Yet Another Yet Another (YAYA)

Jordi Alvarez

Report LSI–96–15–T

Yet Another Yet Another (YAYA)[1]

Jordi Alvarez
Universitat Politècnica de Catalunya
Departament de Llenguatges i Sistemes Informàtics
Mòdul C5, Campus Nord, 08034 Barcelona
jalvarez@lsi.upc.es

[1]It was Yet Another Frame-based System (YAFS). And at the beginning it was A Frame-based System (AFS) because of my poor imagination.

Resum

Aquest és el manual de referència de YAYA (Yet Another Yet Another), un sistema de frames que incorpora eines de raonament terminològic i representació probabilística.

Índex

1 Introducció **3**
- 1.1 Motivació . . . 3
- 1.2 Implementació . . . 3
- 1.3 Característiques Bàsiques . . . 3
- 1.4 Estructura del Manual . . . 4

2 Us de YAYA **5**

3 Comportament Bàsic **7**
- 3.1 Frames . . . 7
  - 3.1.1 Classes . . . 7
  - 3.1.2 Primitives . . . 7
  - 3.1.3 Variables . . . 10
  - 3.1.4 Events . . . 10
- 3.2 Slots . . . 10
  - 3.2.1 Classes . . . 10
  - 3.2.2 Primitives . . . 10
  - 3.2.3 Variables . . . 12
- 3.3 Atributs . . . 12
  - 3.3.1 Classes . . . 12
  - 3.3.2 Primitives . . . 12
  - 3.3.3 Variables . . . 14
- 3.4 Valors . . . 14
  - 3.4.1 Primitives . . . 14
- 3.5 Sistema de Tipus . . . 16
- 3.6 Daemons . . . 17
- 3.7 Events i Missatges . . . 18
  - 3.7.1 Classes . . . 18
  - 3.7.2 Primitives . . . 18
- 3.8 Entrada/Sortida . . . 19
  - 3.8.1 Primitives . . . 19
  - 3.8.2 Events . . . 21

4 Relacions **22**
- 4.1 Transitivitat . . . 22
- 4.2 Herència . . . 23
- 4.3 Classes . . . 23
- 4.4 Variables . . . 24
- 4.5 Primitives . . . 24
- 4.6 Daemons . . . 26

5 Contextes **27**
5.1 Classes 28
5.2 Variables 28
5.3 Events 29
5.4 Primitives 29

6 Raonament Terminològic **32**
6.1 Cadenes Estructurals 32
6.2 Propietats Estructurals Complexes 33
6.3 Definició de Conceptes 34
6.3.1 Classes 34
6.3.2 Variables 35
6.3.3 Events 35
6.3.4 Primitives 35
6.4 Classificació de Conceptes i Instàncies 37
6.4.1 Events 37
6.4.2 Primitives 37

7 Descripcions i Representació Probabilística **40**
7.1 Classes 40
7.2 Events 41
7.3 Primitives 41

8 Classes de Frames **44**

9 Nucli de la Base de Coneixement **46**
9.1 Elements Bàsics 46
9.1.1 Slots 46
9.1.2 Relacions 47
9.1.3 Atributs 49
9.2 Sistema de Tipus 50

Capítol 1

Introducció

1.1 Motivació

La idea de crear un sistema de frames propi va sorgir després de trobar-me amb unes necessitats específiques que cap dels sistemes de frames que teniem a l'abast podia satisfer.

En primer lloc, havia de ser un sistema de frames com els típics, amb nodes, relacions, slots, atributs, valors, herència i contextes. Però a més, havia d'incorporar capacitats de *raonament terminològic* [6] i capacitats de *representació probabilística* de propietats per poder representar conceptes de forma difusa, a partir de les propietats que compleixen les seves instàncies [5].

Vem estar fent proves amb el sistema LOOM [3], que incorpora raonament terminológic. Però ja pel que respecta a raonament terminològic mateix, ens sobraven capacitats per una banda i ens en faltaven per una altra. A més, un element imprescindible és que resultés totalment flexible en la generació i modificació de la base de coneixement, aspecte en el qual vaig trobar certes mancances (per exemple resultava problemàtic esborrar conceptes). A més, encara ens faltava la part de la representació probabilística de propietats, que li hagués tingut que afegir com una capa per damunt del propi LOOM.

A més de tot això, em resultava interessant la idea de desenvolupar un sistema propi que jo mateix pugués modificar i extendre segons les meves conveniències.

1.2 Implementació

YAYA ha estat implementat en CLOS, el sistema d'objectes que proporciona Common LISP. La idea principal en fer-ho (a part de la comoditat d'utilitzar la orientació a objectes) ha estat definir un *sistema de frames modular*.

D'aquesta forma, tenim una sèrie de classes CLOS que ens defineixen una sèrie de comportaments per als frames (veure 8 a la pàgina 44). Aleshores, depenen de quina classe CLOS són instància els frames, tindran un comportament o un altre. Podem combinar en una mateixa base de coneixement (en una mateixa execució de YAYA) frames instanciats desde diferents classes.

D'altra banda, això ens permet extendre fàcilment el sistema de frames amb noves característiques.

1.3 Característiques Bàsiques

- Permet definir conceptes a partir de propietats estructurals complexes (emphraonament terminològic).
- Les descripcions utilitzades per a definir conceptes van més enllà del que Woods anomena *type 1 descriptions* a [8].

- *Classificador* de termes o conceptes: donada la seva definició estructural, el classifica automàticament a la jerarquia.
- Transitivitat de les relacions definible de forma declarativa.
- Tractament de relacions i slots com un node qualsevol, poguent prendre part en una jerarquia.
- Permet un sistema de tipus a la xarxa semàntica (restriccions de presència de slots i valors als nodes depenent del tipus de node, que a la vegada està determinat per la relació especial *instance*).
- *Contextes*, possibilitat de tenir diferents visions de la base de coneixement depenent del contexte on ens posem.
- Possibilitat de definir un concepte de forma probabilística, a partir de les propietats de les seves instàncies.

1.4 Estructura del Manual

El manual està dividit en quatre parts. En la primera, que conté els capítols del 2 al 5, s'explica el comportament comú a la majoria de sistemes de frames existents, en la segona part explico les característiques pròpies del raonament terminològic, en la tercera la representació probabilística; i la quarta, constituïda pels apendixs, consisteix en una descripció de les classes CLOS i els frames basics que ja estan definits.

A continuació tenim un breu comentari del contingut de cada capítol.

Capítol 2 explicació de com fer servir YAYA, amb un breu exemple de com carregar i començar a treballar amb el sistema.

Capítol 3 gestió de frames, slots, atributs i valors. Ús dels daemons existents. Sistema de missatges que ens permet veure en pantalla de forma automàtica les modificacions que es van produïnt a la base de coneixement.

Capítol 4 Tot el referent a relacions. Definició de transitivitats i herència.

Capítol 5 Primitives per a definir i gestionar contextes. Visibilitat de frames segons contextes accessibles i transitivitat de les relacions.

Capítol 6 Element bàsic: la propietat estructural. Capacitat expressiva de les propietats estructurals. Definicions estructurals de conceptes. Classificació de conceptes i d'instàncies.

Capítol 7 Representació probabilística de conceptes. Descripcions.

Capítol 8 Classes CLOS de tipus de frames definides a YAYA i els comportaments que incorporen cada una d'elles.

Capítol 9 Relacions i objectes bàsics de YAYA. Nucli del sistema de tipus. Relacions bàsiques de la quantificació de tipus en YAYA.

Capítol 2

Us de YAYA

De moment, YAYA ha estat implantat únicament sobre CLISP. En el futur també està previst implantar-lo sobre altres sistemes Common LISP.

Per a carregar el sistema hem de carregar el fitxer "load-yaya.lsp". Prèviament podem voler compilar el sistema; això ho fem amb el fitxer "compile-yaya.lsp". Un cop carregat el sistema podem decidir guardar la memòria del CLISP en un fitxer de memòria, de forma que en següents execucions de YAYA únicament haguem de carregar el volcat de memòria.

```
noname> clisp

> (load ''compile-yaya'')
;; Loading file /home/hobbit/prog/lisp/frames/compile-yaya.lsp ...
Compiling file /home/hobbit/prog/lisp/frames/trigger.lsp ...
...
;; Loading of file /home/hobbit/prog/lisp/frames/compile-yaya.lsp is finished.
T

> (quit)

noname> clisp

> (load ''yaya'')
;; Loading file /home/hobbit/prog/lisp/frames/yaya.lsp ...
;; Loading file /home/hobbit/prog/lisp/frames/trigger.fas ...
...
T

> (saveinitmem ''yaya.mem'')
> (quit)
```

Fins aquí ja hem generat el volcat de memòria amb YAYA; i ara ja podem executar CLISP donant-li com a memòria inicial *yaya.mem*.

