stop for editing some of the objects.

4.2 Launching the Editor

Opening an Editor for the current browsed object is done by clicking the "Edit" button (Fig.20).

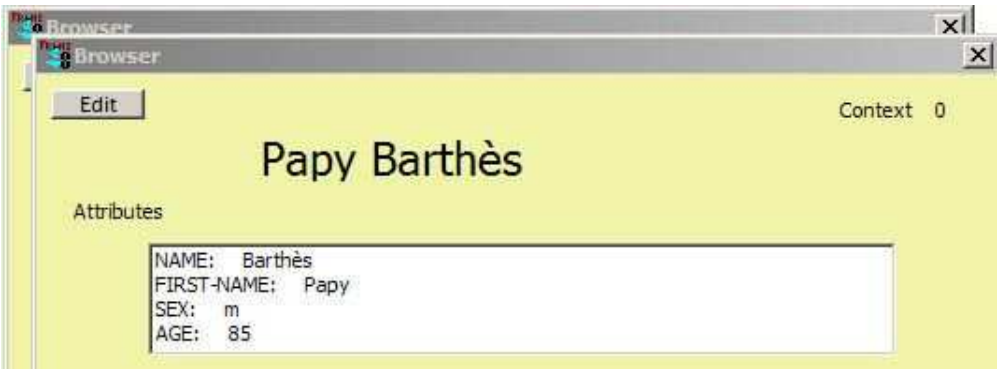

Figure 20: Edit and Exit button

Using the browser is very intuitive.

5 The HELP Dialog

MOSS contains an experimental version of an online help dialog using natural language. The dialog is started when the help mode is selected. In the help mode one can type natural language requests into the input pane of the MOSS window. Such requests are analyzed by MOSS and an answer is eventually produced. Examples of requests are:

- What can you do for me? (Fig.21)

- What is a concept? (Fig.22)

- Give me an example of concept. (Fig.23)

- Show me a concept.

- How do I create an attribute? (Fig.24)

Note that the input sentence must end with a period or a question mark that will not appear on the screen. The syntax needs not be exact and approximate questions, although syntactically incorrect, can be answered, e.g.

- How create attribute?

The sentence contains enough information to be processed.

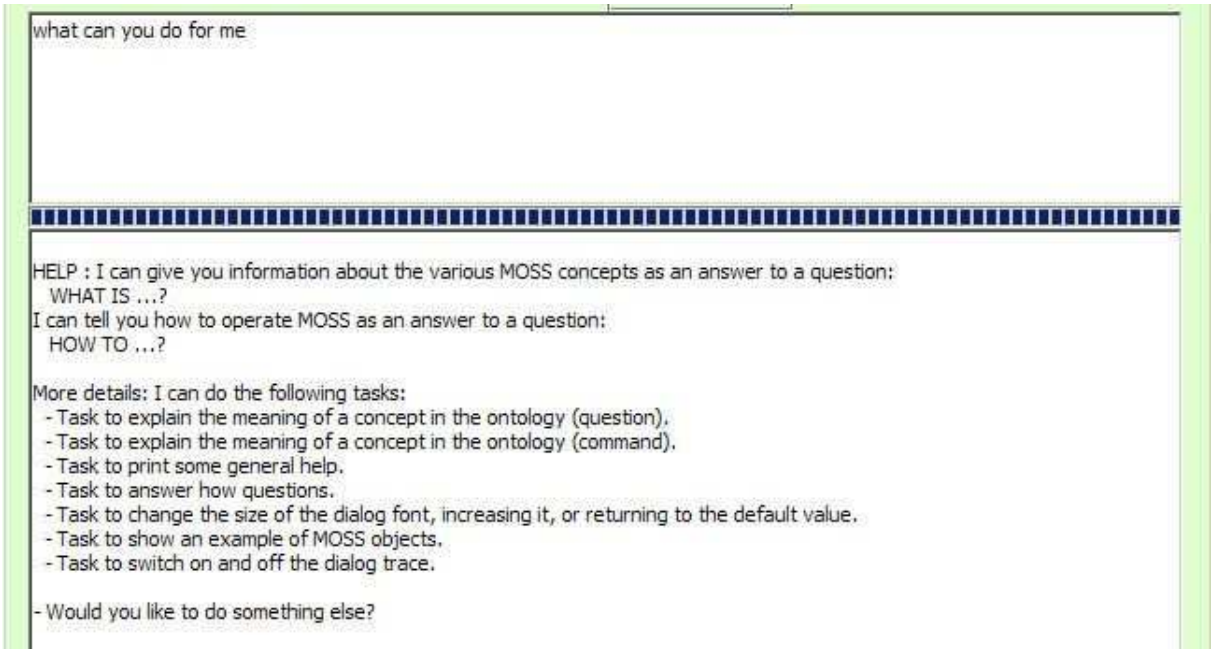

Figure 21: General question in the HELP dialog

Note The reader must be warned that the HELP system is quite incomplete and needs be upgraded with enough (canned) answers to be of real help.