Un cop carregat yaya, podem començar a treballar usant les primitives que ens proporciona. De totes maneres hem de saber que:

- En un primer moment ens situem en el package "USER", desde el qual és accessible el package "FRAMES" (amb alias "FR") que és on està implementada la interfície cap al sistema de frames. Això ens permetrà treballar amb els frames a nivell de sìmbols; és a dir, tractant directament amb els noms dels frames.

- Podem treballar a nivell objecte CLOS si fem servir el package "FRAMES-KERNEL" (amb alias "FRK"). Els dos modes de treball són incompatibles doncs defineixen primitives amb el mateix nom, cosa per la qual no podem usar els dos packages a la vegada. Però si podem utilitzar les seves primitives si únicament tenim accessible un dels dos packages.
- En quant a contextes, just després de carregar yaya estem en el contexte *user*, desde el qual veiem *kernel* i *system-type*.

Capítol 3

Comportament Bàsic

3.1 Frames

3.1.1 Classes

frame

Packages: frames-kernel

Pares:

Slots:

Name Nom del frame

Plist El contingut del frame; una llista de parelles *(slot contingut-slot)*. A la vegada *contingut-slot* és una llista de parelles *(atribut contingut-atribut)*. *contingut-atribut* és una llista de valors.

Descripció: Classe bàsica que defineix el comportament bàsic de qualsevol frame. Tot frame ha de ser instància d'aquesta classe o d'una classe derivada d'aquesta.

3.1.2 Primitives

Link entre frames i noms

Cada frame és un objecte CLOS que a la vegada té un nom. Podem treballar directament a nivell d'objecte CLOS fent servir les primitives del package LISP *frames-kernel*; o bé podem treballar amb els noms dels frames. Per a poder fer això d'una forma còmoda, existeix una capa d'interfície proporcionada pel package *frames* que defineix les mateixes primitives que estan definides a *frames-kernel*, però treballa amb noms. El que fa aquesta capa és obtenir els objectes CLOS a partir dels noms i cridar les primitives de *frames-kernel*.

El link entre noms i objectes CLOS ha estat implementat amb una taula de hash i tres primitives bàsiques que hi accedeixen.

reference-frame(frame)

Packages: frames-kernel

Parametres:

frame l'objecte CLOS corresponent al frame.

Valor de retorn: El nom del frame.

Descripció: Instala un frame a la taula de hash. El nom del frame pot ser qualsevol objecte LISP i s'obté del slot *name* del frame.

find-frame(name)

Packages: frames-kernel

Parametres:

name nom del frame.

Valor de retorn: El frame associat al nom.

dereference-frame(frame)

Packages: frames-kernel

Parametres:

frame El frame que el.liminem de la taula de hash.

Valor de retorn: T si el frame estava a la taula de hash i NIL en cas contrari.

Descripció: El.limina un frame de la taula de hash que relaciona els noms amb els objectes CLOS.

Creació i Esborrat de Frames

create-frame(name &optional frame-class)

Packages: frames-kernel

Parametres:

name El nom del frame que volem crear.

frame-class La classe CLOS de la que volem que el frame sigui instància.

Valor de retorn: El frame ja creat.

Descripció: Crea un frame de nom *name*. Fa servir el macro *define-frame* per a crear-lo. El frame es crea com instància de la classe *frame-class*. En cas de que aquest paràmetre no s'especifiqui, es fa servir la classe referenciada per la variable **default-frame-class**.

define-frame(name &optional direct-slots frame-class &key (value-node t))

Packages: frames-kernel

Parametres:

name El nom del frame

direct-slots Slots dels frames, és una llista d'elements on cada element té un dels següents formats:

$(slot value_1 value_2 ... value_n (attr_1 v_{11} v_{12} ... v_{1n})(attr_2 v_{21} v_{22} ... v_{2n})...)$ Crea un slot que té com a nom *slot* i hi fica els valors corresponents a $value_1$, $value_2$, fins a $value_n$. També crea els atributs $attr_1$, $attr_2$, etc i hi fica els valors corresponents.
Per a aquest format tenim la restricció que $value_1$, $value_2$, fins a $value_n$ han de ser atòmics.

$(slot (value_1 value_2 ... value_n)(attr_1 v_{11} v_{12} ... v_{1n})(attr_2 v_{21} v_{22} ... v_{2n})...)$ Sintaxi alternativa d'una especificació de slot. Si no podem usar la sintaxi anterior per que els valors de l'atribut **value-attribute** no són atòmics, sempre podrem usar aquesta.

frame-class La classe CLOS de la qual volem instanciar el frame.

Valor de retorn: El nom del frame

Descripció: Macro que crea un frame amb slots, atributs i valors associats. Envia l'event *create-frame*. En el cas de que el frame ja estigui creat, l'únic que fa és definir els slots, atributs i valors especificats.

Slots, atributs i valors s'especifiquen aquí pel seu nom i no pas pel frame en si (l'objecte CLOS). Per això, s'ha d'anar en compte i tots els slots i atributs han d'haver estat creats prèviament sempre i quan siguin instàncies de classes de slots i atributs especials. En cas contrari el macro crearà els slots i atributs corresponents, però com a instància de les classes bàsiques: *frame-slot* i *frame-attribute*.

Els valors poden ser objectes LISP comuns o bé referir-se a algun frame de la base de coneixement. En aquest darer cas, també han d'haver estat creats prèviament. En cas contrari, no es produirà cap error però es prendran els valors com a objectes LISP normals (és a dir, accedirem al nom del frame i no al frame).

delete-frame(frame)

Packages: frames-kernel

Parametres:

frame El frame que volem esborrar.

Valor de retorn: El frame que hem esborrat.

Descripció: Esborra un frame de la base de coneixement. L'esborrat es fa valor a valor, de forma que s'elimini també la referència al frame que es fa en altres frames. Per això, fa ús de la funció *delete-slot*. Envia un event amb el mateix nom de la funció.

Varis

framep(object)

Packages: frames-kernel

Parametres:

object L'objecte que volem comprovar si és un frame.

Valor de retorn: t si l'objecte és un frame i nil en cas contrari.

mapframes(fnc)

Packages: frames-kernel

Parametres:

fnc Una funció de dos paràmetres que s'executarà per a tots els frames de la base de coneixement. Els paràmetres de la funció es corresponen al nom del frame i al frame mateix.

Valor de retorn: Retorna nil

Descripció: Aplica la funció que reb com a paràmetre a tots els frames de la base de coneixement.

3.1.3 Variables

default-frame-class

Packages: frames-kernel

Valor per defecte: hdf[1]

Descripció: El tipus de frame que crea *define-frame* per defecte

3.1.4 Events

create-frame

Packages: frames-kernel

Valor per defecte: activat

Descripció: Treu un missatge per pantalla cada cop que es crea un frame.

delete-frame

Packages: frames-kernel

Valor per defecte: activat

Descripció: Treu un missatge per pantalla cada cop que s'esborra un frame.

3.2 Slots

3.2.1 Classes

frame-slot

Packages: frames-kernel

Pares: frame

Descripció: Classe derivada de frame per als frames que també són slots. Tots els frames que siguin slots hauran de ser instància d'aquesta classe o d'una classe derivada.

3.2.2 Primitives

create-slot(frame slot)

Packages: frames-kernel

Parametres:

frame El frame en el qual volem crear un slot.

slot El slot que volem crear (ha de ser una instància de *frame-slot*.

Valor de retorn: El slot.

[1] Veure capítol 8.

Descripció: Crea un slot en un frame. Usa la funció *create-default-attributes* per crear els atributs per defecte per a aquest frame i slot. Si el slot ja existia no el crea, però sempre crida a la funció per crear els atributs per defecte.

delete-slot(frame slot)

Packages: frames-kernel

Parametres:

frame El frame en el qual volem esborrar un slot.

slot El slot que volem esborrar.

Valor de retorn: El slot.

Descripció: Esborra un slot d'un frame. En el cas de que el slot no estigués present en el frame, no fa res. Usa la funció *delete-attribute* per esborrar tots els atributs.

get-local-slots(frame &optional keyword)

Packages: frames-kernel

Parametres:

frame El frame del qual volem obtindre els slots.

keyword Paràmetre opcional que ens indica quins slots ens interessen. Pot ser *:explicit*, *:default* o nil.

Valor de retorn: Retorna una llista amb els slots locals del frame. Si *keyword* és *:explicit* ens retorna els slots presents en la definició del frame. Si és *:default* ens retorna aquells slots que estan per defecte, els que estan en la definició del tipus (marcat per la relació *is-a*. I si no s'especifica, els retorna tots.

local-slotp(frame slot &optional keyword)

Packages: frames-kernel

Parametres:

frame El frame.

slot L'slot del qual volem comprovar la presència.

keyword Ens indica el tipus de slots que volem tindre en compte (veure *get-local-slots*).

Valor de retorn: t si el slot està en el frame i nil si no hi és (sempre tenint en compte el valor de *keyword*).

get-slots(frame &optional keyword)

Descripció: Funció idèntica a *get-local-slots* però amb herència. Té en compte els slots del frame i els dels seus antecessors. Crida a la funció *frame-precedence-list* que li retorna tots els antecessors del frame.

slotp(frame slot &optional keyword)

Descripció: Idèntica a *local-slotp* però amb herència (veure *get-slots*.

3.2.3 Variables

conceptualization-link

Packages: frames-kernel

Valor per defecte: *instance*

Descripció: Referència al frame que correspon al link entre les instàncies i les classes de les que són instància, típicament anomenada *instance*.

Nota: Per ús intern

instantiation-link

Packages: frames-kernel

Valor per defecte: *instance+inv*

Descripció: Referència al frame que correspon al link entre les classes i les seves instàncies, típicament *instance+inv*.

Nota: Per ús intern

3.3 Atributs

3.3.1 Classes

frame-attribute

Packages: frames-kernel

Pares: frame

Descripció: Classe derivada de frame per als frames que també són atributs. Tots els frames que siguin a la vegada atributs hauran de ser instància d'aquesta classe o d'una classe derivada.

3.3.2 Primitives

attributep(frame slot attribute)

Descripció: Aquesta funció és idèntica a *local-attributep* però amb herència.

create-attribute(frame slot attribute)

Packages: frames-kernel

Parametres:

frame El frame corresponent.

slot El slot en el qual volem crear l'atribut.

attribute L'atribut que volem crear.

Valor de retorn: L'atribut.

Descripció: Crea l'atribut en el slot *slot* corresponent al frame *frame*. En el cas de que el slot no existeixi en el frame, el crea.

create-default-attributes(frame slot)

Packages: frames-kernel

Parametres:

frame El frame en qüestió.

slot El slot del qual volem crear els atributs per defecte.

Valor de retorn: Una llista amb els atributs creats.

Descripció: Mètode que crea els atributs que un slot d'un tipus determinat en un frame d'un tipus determinat té sempre per defecte. El mètode bàsic està definit per als tipus *frame* i *frame-slot*; i crea el atribut referenciat per la variable **value-attribute**, que representa l'atribut que conté el valor del slot. Aquest mètode és cridat al crear un slot, de forma que immediatament després de crear un slot, aquest ja tindrà els atributs bàsics.

delete-attribute(frame slot attribute)

Packages: frames-kernel

Parametres:

frame El frame en el qual volem esborrar l'atribut.

slot El slot del que volem esborrar l'atribut.

attribute L'atribut que volem esborrar.

Valor de retorn: L'atribut.

Descripció: Esborra un atribut d'un slot d'un frame. Crida al mètode *delete-any-attribute-values* per tal d'esborrar tots els valors de la'tribut. En el cas de que el slot no estigui present en el frame o l'atribut no estigui en el slot, no fa res.

get-attributes(frame slot)

Descripció: Idèntica a *get-local-attributes* però amb herència.

get-local-attributes(frame slot)

Packages: frames-kernel

Parametres:

frame El frame corresponent.

slot El slot del qual volem obtenir els atributs.

Valor de retorn: La llista d'atributs presents en el slot *slot* del frame *frame*. Si el slot no està en el frame, el mètode retorna nil.

local-attributep(frame slot attribute)

Packages: frames-kernel

Parametres:

frame El frame en el qual volem comprovar si hi és l'atribut.

slot L'slot corresponent.

attribute L'atribut del qual volem comprovar la seva presència. Ha de ser una instància de *frame-attribute*.

Valor de retorn: t si l'atribut hi és i nil en cas contrari. Aquesta funció no fa herència, es restringeix a la definició local del frame.

3.3.3 Variables

value-attribute

Packages: frames-kernel

Valor per defecte: *value*

Descripció: Referència al frame que correspon a l'atribut *valor* d'un slot.

Nota: Per ús intern

3.4 Valors

3.4.1 Primitives

add-attribute-value(frame slot attribute value)

Packages: frames-kernel

Parametres:

frame, slot i attribute Ens diuen on volem afegir el valor.

value El valor que volem afegir.

Valor de retorn: El valor afegit, o nil en cas de que no l'hagi afegit.

Descripció: Afegeix un valor a un atribut d'un slot d'un frame. El valor únicament s'afegeix als valors ja existents a l'atribut en el cas de que no hi sigui prèviament. En cas de que el slot i/o el atribut no existeixin, els crea. Crida *add-attribute-values* per a fer l'operació.

add-attribute-values(frame slot attribute values)

Packages: frames-kernel

Parametres:

frame, slot i attribute Ens diuen on volem afegir el valor.

values La llista de valors que volem afegir.

Valor de retorn: Una llista amb els valors que realment s'han afegit a l'atribut.

Descripció: Idèntica a *add-attribute-value* però per una llista de valors. Executa els *daemons put-before-daemons* i *put-after-daemons* abans i després d'afegir els valors corresponents.

delete-attribute-value(frame slot attribute value)

Packages: frames-kernel

Parametres:

frame, slot i attribute Ens diuen d'on volem esborrar el valor.

value El valor que volem esborrar.

Valor de retorn: El valor esborrat o nil si no ha pogut esborrar-lo (perque no hi era).

Descripció: Esborra un valor d'un atribut. Si el valor no hi era, o el atribut o el slot no hi eren, no fa res i retorna nil. Executa els *daemons delete-before-daemons* i *delete-after-daemons* abans i després d'esborrar el valor (i sempre que l'esborri).

delete-any-attribute-values(frame slot attribute)

Packages: frames-kernel

Parametres:

frame, slot i attribute Ens diuen d'on volem esborrar els valors.

Valor de retorn: Una llista amb els valors esborrats. Nil si el atribut o el slot no hi són.

Descripció: Esborra tots els valors d'un atribut. Si l'atribut o el slot no hi són, no fa res. Executa els *daemons delete-before-daemons* i *delete-after-daemons* abans i després d'esborrar els valors.

get-attribute-value(frame slot attribute)

Descripció: Idèntic a *get-attribute-values* però retorna els valors com valors múltiples.

get-attribute-values(frame slot attribute)

Descripció: Idèntic a *get-direct-attribute-values* però amb herència.

get-direct-attribute-value(frame slot attribute)

Descripció: Mètode idèntic a *get-direct-attribute-values* (mètode al qual, a més, crida), amb la salvetat de que retorna els valors com a valors múltiples. Útil sobretot quan sabem que només hi ha un valor.

get-direct-attribute-values(frame slot attribute)

Packages: frames-kernel

Parametres:

frame, slot i attribute Ens diuen d'on volem agafar els valors.

Valor de retorn: Una llista amb els valors de l'atribut o nil si l'atribut o el slot no existeixen.

Descripció: Retorna els valors corresponents i executa els *daemons get-before-daemons* i *get-after-daemons* abans i després d'agafar els valors. En el cas de que *attribute* sigui **value-attribute**, la funció retorna també els valors per defecte del slot, especificats als frames que tenen per instància (donats pel link **conceptualization-link**) a *frame*.

put-attribute-value(frame slot attribute value)

Descripció: Idèntic a *put-attribute-values* (mètode al qual, a més, crida) però posa un únic valor a látribut en lloc d'una llista de valors.

put-attribute-values(frame slot attribute values)

Packages: frames-kernel

Parametres:

frame, slot i attribute Ens indiquen on volem posar els valors.

values La llista de valors que volem posar en l'atribut.

Valor de retorn: La llista de valors afegits al slot.