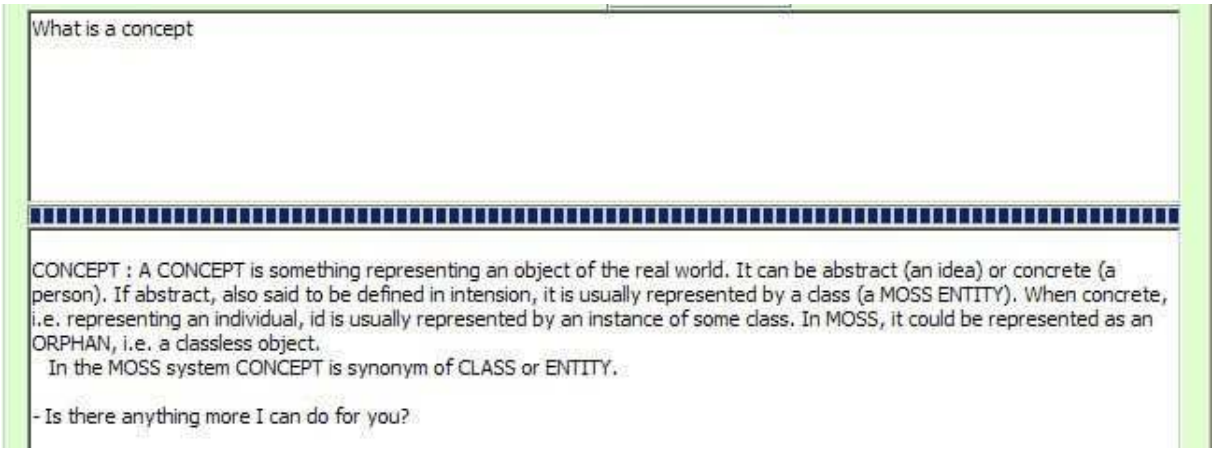

Figure 22: Asking for the definition of some MOSS concept

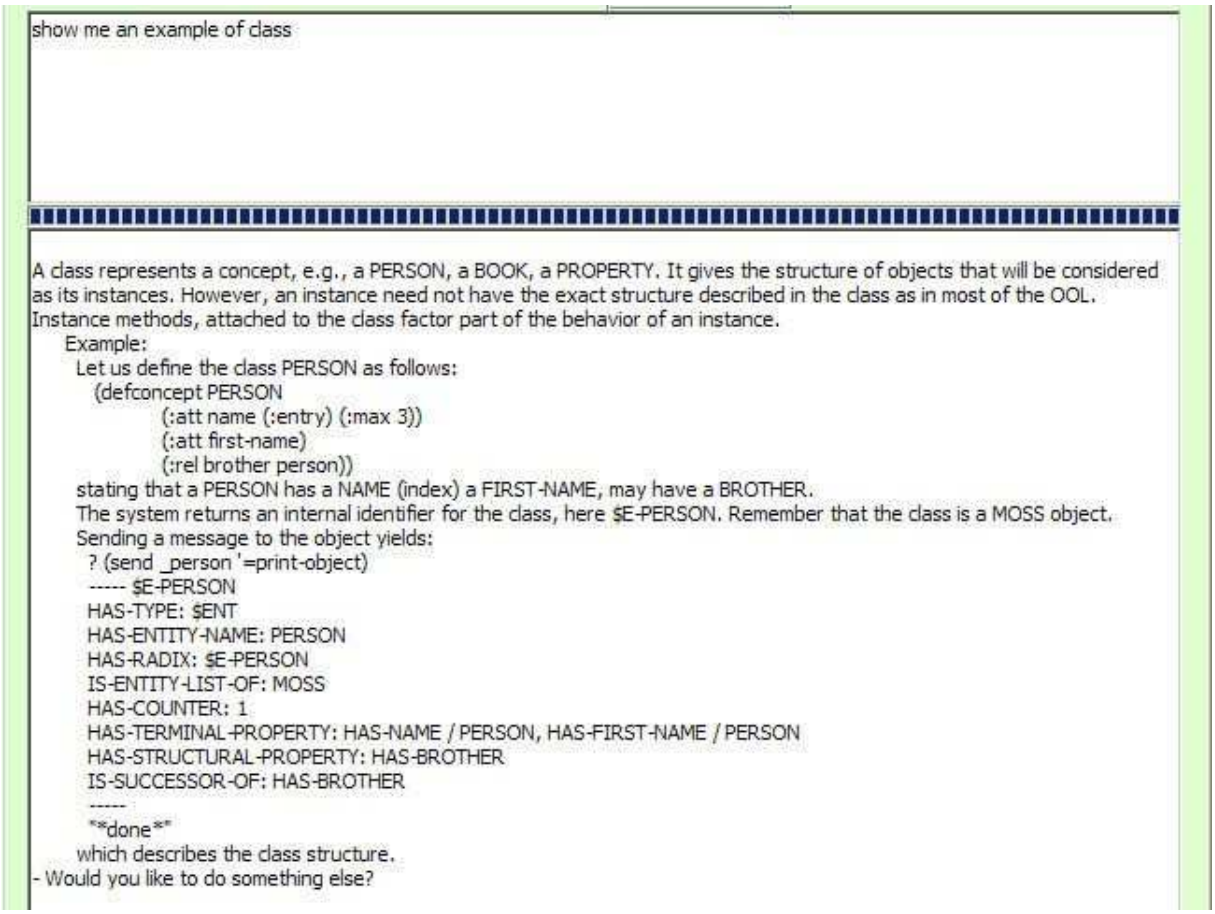

Figure 23: Displaying an example of class (concept)