Descripció: Modifica el valor d'un atribut. Esborra tots els valors que contenia prèviament (cridant a *delete-any-attribute-values*) i posteriorment posa els valors indicats amb *add-attribute-values*. Evidentment, s'executen tan els *daemons* de borrat com els d'afegit.

A més d'aquests mètodes, i per tal de fer més comode el treball, s'han definit una sèrie de mètodes per accedir directament a l'atribut que conté el valor d'un slot. Aquests mètodes són equivalents als que acabem de veure amb l'afegit que tenen l'atribut **value-attribute** implícit. Els mètodes són els següents:

- *add-value*
- *add-values*
- *delete-value*
- *delete-any-values*
- *get-direct-value*
- *get-direct-values*
- *get-value*
- *get-values*
- *put-value*
- *put-values*

3.5 Sistema de Tipus

YAYA incorpora un control molt senzill de tipus que permet validar la informació present en els frames. Les comprovacions que pot fer són de domini de slots i valors. La validació es fa a partir de dues primitives: *valid-slot* i *valid-value*, que comproven la correctesa de la presència d'un slot en un frame i la d'un valor en un slot respectivament.

La comprovació de tipus no està integrada en les classes bàsiques de frames. Tampoc s'han creat classes derivades per les que les primitives d'afegir slots i valors; però es podria fer molt fàcilment.

Les comprovacions es basen en el mecanisme d'instanciació i la definició de dominis i rangs per als slots. Aquesta definició es fa en el frame corresponent al slot amb els slots *domain* i *range*. El que especifiquem mitjançant aquests slots és els tipus de frame que poden tenir al slot com a slot i els tipus de frame que pot tenir el slot com a valor.

Aleshores, per a fer les comprovacions sol hem de mirar que el frame on es troba el slot sigui instància (mitjançant el link **conceptualization-link**) d'algun dels tipus (o conceptes) que el slot té com a domini. (Per a buscar els conceptes dels quals és instància, sempre es fa servir la funció *get-relatives*, que fa ús de la transitivitat (veure secció 4.1).

En quant als valors, les comprovacions són semblants: els valors han de ser instància d'algun dels conceptes que el slot té com a rang.

valid-slot(frame slot)

Packages: frames-kernel

Parametres:

frame El frame en el que volem fer la comprovació.

slot El slot que volem comprovar si és correcte.

Valor de retorn: t si la presència del slot és correcta i nil en cas contrari.

valid-value(frame slot value)

Packages: frames-kernel

Parametres:

frame El frame en el que volem fer la comprovació.

slot El slot en el que volem fer la comprovació.

value El valor que volem comprovar si és correcte.

Valor de retorn: t si la presència del valor és correcta i nil en cas contrari. La correctesa de la presència del valor implica també la correctesa de la presència del slot.

3.6 Daemons

Molts sistemes de frames incorporen daemons; és a dir, procediments que s'executen automàticament quan s'efectua alguna operació sobre la base de coneixement. Al estar definit YAYA en CLOS, els daemons es poden definir fàcilment amb *mètodes after i before* per a les primitives que nosaltres volguem. De totes maneres, sembla interessant poder agrupar situacions semblants en un sol daemon. Per aquest motiu YAYA incorpora mètodes específics per a definir daemons en sis tipus de situacions bàsiques per a valors.

Aquestes sis situacions (amb els mètodes corresponents) són:

- Abans d'agafar un valor (*get-before-daemons*).
- Després d'agafar un valor (*get-after-daemons*).
- Abans de posar un valor (*put-before-daemons*).
- Després de posar un valor (*put-after-daemons*).
- Abans d'esborrar un valor (*delete-before-daemons*).
- Després d'esborrar un valor (*delete-after-daemons*).

Amb aquestes sis situacions bàsiques i la seva combinació cobrim tot el que podem fer amb les primitives de la secció 3.4.1. El fet de definir mètodes especials per als daemons ens permet per exemple, que sempre que s'afegeixi un valor, sigui amb la primitiva que sigui, s'executi el mateix daemon.

Els dos mètodes corresponents a *get-* tenen tres paràmetres: el frame, el slot i l'atribut. Els altres quatre mètodes (corresponents a *put-* i *delete-* tenen un quart paràmetre corresponent al valor que es posa o s'esborra.

Tots sis mètodes han estat definits per a les classes bàsiques per al frame, slot i atribut (corresponents a *frame*, *frame-slot* i *frame-attribute*) de forma que no fan res. En el cas que necessitem que els daemons fagin alguna cosa haurem de derivar una classe (corresponent al frame, al slot o al atribut, segons ens interessi) i definir el mètode corresponent als daemons per aquella classe.

Per exemple, si volguéssim mantenir un comptador de les vegades que hem accedit als valors per a un tipus de frame determinat, podríem fer el següent:

En primer definir una classe derivada de *frame*

```
(defclass frame-amb-comptador (frame) ())
```

Un cop fet això, definim el mètode per als daemons que actualitza el comptador (tindríem un comptador per a cada tripleta *¡frame,slot,atributi*); i una funció per accedir als comptadors.

```
(let ((comptadors (make-hash-table)))

  (defmethod get-before-daemons :after ((f frame-amb-comptador) (s frame-slot)
                                        (a frame-attribute))
    (let* ((tripleta (cons f (cons s a)))
           (c (gethash tripleta comptadors)))
      (setf (gethash tripleta comptadors) (if c (1+ c) 1))))

  (defun get-access-counter (f s a)
    (let ((c (gethash tripleta comptadors))) (if c c 0))))
```

3.7 Events i Missatges

Per tal de poder saber com s'està modificant la base de coneixement en cada moment, s'han introduït els events. Cada cop que una primitiva envia un event, s'imprimeix un missatge informatiu per la pantalla. Els events poden estar activats o desactivats, de forma que podem escollir imprimir o no certs missatges informatius.

3.7.1 Classes

event

Packages: frames-kernel

Slots:

activation Ens diu si un event està activat o no.

comment El missatge informatiu que treu per pantalla.

parameters Els noms dels paràmetres que després imprimirà cada cop que s'envii l'event. Normalment els paràmetres es correspondran amb els paràmetres de la primitiva que envia l'event.

Descripció: Cada event tindrà el seu objecte event corresponent. Cada objecte té la informació suficient per saber si ha d'imprimir el missatge corresponent i per, si ho ha de fer, fer-ho.

3.7.2 Primitives

activate-event(event-name)

Packages: frames-kernel

Parametres:

event-name El nom de l'event que volem activar.

Valor de retorn: Retorna sempre t.

Descripció: Activa un event determinat.

deactivate-event(event-name)

Packages: frames-kernel

Parametres:

event-name El nom de l'event que volem desactivar.

Valor de retorn: Retorna sempre nil.

Descripció: Desactiva un event determinat.

define-event(name default comment &rest parameters)

Packages: frames-kernel

Parametres:

name El nom de l'event.

default Si és t, indica que l'event estarà activat desde el començament; i si és nil estarà desactivat.

comment El missatge que imprimeix.

parameters Una llista amb els noms dels paràmetres que s'imprimiran al imprimir el missatge.

Valor de retorn: L'event definit.

Descripció: Defineix un event al qual després podrem accedir fàcilment per el nom.

send-event(name &rest parameters)

Packages: frames-kernel

Parametres:

name El nom de l'event.

parameters Una llista amb els valors reals dels paràmetres. Al imprimir el missatge, es faran correspondre aquests valors amb els noms corresponents (que havíem definit a *define-event.*

Valor de retorn: L'event.

Descripció: Envia un event. El resultat és l'impressió d'un missatge per pantalla corresponent a l'event.

3.8 Entrada/Sortida

YAYA té un sistema bàsic d'entrada/sortida que permet imprimir frames per pantalla i guardar frames en disc. Les funcions d'escritura en disc graven els frames en ASCII, fent ús del macro *define-frame*; cosa que ens permetrà posteriorment carregar els fitxers generats com fitxers LISP estàndar.

3.8.1 Primitives

print-frame(name &optional (stream t))

Packages: frames

Parametres:

name El nom del frame que volem imprimir.

stream El stream en el qual el volem imprimir.

Valor de retorn: El nom del frame si el frame existeix i nil en cas contrari.

Descripció: Imprimeix un frame en un stream.

pp(name &optional (stream t))

Packages: frames-kernel

Parametres:

name El nom del frame que volem imprimir.

stream El stream en el qual el volem imprimir.