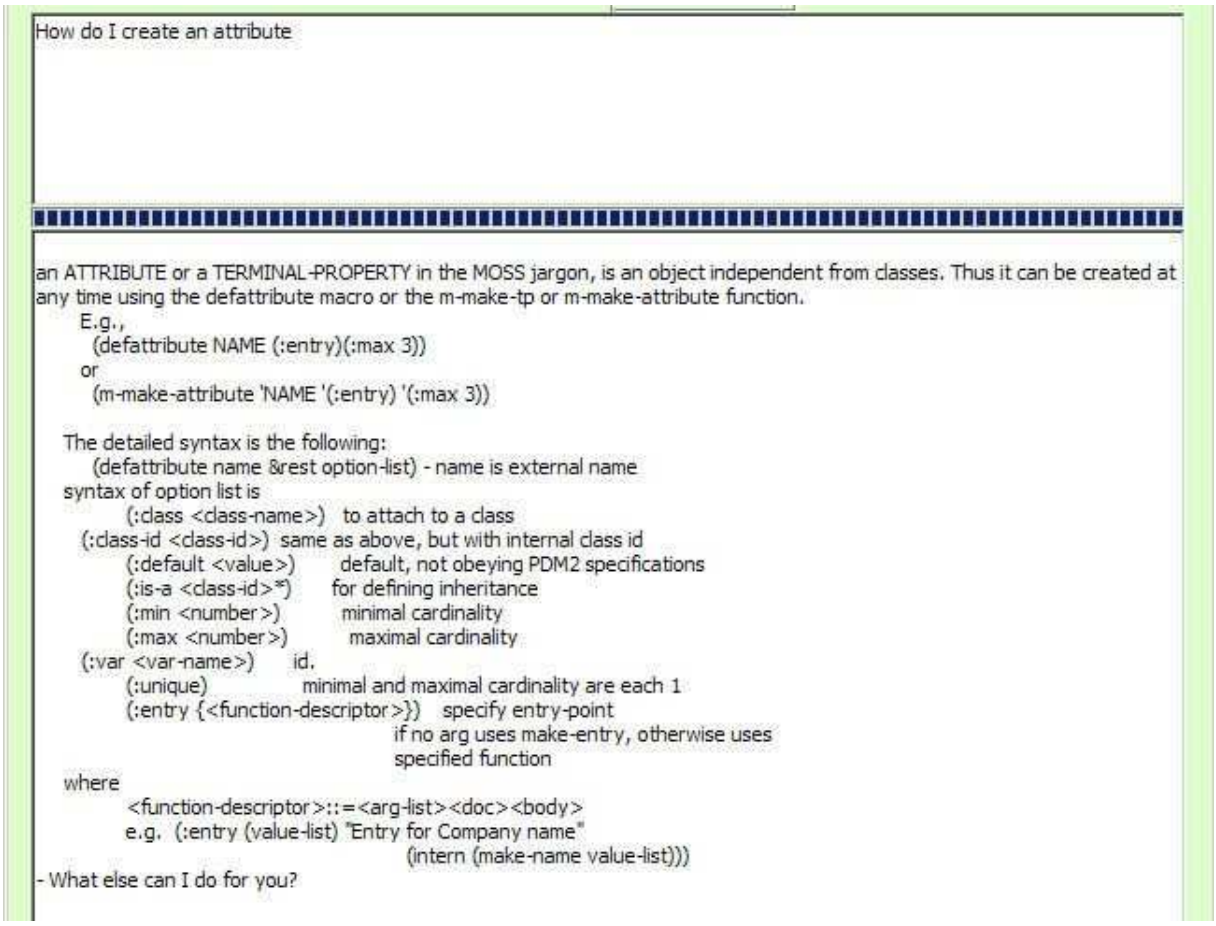

Figure 24: How to create an attribute

The appendix contains two examples of knowledge bases. The first one, FAMILY, deals with family links that are complex enough to exercise the possibilities of querying the knowledge base, the second one, V3S, comes from an OMAS application, and shows how to organize an ontology that can be submitted to the SOL compiler in order to produce an OWL output as well as various files (browser, grapher, ...).

A The Family Knowledge Base

The family knowledge base is the one used in this document. It is a fairly old test file that contains persons, organisms and links between such persons and organisms. Family links are complex enough to generate interesting queries.