Valor de retorn: El nom del frame si el frame existeix i nil en cas contrari.

Descripció: Versió macro de *print-frame*.

save-kb(file)

Packages: frames-kernel

Parametres:

file Nom del fitxer en què volem grabar la base de coneixement.

Valor de retorn: t si els frames s'han grabat i nil si hi ha hagut problemes.

Descripció: Graba tots els frames de la base de coneixement en un fitxer.

save-taxonomy(file root-frame)

Packages: frames

Parametres:

file Nom del fitxer en què volem grabar la taxonomia.

root-frame El frame que és l'arrel de la taxonomia que volem grabar.

Valor de retorn: t si els frames s'han grabat i nil si hi ha hagut problemes.

Descripció: Graba una taxonomia sencera en un fitxer. La relació d'especialització entre l'arrel de la taxonomia i les subclasses ens ve determinada pel link **specialization-link**.

save-taxonomies(file root-frames)

Packages: frames

Parametres:

file Nom del fitxer en què volem grabar les taxonomies.

root-frames Els frames que són l'arrel de les taxonomies que volem grabar.

Valor de retorn: t si els frames s'han grabat i nil si hi ha hagut problemes.

Descripció: Idèntica a la funció *save-taxonomy* però per a vàries taxonomies.

save-frames(file frames)

Packages: frames-kernel

Parametres:

file Nom del fitxer en què volem grabar les taxonomies.

frames Els frames que volem grabar.

Valor de retorn: t si els frames s'han grabat i nil si hi ha hagut problemes.

Descripció: Graba un conjunt de frames en un fitxer.

save-frame(frame &optional (stream t))

Packages: frames-kernel

Parametres:

name El nom del frame que volem grabar.

stream El stream en el qual el volem grabar.

Valor de retorn: El nom del frame.

Descripció: Graba un frame en un stream. Al contrari de les altres funcions, que obrien i tancaven el fitxer en el qual grabaven, aquesta el reb ja obert com un stream.

Degut a les restriccions sobre valors comentades en l'explicació del macro *define-frame*, quan volem grabar un conjunt de frames no és aconsellable fer servir aquesta funció directament. Si el frame està relacionat amb altres frames que no s'han grabat prèviament, al llegir el fitxer grabat, els valors es prendran com a símbols i no com a frames. Per això és aconsellable fer servir les funcions que graben conjunts de frames i no fer servir mai aquesta directament.

Totes les funcions que graben conjunts de frames fan ús d'aquesta funció; però prèviament a especificar els slots de qualsevol frame, els han creat tots com a nodes de forma que quan els referenciem ja sabrem que són frames.

3.8.2 Events

save-frame

Packages: frames

Descripció: Event corresponent a la primitiva del mateix nom i que per tant s'envia cada cop que es grava un frame en un fitxer.

Capítol 4

Relacions

Les relacions són un tipus especial de slot que es caracteritzen perque els valors amb els que tracten són sempre frames, a diferència d'un slot qualsevol, que pot amagatzemar qualsevol tipus de valor.

Cada relació pot tenir una relació inversa, de forma que si un node està relacionat amb un altre mitjançant una relació, podem afirmar que el segon està relacionat amb el primer mitjançant la relació inversa. A més també podem definir transitivitats, de forma que per què dos nodes estiguin relacionats no és necessari que estiguin units per el link corresponent; sino que poden estar conectats indirectament a través d'una cadena de nodes i relacions.

Tot això fa que el comportament d'una relació sigui bastant diferenciat del d'un slot qualsevol. Aquesta diferència en el comportament es plasma en la creació d'una nova classe CLOS derivada de *frame-slot* i anomenada *relation*.

4.1 Transitivitat

La definició de la transitivitat està presa més o menys directament del *Knowledge Craft* [4]. El punt de partida bàsic és la conexió directa entre dos frames o nodes. A partir d'aquí, disposem d'una sèrie d'operadors que ens permeten anar combinant transitivitats més senzilles en més complexes.

Aquí tenim els elements bàsics a partir dels quals podem començar a construir transitivitats:

t Donat un node, els nodes relacionats amb ells seran tots aquells que estiguin conectats mitjançant alguna relació, sense importar quina.

nil Els nodes relacionats són el conjunt buit.

relation-name Donat un node, els nodes que estaran relacionats amb ell seran aquells que estiguin directament conectats per la relació donada.

(path relation-name) Anem a buscar la transitivitat corresponent a la relació.

I a continuació tenim els operadors amb els que podem construïr transitivitats cada cop més complexes:

t=(or $< t_1 >< t_2 > ... < t_n >$) El conjunt de nodes relacionats serà la unió dels nodes relacionats mitjançant qualsevol de les sub-transitivitats.

t=(list $< t_1 >< t_2 > ... < t_n >$) Donat un node inicial, direm que un altre node està relacionat amb ell mitjançant t si existeixen una sèrie de nodes intermitjos $n_1, n_2 ... n_(n-1)$ tals que el node inicial està relacionat amb n_1 mitjançant t_1, a la vegada n_1 està relacionat amb n_2 mitjançant t_2, i així successivament fins que $n_(n-1)$ està relacionat amb el node final mitjançant t_n.

t=(list* $< t_1 >< t_2 > ... < t_n >$) Igual que per a l'operador *list* però amb l'afegit que els nodes intermitjos també es consideren nodes relacionats.

t=(repeat t' inf sup) Aquí direm que un node està relacionat amb el node inicial si hi ha m nodes intermitjos (on m està entre *inf-1* i *sup-1*) de forma que el node inicial està relacionat usant t' amb n_1, a la vegada n_1 ho està amb n_2 usant t', i així fins que arribem a que n_m ho està amb el node final també usant t'.

4.2 Herència

L'herència ens serveix principalment per a les primitives del capítol 3. Es fa herència dels conceptes més generals als més específics.

El mètode bàsic que criden totes les primitives que fan herència és *frame-precedence-list*. Aquest mètode, donat un concepte, retorna una llista amb tots els conceptes més generals que ell. L'ordre en el que apareixen els conceptes a la llista és relevant, doncs els conceptes estan ordenats de més específics a més generals. És a dir, si un concepte a apareix abans que un altre concepte b, sabem que b és més general que a o bé que els conceptes no són comparables (el que sabem segur és que a no és més general que b).

En quan a la herència múltiple, evidentment es pot fer; però YAYA no disposa de cap mecanisme per a fer que els conceptes no comparables apareguin en un ordre determinat dins de la *frame-precedence-list*. En el cas de que necessitéssim algun mecanisme d'aquest estil sempre es podria crear una classe de frame derivada i redefinir al mètode *frame-precedence-list* per a aquesta classe.

4.3 Classes

relation

Packages: frames-kernel

Pares: frame-slot

Descripció: Incorpora al comportament de *frame-slot* el tractament per a les relacions inverses i certs mecanismes automàtics de generació de coneixement sobre la transitivitat de les relacions.

inverse-linked-attribute

Packages: frames-kernel

Pares: frame-attribute

Descripció: No tots els atributs en una relació tenen per què actualitzar-se automàticament quan s'actualitzen els atributs de la relació inversa.

Per exemple, el atribut corresponent al valor del slot, clarament requereix actualització automàtica del valor. Però en canvi, no tindria cap sentit fer una actualització automàtica del valor per a un attribute *domain* que ens especifiqués el domini del valor.

Aquesta classe fa la distinció entre els que s'actualitzen automàticament i els que no. Afegeix al comportament de *attribute* l'automatització del lligam corresponent a les relacions inverses.

4.4 Variables

inverse-link

Packages: frames-kernel

Valor per defecte: La relació *inverse* del nucli de la base de coneixement.

Descripció: Referència la relació que ens dóna la relació inversa.

generalization-link

Packages: frames-kernel

Valor per defecte: La relació *is-a* del nucli de la base de coneixement.

Descripció: Referéncia la relació amb la qual podem accedir a conceptes més generals a partir d'un específic.

specialization-link

Packages: frames-kernel

Valor per defecte: La relació *is-a+inv* del nucli de la base de coneixement.

Descripció: Referéncia la relació amb la qual podem accedir a conceptes més específics a partir d'un de més general.

transitivity-slot

Packages: frames-kernel

Valor per defecte: El slot *transitivity* del nucli de la base de coneixement.

Descripció: El slot que conté l'especificació de la transitivitat d'una relació.

4.5 Primitives

frame-precedence-list(frame)

Packages: frames-kernel

Parametres:

frame El frame del que volem conèixer els séus antecessors.