The family knowledge base is produced in the common-graphics or common-lisp package. Four concepts are defined. Two macros, mp and ms, are defined to ease the input of individuals, a small expert system is defined to close the family links.

```
;;;============================================================================
;;;06/08/33
;;; F A M I L Y (File FAMILY-Mac.LSP)
;;;
;;; Copyright Jean-Paul BarthËs @ UTC 1994
;;;
;;;============================================================================

;;; This file creates a family for checking the LOB window system, and the query
;;; system. Thus it requires the corresponding modules.
;;;1995
;;;  2/13 Adding relationships in the family to be able to use the file for
;;;       testing the query system
;;;       Adding other persons not actual members of the family
;;;  2/18 Adding a mechanism for saturating the family links -> v 1.1.1
;;;  2/19 Adding the age property to all persons and birth date (year)
;;;       Changing the version name to comply with the curreunt MOSS version 3.2
;;;       -> v 3.2.2
;;;  2/25 Adding orphans -> v 3.2.3
;;;1997
;;;  2/18 Adding in-package for compatibility with Allegro CL on SUNs
#|
2003
 0517 used to test changes to the new MOSS v4.2a system
2004
 0628 removing ids from the class definitions
 0715 adding :common-graphics-user-package for ACL environments
 0927 changinging the è and è characters to comply with ACL coding...
2005
 0306 defining a special file for mac because of the problem of coding the French
      letters
 1217 Changing the file to adapt to Version 6.
2006
 0822 adding count to the inference loop
|#

#+MCL
(in-package "COMMON-LISP-USER")
#+(AND MICROSOFT-32 IDE)
```

```
(in-package :common-graphics-user)
#-IDE
(in-package :common-lisp-user)

#+(or ALLEGRO MICROSOFT-32)
(eval-when (:load-toplevel :compile-toplevel)
  (use-package :moss))

(moss::mformat "~%;*** loading Family 3.4 Tests ***")

(eval-when (load compile eval)
  (use-package :moss))

;;; remove tracing while objects are built
(moss::toff)

(setq moss::*allow-forward-references* nil)

(defconcept (:en "PERSON" :fr "PERSONNE")
  (:doc "Model of a physical person")
  (:tp (:en "NAME" :fr "NOM") (:min 1)(:max 3)(:entry))
  (:tp (:en "FIRST-NAME" :fr "PRENOM"))
  (:tp (:en "NICK-NAME" :fr "SURNOM"))
  (:tp (:en "AGE" :fr "AGE") (:unique))
  (:tp (:en "BIRTH YEAR" :fr "ANNEE DE NAISSANCE") (:unique))
  (:tp (:en "SEX" :fr "SEXE") (:name sex)(:unique))
  (:sp (:en "BROTHER" :fr "FRERE") "PERSON")
  (:sp (:en "SISTER" :fr "SOEUR") "PERSON")
  (:rel (:en "HUSBAND" :fr "MARI") "PERSON")
  (:rel (:en "WIFE" :fr "FEMME") "PERSON")
  (:rel (:en "MOTHER" :fr "MERE") "PERSON")
  (:rel (:en "FATHER" :fr "PERE") "PERSON")
  (:rel (:en "SON" :fr "FILS") "PERSON")
  (:rel (:en "DAUGHTER" :fr "FILLE") "PERSON")
  (:rel (:en "NEPHEW" :fr "NEVEU") "PERSON")
  (:rel (:en "NIECE" :fr "NIECE") "PERSON")
  (:rel (:en "UNCLE" :fr "ONCLE") "PERSON")
  (:rel (:en "AUNT" :fr "TANTE") "PERSON")
  (:rel (:en "GRAND-FATHER" :fr "GRAND PERE") "PERSON")
  (:rel (:en "GRAND-MOTHER" :fr "GRAND MERE") "PERSON")
  (:rel (:en "GRAND-CHILD" :fr "PETITS-ENFANTS") "PERSON")
  (:rel (:en "COUSIN" :fr "COUSIN") "PERSON")
  )

(defconcept (:en "COURSE" :fr "COURS")
  (:tp (:en "TITLE" :fr "TITRE") (:unique))
  (:tp (:en "LABEL" :fr "CODE") (:entry))
  (:doc "Course followed usually by students")
  )

(defconcept (:en "STUDENT" :fr "ETUDIANT")
  (:is-a "PERSON")
  (:rel (:en "COURSES" :fr "COURS") "COURSE")
```

```
  )

(defconcept (:en "ORGANISM" :fr "ORGANISME")
  (:tp (:en "NAME" :fr "NOM"))
  (:tp (:en "ABBREVIATION" :fr "SIGLE") (:entry))
  (:rel (:en "EMPLOYEE" :fr "EMPLOYE") "PERSON")
  (:rel (:en "STUDENT" :fr "ETUDIANT") "STUDENT")
  )

;;;(defownmethod =make-entry (HAS-NAME HAS-PROPERTY-NAME TERMINAL-PROPERTY
;;;                                    :class-ref PERSON)
;;;  (data)
;;;  "Builds an entry point for person using standard make-entry primitive"
;;;  (moss::%make-entry-symbols data))

(defownmethod =if-needed (AGE HAS-PROPERTY-NAME ATTRIBUTE :class-ref PERSON)
  (obj-id)
  (let ((birth-year (car (send obj-id '=get 'HAS-BIRTH-YEAR))))
    (if (numberp birth-year)
      ;; do not forget, we must return a list of values
      (list (- (get-current-year) birth-year)))))

(definstmethod =summary PERSON ()
  "Extract first names and names from a person object"
  ;(append (HAS-FIRST-NAME) (HAS-NAME))
  (let ((result (format nil "~{~A~^ ~} ~{~A~^-~}" (Has-first-name) (has-name))))
    (if (string-equal result " ")
      (setq result "<no data available>"))
    (list result)))

;;; First define a macro allowing to create persons easily

(defMacro mp (name first-name sex var &rest properties)
  `(progn
     (defVar ,var)
     (definidividual "PERSON"
                     ("NAME" ,@name)
                     ("FIRST-NAME" ,@first-name)
                     ("SEX" ,sex)
                     (:var ,var)
                     ,@properties)))

(defMacro ms (name first-name sex var &rest properties)
  `(progn
     (defVar ,var)
     (definidividual STUDENT
                     ("NAME" ,@name)
                     ("FIRST-NAME" ,@first-name)
                     ("SEX" ,sex)
                     (:var ,var)
                     ,@properties)))

;;; Instances
```

```
(mp ("BarthËs") ("Jean-Paul" "A") "m" _jpb ("BIRTH YEAR" 1945))
(mp ("BarthËs" "Biesel") ("Dominique") "f" _dbb ("HUSBAND" _jpb)("BIRTH YEAR" 1946))
(mp ("BarthËs") ("Camille" "Xavier") "m" _cxb ("FATHER" _jpb) ("MOTHER" _dbb)
    ("BIRTH YEAR" 1981))
(ms ("BarthËs") ("Peggy" "Sophie") "f" _psb ("FATHER" _jpb) ("MOTHER" _dbb)
    ("BROTHER" _cxb)("BIRTH YEAR" 1973))
(mp ("BarthËs") ("Pierre-Xavier") "m" _pxb ("BROTHER" _jpb)("AGE" 54))
(mp ("BarthËs") ("Claude" "Houzard") "f" _chb ("HUSBAND" _pxb)("AGE" 52))
(mp ("BarthËs") ("Antoine") "m" _ab ("FATHER" _pxb)("MOTHER" _chb)
    ("COUSIN" _cxb _psb)("AGE" 27))
(mp ("BarthËs") ("SËbastien") "m" _sb ("FATHER" _pxb)("MOTHER" _chb)
    ("BROTHER" _ab)("COUSIN" _cxb _psb)("AGE" 24))
(mp ("BarthËs") ("Emilie") "f" _eb ("FATHER" _pxb)("MOTHER" _chb)
    ("BROTHER" _ab _sb)("COUSIN" _cxb _psb)("AGE" 21))
(mp ("BarthËs") ("Papy") "m" _apb ("SON" _jpb _pxb)("AGE" 85))
(mp ("BarthËs") ("Mamy") "f" _mlb ("SON" _jpb _pxb)("HUSBAND" _apb)("AGE" 81))
(mp ("Labrousse" "BarthËs") ("Marie-GeneviËve") "f" _mgl ("FATHER" _apb)
    ("MOTHER" _mlb)("AGE" 48)("BROTHER" _jpb _pxb))
(mp ("Labrousse") ("Michel") "m" _ml ("WIFE" _mgl)("AGE" 52))
(mp ("Labrousse") ("Sylvie") "f" _sl ("FATHER" _ml)("MOTHER" _mgl)
    ("COUSIN" _psb _cxb _ab _sb _eb)("AGE" 19))
(mp ("Labrousse") ("Claire") "f" _cl ("FATHER" _ml)("MOTHER" _mgl)("SISTER" _sl)
    ("COUSIN" _psb _cxb _ab _sb _eb)("AGE" 18))
(mp ("Labrousse") ("AgnËs") "f" _al ("FATHER" _ml)("MOTHER" _mgl)