Valor de retorn: Una llista amb els antecessors del frame. Els antecessors de *frame* són els nodes relacionats amb ell mitjançant **generalization-link** (fem servir la transitivitat d'aquesta relació). La llista d'antecessors està ordenada de concepte més específic a concepte més general.

get-local-relations(frame)

Packages: frames-kernel

Parametres:

frame El frame sobre el qual estem consultant.

Valor de retorn: Una llista amb tots els slots locals del frame que a la vegada són relacions.

get-relations(frame)

Packages: frames-kernel

Parametres:

frame El frame sobre el qual estem consultant.

Valor de retorn: Una llista amb tots els slots del frame que a la vegada són relacions. Aquesta funció crida a *get-slots* i per tant fa herència.

get-relatives(frame relation)

Packages: frames-kernel

Parametres:

frame El frame del que volem trobar els frames relacionats.

relation La relació amb la que busquem els nodes relacionats.

Valor de retorn: Una llista amb tots els frames relacionats amb el frame *frame* mitjançant *relation*.

Descripció: Si la relació té una transitivitat definida (accessible mitjançant el slot **transitivity-slot**) la funció fa servir aquesta transitivitat.

En cas de que no la tingui, s'agafa com a transitivitat el nom de la relació. Això vol dir que en aquest cas la funció únicament retornarà els frames relacionats directament amb *frame* mitjançant la relació.

instancep(frame)

Packages: frames-kernel

Parametres:

frame El frame sobre el que estem consultant.

Valor de retorn: t si *frame* és instància directa d'algun frame (mira si està conectat amb algun altre frame amb el link **conceptualization-link**).

relatedp(frame-1 relation frame-2)

Packages: frames-kernel

Parametres:

frame-1 El frame inicial per fer la cerca.

relation La relació amb la que fem la cerca.

frame-1 El frame final de la cerca.

Valor de retorn: t si *frame-1* està relacionat amb *frame-2* mitjançant *relation*.

relationp(object)

Packages: frames-kernel

Parametres:

object L'objecte que volem saber si és una relació.

Valor de retorn: t si l'objecte és una relació i nil en cas contrari.

unlink(frame relation inverse)

Packages: frames-kernel

Parametres:

frame El frame que volem deslligar de la xarxa (o d'un troç de xarxa.

relation La relació de la qual volem deslligar el node.

inverse La relació que actua com inversa. Tot i que en la majoria de casos serà la inversa de *relation*, pot ser interessant que sigui una relació diferent a la inversa de *relation*.

Valor de retorn: Dos valors com a valors múltiples: Una llista de nodes conectats al node mitjançant *relation* i una altra llista de nodes amb els frames conectatsa *frame* mitjançant la relació inversa.

Descripció: Desconecta els nodes que estan conectats a *frame* mitjançant *relation* i *inverse* i els conecta entre ells.

Útil per eliminar nodes de forma transparent. Per exemple per eliminar-los d'una jerarquia sense que es perdi la relació entre subconceptes séus i superconceptes séus.

4.6 Daemons

L'especialització dels daemons *put-after-daemons* i *delete-after-daemons* per a la classe *inverse-linked-attribute* es el que ens permet gestionar automàticament l'actualització de les relacions inverses.

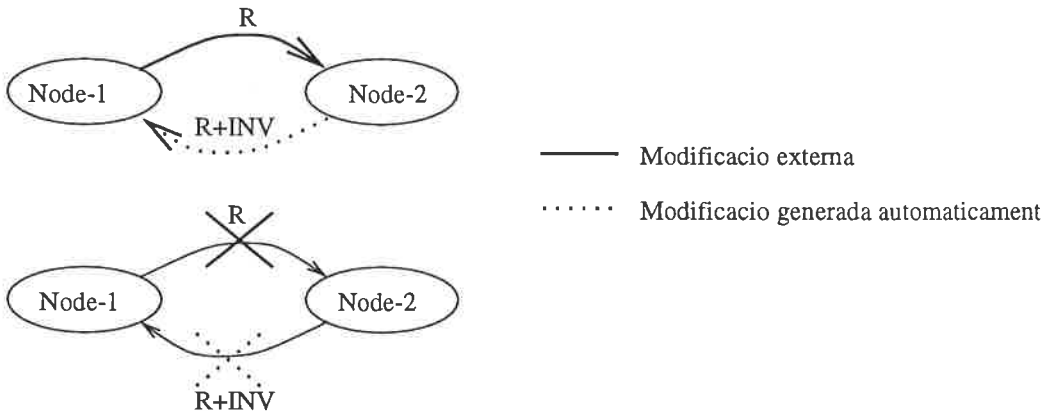


Figura 4.1: Actualització automàtica de relacions inverses feta pels daemons.

A la figura 4.1 podem veure com al crear un link de relació entre dos nodes, es crea automàticament el link invers amb el sentit del link canviat. I al eliminar el link, el link invers també s'esborra.

Capítol 5

Contextes

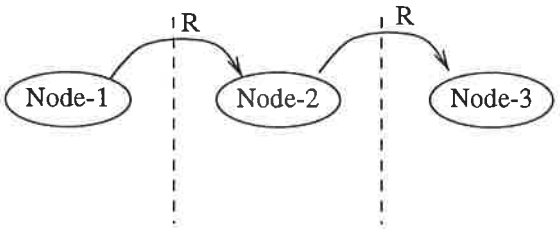
Els contextes, presents en bastants sistemes de frames, ens permeten tenir diferents nivells de visibilitat sobre el sistema de frames.

Disposem de dos tipus de frames: els incontextuals (instanciats directament de *frame* o una classe derivada que no sigui derivada de *context-frame*) i els contextuals (instanciats de *context-frame* o una classe derivada). Els primers són sempre visibles, mentre que els segons tenen un contexte associat i seran visibles sempre que el contexte associat sigui accessible desde el contexte actual.

Per a donar un contexte a un frame al definir-lo, ho podem fer bé mitjançant el keyword *:context* del macro *define-frame*; o bé donant-lo implícitament en el nom del frame. El nom complet del frame consisteix en el nom que nosaltres li haguem donat al frame, precedit pel nom del contexte en el qual està (o velem que estigui) el frame (ambós noms estan separats pel caràcter '.').

Si estem treballant amb la interfície (package "FRAMES"), sempre que volguem referenciar un frame que no estigui en el contexte actual o que no sigui incontextual, ho hem de fer amb el seu nom complet: *nom-contexte.nom-frame*. Si el frame està en el contexte actual, podem escollir entre usar únicament el nom del frame o usar el nom complet.

La visibilitat o no visibilitat dels frames és completament transparent a lúsuari, que només ha de definir els contextes, posar cada frame en el que li interessi i situar-se en el contexte que vulgui en cada moment.




```
> (set-context 'A)
> (use-contexts '(B C))
> (get-value 'node-1 'r)
node-2

> (unuse-contexts '(B))
> (get-value 'node-1 'r)
node-3
```

Figura 5.1: Transparència de la visibilitat i no visibilitat dels frames en els contextes.

En la figura 5.1 podem veure com al cridar qualsevol primitiva de YAYA, es té en compte la visibilitat dels frames. També es pot veure com, en el cas de les relacions, els frames no visibles s'*amaguen*, de forma que si agafem el valor d'una relació per a un frame i aquest està conectat amb un frame no visible, la primitiva no retornarà necessàriament nil sino que buscarà el primer frame visible relacionat amb el frame inicial. Això pot ser bastant útil de cara a amagar de forma totalment transparent troços de jerarquia, tindre troços de jerarquia independents, etc.

La visibilitat de contextes té una estructura jeràrquica; o d'una altra manera, compleix la propietat transitiva. És a dir, quan un contexte és visible per un altre contexte, també ho seran

tots els que són visibles per el primer.

Inicialment hi ha 3 contextes: *kernel*, *type-system* i *user*. El primer és visible desde els altres dos i conté els slots, relacions i atributs bàsics per començar a treballar. *type-system* és visible desde *user* i conté el sistema de tipus que modelitza el mateix sistema de representació de YAYA. *user* és un contexte buit que YAYA crea per a poder treballar en algun contexte al començar sense haver-lo de crear i definir les visibilitats.

5.1 Classes

context-frame

Packages: frames-kernel

Pares: frame

Slots:

context El contexte en el que es troba el frame.