    ("SISTER" _sl _cl)("COUSIN" _psb _cxb _ab _sb _eb)("AGE" 15))
(mp ("Canac") ("Bruno") "m" _bc)

;---
(ms ("Shen") ("Weiming") "m" _wms)
(ms ("de Azevedo") ("Hilton") "m" _hda)
(ms ("Scalabrin") ("Edson") "m" _es)
(ms ("Marchand") ("Yannick") "m" _ym)
(ms ("Vandenberghe") ("Ludovic") "m" _lv)
;---
(mp ("Fontaine") ("Dominique") "m" _df)
(mp ("Li") ("ChuMin") "m" _cml)
(mp ("Kassel") ("Gilles") "m" _gk)
(mp ("GuÈrin") ("Jean-Luc") "m" _jlg)
(mp ("Trigano") ("Philippe") "m" _pt)

(m-definstance ORGANISM ("ABBREVIATION" "UTC")
               ("NAME" "UniversitÈ de Technologie de CompiËgne")
               ("EMPLOYEE" _jpb _dbb _df _gk _pt)
               ("STUDENT" _wms _hda))
(m-definstance ORGANISM  ("ABBREVIATION" "IC")
               ("NAME" "Imperial College")
               ("STUDENT" _psb))

(moss::mformat "~%*** saturating family-links ***")

;(setq *persons* (send *qh* '=process-query '(person)))
(defParameter *persons*
```

```
  (list _jpb _dbb _cxb _psb _pxb _chb _ab _sb _eb _apb _mlb _mgl _ml _sl _cl _al _wms
        _hda _es _ym _lv _df _cml _gk _jlg _pt))
(defVar *rule-set* nil)
(setq *rule-set* nil)
(defMacro ar (&rest rule)
  `(push ',rule *rule-set*))
;;;  Rules to close the family links
;;;
;;; If person P1 has-sex M
;;;              has-brother P2
;;;    then person P2 has-brother P1
(ar "m" has-brother has-brother)
;;;
;;; If person P1 has-sex M
;;;              has-sister P2
;;;    then person P2 has-brother P1
(ar "m" has-sister has-brother)
;;;
;;; If person P1 has-sex M
;;;              has-father P2
;;;    then person P2 has-son P1
(ar "m" has-father has-son)
;;;
;;; If person P1 has-sex M
;;;              has-mother P2
;;;    then person P2 has-son P1
(ar "m" has-mother has-son)
;;;
;;; If person P1 has-cousin P2
;;;    then person P2 has-cousin P1
(ar "m" has-cousin has-cousin)
(ar "f" has-cousin has-cousin)
;;;
;;; If person P1 has-sex M
;;;              has-son P2
;;;    then person P2 has-father P1
(ar "m" has-son has-father)
;;;
;;; If person P1 has-sex M
;;;              has-daughter P2
;;;    then person P2 has-father P1
(ar "m" has-daughter has-father)
;;;
;;; If person P1 has-sex M
;;;              has-wife P2
;;;    then person P2 has-husband P1
(ar "m" has-wife has-husband)
;;;
;;; If person P1 has-sex F
;;;              has-brother P2
;;;    then person P2 has-sister P1
(ar "f" has-brother has-sister)
;;;
```

```
;;; If person P1 has-sex F
;;;              has-sister P2
;;;    then person P2 has-sister P1
(ar "f" has-sister has-sister)
;;;
;;; If person P1 has-sex F
;;;              has-father P2
;;;    then person P2 has-daughter P1
(ar "f" has-father has-daughter)
;;;
;;; If person P1 has-sex F
;;;              has-mother P2
;;;    then person P2 has-daughter P1
(ar "f" has-mother has-daughter)
;;;
;;; If person P1 has-sex F
;;;              has-son P2
;;;    then person P2 has-mother P1
(ar "f" has-son has-mother)
;;;
;;; If person P1 has-sex F
;;;              has-daughter P2
;;;    then person P2 has-mother P1
(ar "f" has-daughter has-mother)
;;;
;;; If person P1 has-sex F
;;;              has-husband P2
;;;    then person P2 has-wife P1
(ar "f" has-husband has-wife)


;;; the version using the MOSS service functions does not seem significantly
;;; faster than the one using the messages

(defun apply-rule (p1 p2 rule)
  "produces a list of messages to send to modify the data.
Arguments:
   p1: person 1
   p2: person 2
   rule: (sex) rule to be applied
Return:
   nil or message to be sent."
  ;; note that we use =get and not =get-id that would require the local id of
  ;; the specified property, e.g. $t-person-name or $t-student-name, which
  ;; cannot be guarantied in a global rule
  (let ((sex (car rule))
        (sp-name (cadr rule))
        (ret-sp-name (caddr rule)))
    (if (and (equal sex
                    (car (moss::%get-value
                          P1
                          (moss::%get-property-id-from-name _jpb 'has-sex))))
             (member P2 (moss::%get-value
```

```
                        P1
                        (moss::%get-property-id-from-name P1 sp-name))))
      `(send ',P2 '=add-related-objects ',ret-sp-name ',P1))))

#|
(defun apply-rule (p1 p2 rule)
  "produces a list of messages to send to modify the data.
Arguments:
   p1: person 1
   p2: person 2
   rule: (sex) rule to be applied
Return:
   nil or message to be sent."
  ;; note that we use =get and not =get-id that would require the local id of
  ;; the specified property, e.g. $t-person-name or $t-student-name, which
  ;; cannot be guarantied in a global rule
  (let ((sex (car rule))
        (sp-name (cadr rule))
        (ret-sp-name (caddr rule)))
    (if (and (equal sex (car (send P1 '=get 'HAS-SEX)))
             (member P2 (send P1 '=get sp-name)))
      `(send ',P2 '=add-sp ',ret-sp-name ',P1))))
|#

#|
(apply-rule _pxb _jpb '("m" HAS-BROTHER HAS-BROTHER))
|#

(defun close-family-links (family-list rule-set)
  "Function that takes a list of persons and computes all the links that must be added
to such persons to close the parent links brother, sister, father, mother, son,
daughter, cousin. It executes a loop until no more changes can occur, applying
rules from a rule set. The result is a list of instructions to be executed."
  (let ((change-flag t) action action-list (count 0))
    (print count)
    ;; Loop until no more rule applies
    (loop
      (unless change-flag (return nil))
      ;; reset flag
      (setq change-flag nil)
      ;; loop on couple of objects in the family-list
      (dolist (P1 family-list)
        (dolist (P2 family-list)
          ;; loop on each rule
          (dolist (rule rule-set)
            ;; if the rule applies it produces an instruction to be executed
            (setq action (apply-rule P1 P2 rule))
            ;; which is appended to the instruction list
            (when (and action
                       (not (member action  action-list :test #'equal)))
              ;; print a mark into the debug trace to tell user we are doing
              ;; some work
              (prin1 '*)
```

```
                (incf count)
                (if (eql (mod count 50) 0) (print count)) ; new line after 50 *s
                (push action action-list)
                ;; in addition we mark that a changed occured by setting the change flag
                (setq change-flag t))))))
    ;; return list of actions
    (reverse action-list)))

(moss::mformat "~%*** closing family links (takes some time) ***")

(defVar instructions)
(setq instructions (close-family-links *persons* *rule-set*))
(mapcar #'eval instructions)

;;; Adding orphans

(m-defobject ("NAME" "Dupond")("sex" "m"))
(m-defobject ("NAME" "Durand" )("SEX" "f")("AGE" 33))

(moss::mformat "~%*** family file loaded ***")
```