Descripció: Frame que incorpora la possibilitat de treballar amb contextes. El frame serà visible o no segons el contexte al qual pertanyi sigui accessible o no. En una mateixa base de coneixement podem barrejar frames que siguin instància d'aquesta classe (o alguna derivada) i frames que no ho siguin. Els que no ho siguin seran sempre visibles.

context-frame-slot

Packages: frames-kernel

Pares: context-frame, frame-slot

Descripció: Incorpora la funcionalitat dels contextes a la classe bàsica *frame-slot*.

context-frame-attribute

Packages: frames-kernel

Pares: context-frame, frame-attribute

Descripció: Incorpora la funcionalitat dels contextes a la classe bàsica *frame-attribute*.

context-relation

Packages: frames-kernel

Pares: context-frame, relation

Descripció: Incorpora la funcionalitat dels contextes a la classe bàsica *relation*.

5.2 Variables

context

Valor per defecte: *user*

Descripció: El contexte actual. Inicialment el contexte actual és *user*, el contexte de treball que YAYA crea per defecte.

special-context-relations

Valor per defecte: (*is-a is-a+inv inverse transitivity*)

Descripció: Una llista de relacions per les que la no visibilitat de nodes adjacents no es pot convertir en un *get-relatives* ja que això ens portaria a un bucle infinit.

5.3 Events

create-context

Descripció: Event que s'envia per la primitiva del mateix nom cada cop que es crea un contexte.

delete-context

Descripció: Event que s'envia per la primitiva del mateix nom cada cop que s'esborra un contexte.

5.4 Primitives

define-context(context visibles)

Packages: frames-kernel

Parametres:

context El nom del contexte a definir.

visibles Una llista amb els contextes visibles desde *context*.

Valor de retorn: Una llista amb els contextes visibles desde *context*.

Descripció: Defineix un contexte. Envia l'event de creació del contexte.

remove-context(context)

Packages: frames-kernel

Parametres:

context El contexte a eliminar.

Valor de retorn: La llista de contextes visibles desde *context*.

Descripció: Elimina un contexte i envia l'event corresponent. No fa res amb els frames correponents al contexte; així doncs, això és tasca de l'usuari.

push-context(context)

Packages: frames-kernel

Parametres:

context El nom del contexte a apilar.

Valor de retorn: context

Descripció: YAYA dóna facilitats per mantenir una pila de contextes, de forma que quan nosaltres volguem veure la base de coneixement des d'un nou contexte, no necessitem saber en quin contexte estem si després l'hem de restaurar. L'únic que necessitem fer és un *push-context* i un *pop-context*. (veure *pop-context*.)

pop-context()

Packages: frames-kernel

Valor de retorn: El nou contexte actual.

Descripció: Fa un pop de la pila de contextes guardats. Modifica la variable **context** ficant-hi el cim de la pila. En el cas de que la pila estigués buida, **context** es fica a nil. (veure *push-context*.)

*use-contexts(visibles &key (from *context*))*

Packages: frames-kernel

Parametres:

visibles La llista de contextes que volem afegir als contextes visibles.

from El contexte desde el qual volem que siguin visibles la llista anterior. Per defecte és el contexte actual.

Valor de retorn: Una llista amb els contextes visibles desde from.

Descripció: Afegeix *visibles* a la llista de contextes visibles desde *from*.

*unuse-contexts(visibles &key (from *context*))*

Packages: frames-kernel

Parametres:

visibles La llista de contextes que eren visibles i ja no volem que ho siguin.

from El contexte desde el qual volem que ja no siguin visibles la llista anterior. Per defecte és el contexte actual.

Valor de retorn: Una llista amb els contextes visibles desde from.

Descripció: Elimina un conjunt de contextes que eren visibles de la llista de contextes visibles de *from*. Només podem eliminar d'aquesta llista els contextes que són directament visibles desde *from* i no els que són visibles per transitivitat.

*visible-contexts(&key (from *context*))*

Packages: frames-kernel

Parametres:

from El contexte del qual volem saber els accessibles.

Valor de retorn: Una llista amb tots els contextes que són visibles desde *from*.

visiblep(frame &key (from context))

Packages: frames-kernel

Parametres:

frame El frame que volem saber si és visible.

from El contexte desde el que volem saber si el frame és visible. Per defecte és **context**.

Valor de retorn: t si el frame és visible desde el contexte *from*, i nil en cas contrari.

set-context(context)

Packages: frames-kernel

Parametres:

context El nou contexte actual.

Valor de retorn: El contexte actual, abans de cridar a *set-context*.

Descripció: Modifica el contexte actual (**context**).

filter-visible-frames(frame &key (from context))

Packages: frames-kernel

Parametres:

frames El frames dels que volem saber els visibles.

from El contexte desde el que volem *filtrar* els frames. Per defecte és **context**.

Valor de retorn: Una llista amb el subconjunt de *frames* que són vislbes desde *from*.

Capítol 6

Raonament Terminològic

El raonament terminològic (veure [2, 6]), també anomenat *lògica descriptiva*, consisteix bàsicament en, a més de declarar conceptes com fem amb qualsevol sistema de frames o xarxa semàntica, definir-los amb algun llenguatge amb el que poguem raonar (típicament una restricció del càlcul de predicats, on els predicats bàsics funcionen de lligam entre el sistema de frames i el llenguatge de definició de termes (que a la vegada són els frames de la xarxa)).

El fet de poder raonar amb les definicions ens permet principalment classificar els conceptes de forma automàtica en la jerarquia, així com classificar les instàncies sota el concepte adequat.

6.1 Cadenes Estructurals

YAYA incorpora eines de *raonament terminològic*. A nosaltres únicament ens interessa el raonament que es pugui fer a partir de tipus i relacions entre nodes. Per això, tenim un únic element bàsic: la *cadena estructural conceptualitzada*. Altres sistemes, com el LOOM [3] incorporen també altres elements bàsics que li permeten fer comprovacions de cardinalitat o inclús fer comprovacions sobre els valors dels slots.

Abans de definir la cadena estructural conceptualitzada, veiem primer que és una cadena estructural. Anomenem *cadena estructural* a una llista ordenada de nodes, cadascun dels quals està relacionat amb el següent per una relació determinada. Per exemple a la figura 6.1 tenim un exemple d'un troç de xarxa semàntica i de tres cadenes estructurals contingudes en ella.

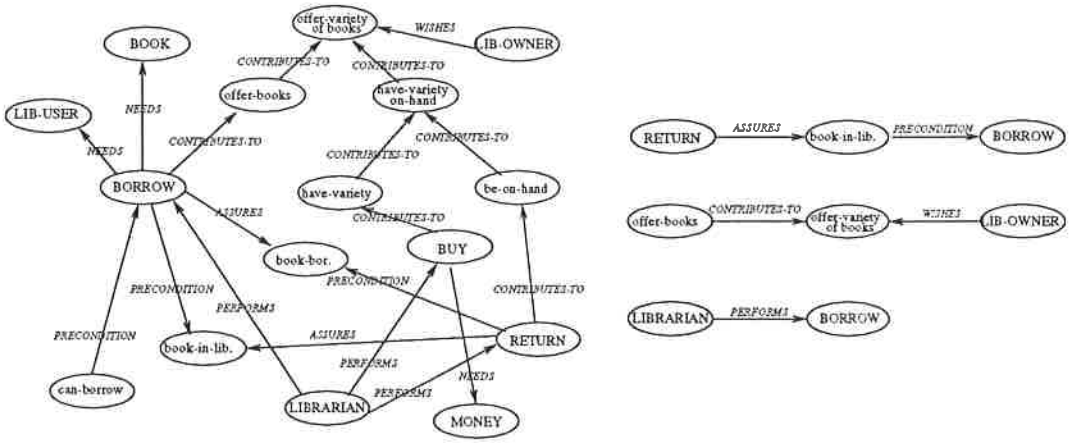


Figura 6.1: Una xarxa semàntica i tres cadenes estructurals contingudes en ella.

Una *cadena estructural conceptualitzada* és una cadena estructural on els objectes originals de la cadena estructural han estat substituïts amb els conceptes dels que són instància. A la figura

6.2 tenim un exemple de conceptualització de cadenes estructurals.