B The V3S Ontology and Knowledge Base

The V3S knowledge base is a small example of a knowledge base used by an OMAS agent named COLOMBO in charge of modeling an industrial environment. The small extract is intended to indicate some features of the ontology file.

The following points should be noticed:

- The ontology is defined in the name space (package) :colombo.
- The ontology uses the :moss and :omas name spaces.
- A `defontology` macro gives the name of the ontology and the language in which it is defined (here French):

```
(defontology
  (:title "COLOMBO ontology")
  (:version "1.0")
  (:language :fr)
  )
```

- The ontology uses `defchapter` and `defsection` macros to organize the ontology as a knowledge book.
- Actions contain default rules that are interpreted by a special inference engine contained in the COLOMBO agent.

```
;;;-*- Mode: Lisp; Package: "COLOMBO" -*-
;;;============================================================================
;;;08/02/03
;;;             AGENT COLOMBO: ONTOLOGY (file COLOMBO-ontology.lisp)
;;;             Copyright Barthes@UTC, 2008
;;;
;;;============================================================================

;;; the idea is to model objects of the environment in a separate agent: COLOMBO
;;; Each agent modelling a specific human, will send order or inquiries to COLOMBO
;;; COLOMBO maintains the model representing the environment, regardless of what
;;; each agent may believe
;;; Actions are represented by methods that hat preconditions and consequences
;;; when the application is successful or if it is a failure
;;; Preconditions are tested typically by checking the status (vale) of some
;;; property of some object. This can be done by applying a query to the object base

(eval-when (compile load eval)
  (unless (find-package :COLOMBO) (make-package "COLOMBO")))

(in-package :COLOMBO)

(eval-when (compile load eval)
  (use-package :moss)
  (use-package :omas)
  ) ; reexport it for use in other packages

(defontology
  (:title "COLOMBO ontology")
  (:version "1.0")
```

```
  (:language :fr)
  )

;;;============================================================================ ACTIONS

(defchapter
  (:name :en "mv-actions"
         :fr "actions MV"))

(defconcept "mv-action"
  (:att "nom" (:entry) (:unique))
  (:att "position")
  (:att "orientation")
  (:att "rules")
  (:att "method" (:default =close-gate))
  )

;;;============================================================================ OBJETS

(defchapter
  (:name :en "objects"
         :fr "objets"))

;;;------------------------------------------------------ OBJETS OUVRABLES

(defsection
  (:name  :fr "objet ouvrable"))

(defconcept "objet-ouvrable"
  )

;;;------------------------------------------------------------------------- VANNE

(defconcept "vanne"
  (:is-a "objet-ouvrable")
  (:att "identifiant" (:unique)(:entry))
  (:att "position" (:unique))
  (:att "orientation" (:unique) (:default '(1 0 0 0)))
  ;(:rel "mv-action" (:to "mv-action"))
  (:att "etat" (:one-of "normal" "grippe" "casse")(:default "normal"))
  (:att "statut" (:one-of "ouvert" "ferme" "inconnu")(:default "inconnu"))
  (:doc :fr "une VANNE est un dispositif present sur une canalisation qui controle ~
             le flux de liquide dans celle-ci.")
  )

(definstmethod =summary VANNE ()
  (HAS-IDENTIFIANT))

;;;---------------------------------------------------------------------- ACTIONS

(defchapter
  (:name :en "mv-actions"
         :fr "actions MV"))
```

```
;;;------------------------------------------------------------ OUVRIR VANNE

(defindividual "mv-action"
  ("nom" "ouvrir vanne")
  ("rules"
   (:if (?* "vanne" ("statut" :is "ouvert"))
     :then
     (:return :success))
   (:if (?* "vanne" ("statut" :is "ferme"))
     (?* "vanne" ("etat" :is-not "grippe"))
     :then
     (:set ?* "statut" "ouvert")
     (:return :success))
   (:if
     (?* "vanne" ("statut" :is "grippe"))
     :then
     (:return :failure))
   )
  (:var _ouvrir-vanne))

(definstmethod =summary MV-ACTION ()
  (HAS-NOM))

;;;------------------------------------------------------------ FERMER VANNE

(defindividual "mv-action"
  ("nom" "fermer vanne")
  ("rules"
   (:if (?* "vanne" ("statut" :is "ferme"))
     :then
     (:return :success))
   (:if (?* "vanne" ("statut" :is "ouvert"))
     (?* "vanne" ("etat" :is-not "grippe"))
     :then
     (:set ?* "statut" "ferme")
     (:return :success))
   (:if
     (?* "vanne" ("statut" :is "grippe"))
     :then
     (:return :failure))
   )
  (:var _fermer-vanne))

;;; adding default actions to vanne

(defrelation "mv-action" "vanne" "mv-action"
  (:default _ouvrir-vanne _fermer-vanne))

;;;============================================================================
;;;
;;;                                  INDIVIDUALS
;;;
```

```
;;;=============================================================================

(defchapter
  (:name :en "OBJECTS"
         :fr "OBJETS"))

;;;----------------------------------------------------------------------- VANNES

(defsection
  (:name :en "gates"
         :fr "vannes"))

(catch :error (defindividual "vanne"
  ("identifiant" "V234")
  ("statut" "ferme")
  ("etat" "grippe")
  (:var _vanne234)))

(defindividual "vanne"
  ("identifiant" "V345")
  ("statut" "ouvert")
  (:var _vanne345))

(defindividual "vanne"
  ("identifiant" "V101")
  ("statut" "ferme")
  (:var _vanne101))

:EOF
```