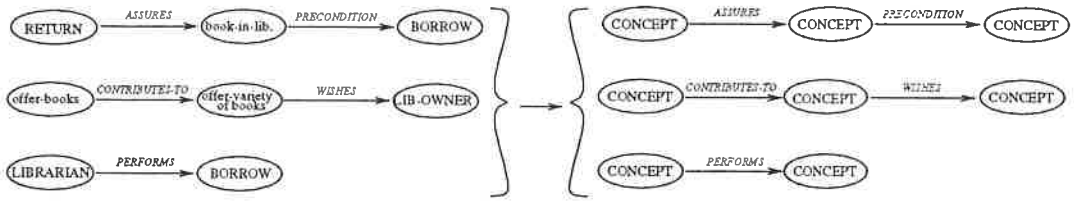


Figura 6.2: Conceptualització de les estructures de la figura anterior. A l'exemple suposem que tots els nodes són instància d'un mateix concepte anomenat *concept*.

Per a representar les cadenes estructurals en el nostre llenguatge, farem servir l'operador *:STRC*. Per exemple, les tres cadenes estructurals conceptualitzades de la figura 6.2 es representen com:

(:STRC *concept assures concept precondition concept*)

(:STRC *concept contributes-to concept wishes concept*)

(:STRC *concept performs concept*)

Preferim treballar amb cadenes estructurals conceptualitzades i no simplement amb cadenes estructurals perque, segons el nostre parer, el coneixement que ens interessarà de cara a raonar amb les definicions dels conceptes està en la mateixa jerarquia de conceptes i no en les instàncies d'aquests.

6.2 Propietats Estructurals Complexes

Podem combinar cadenes estructurals conceptualitzades per a formar propietats estructurals més complexes. Els elements bàsics per a combinar les cadenes estructurals són els *operadors lògics* (*:AND*, *:OR*, *:NOT*)

Per exemple podem definir una mare, com una persona de sexe femení que té fills. La definició estructural de mare seria de la següent forma:

```
(:AND (:STRC persona sexe femeni)
      (:OR (:STRC persona fill persona)
           (:STRC persona filla persona)))
```

També podríem definir el concepte de solter de la següent manera:

```
(:AND (:STRC persona sexe femeni)
      (:NOT (:STRC persona conjuge persona)))
```

Tot això ens permet especificar propietats equivalents a les *descripcions de tipus 1* de Woods [8]. Però les capacitats expresives de YAYA van una mica més enllà, ja que permet fer *unificació de nodes* (bé es tracti de conceptes o bé es tracti de relacions) dins les propietats estructurals. Això ho podem fer amb l'operador *UNIFY*, que a diferència dels tres operadors lògics que havíem vist fins ara, no combina propietats estructurals sinó que les modifica.

Amb UNIFY podem fer coses força interessants com per exemple especificar les relacions que són reflexives respecte d'algun tipus o concepte (totes les instàncies del concepte estan relacionats amb ells mateixos mitjançant la relacio). Per a més comoditat representarem l'operador UNIFY com l'operador infix ':', situat entre el element unificat i el nom que li donem. Així, sempre que veiem, element:nom, en realitat voldrem dir (:UNIFY element nom)):

```
(:AND (:STRC relation:r frame:x relation:r frame:x)
      (:STRC relation:r frame:x every+inv type))
```

En aquesta definició apareix un element nou: la relació *every+inv*. Aquesta relació és la inversa de *every*, que és la manera de representar el *per a tot* de YAYA. En quant a la definició, a la primera cadena estructural diem que un element està relacionat amb ell mateix mitjançant r, i a la segona diem que això passa per a tots els elements d'un tipus. Aleshores, les relacions que compleixin això, seran reflexives per al tipus en qüestió.

També podem apreciar que no es fa distinció entre conceptes i relacions, i que les cadenes estructurals poden començar tant en conceptes com en relacions.

Amb una mica més de complexitat podem definir altres propietats matemàtiques sobre les relacions, com per exemple: la simètrica, associativa, commutativa, transitiva, etc.

Com a exemple d'ús de UNIFY per a definir conceptes i no pas relacions, suposem que estem representant programes. Podríem definir el concepte de funció recursiva com: *(:STRC funcio:x CRIDA-A funcio:x)*; és a dir, una funció que es crida a ella mateixa.

Però pot ser que sigui una funció recursiva que no es cridi directament, sino indirectament. Fins ara, les relacions, en les cadenes estructurals, tenien el significat d'unir dos nodes. En casos com aquest es veu que és força interessant no només tindre en compte links directes en les propietats estructurals sinó també les transitivitats de les relacions.

Així doncs introduïm un nou operador anomenat *PATH* que s'aplica a les relacions, dintre de les cadenes estructurals. Aleshores, en lloc de buscar un link directe el que farem és buscar que els dos nodes colindants estiguin relacionats (directa o indirectament). D'aquesta forma, podem modificar la definició de funció recursiva com: *(:STRC funcio:x (:PATH CRIDA-A) funcio:x)*.

6.3 Definició de Conceptes

6.3.1 Classes

hierarchical-frame

Pares: frame

Packages: frames-kernel, frames

Descripció: frame al que se li han afegit certes capacitats que el fan més adient per intervindre en una jerarquia. De moment aquestes capacitats es redueixen a que quan s'esborra el frame, es fa d'una forma transparent; és a dir, els conceptes pares passen a ser conectats amb els conceptes fills amb la relació **specialization-link** i amb les instàncies del concepte esborrat amb la relació **instantiation-link**.

structural-frame

Pares: hierarchical-frame

Packages: frames-kernel, frames

Slots:

structural-property La propietat estructural que defineix el concepte.

Descripció: Aquest tipus de frame incorpora una propietat estructural com a definició, cosa que permet raonar sobre la definició i fer totes les operacions que hem vist al principi del capítol.

6.3.2 Variables

default-structural-frame-class

Packages: frames-kernel, frames

Valor per defecte: hdf[1]

Descripció: La classe que es farà servir per defecte per als frames structurals a *create-structure*.

structural-relations

Packages: frames-kernel, frames

Descripció: Una llista amb les relacions que poden prendre part en les cadenes estructurals. Útil ja que ens pot interessar restringir les relacions a partir de les que raonem. Usada per *get-structures* i per *get-conceptualised-structures**.

6.3.3 Events

create-structure

Packages: frames-kernel

Descripció: Event que s'activa cada cop que es crea una estructura.

6.3.4 Primitives

*create-structure(parents strc &key (reclassify (list *specialization-link* *instantiation-link*)))*

Packages: frames-kernel

Parametres:

parents Una llista amb els conceptes més específics que subsumeixen (és a dir, són més generals) que l'estructura que volem crear.

strc La propietat estructural que serà la definició del nou frame creat.

reclassify Una llista de relacions. En cas de que els frames relacionats amb algun dels pares (per aquestes relacions) sigui subsumit pel frame creat, el link corresponent a la relació de la llista, serà redirigit cap al nou frame acabat de crear. Paràmetre especialment dissenyat per a reclassificar subconceptes i instàncies de *parents* que siguin subsumits per l'estructura creada.

Valor de retorn: Dos valors com a valors múltiple: el frame corresponent a l'estructura i t si aquest s'ha creat o nil si ja estava creat prèviament.

Descripció: Crea un frame estructural. Un cop creat el frame, l'integra a la jerarquia. La funció mira si l'estructura ja esta creada, cas en que no la crea, retornant el concepte corresponent a l'estructura trobada. A més, envia l'event de creacio d'estructura.

conceptualise-structure(strc)

[1] Veure capítol 8.

Packages: frames-kernel

Parametres:

strc La cadena estructural que volem conceptualitzar.

Valor de retorn: Una llista amb les diferents conceptualitzacions de *strc*.

Descripció: Degut a que un frame pot ser instància de més d'un concepte, la conceptualització pot donar lloc a més d'una cadena estructural.

D'altra banda, ara per ara calcula únicament les cadenes estructurals conceptualitzades senzilles, sense els operadors UNIFY i PATH.

generalize-structure(strc concepts)

Packages: frames-kernel

Parametres:

strc L'estructura que volem generalitzar.

concepts Una llista amb els únics conceptes que poden intervindre en l'estructura generalitzada.

Valor de retorn: Una llista amb les estructures generalitzades (en el cas d'herència múltiple pot resultar més d'una estructura. De moment només en retorna una, garantitzant que els conceptes que hi apareixen no subsumeixen cap altre que hi podria aparéixer (però no garantitza que tots els que no subsumeixen cap altre i hi podrien apareixer hi apareguin).

Descripció: Per a cada concepte en la estructura que no estigui en *concepts*, la funció busca el més específic que és més general que ell i a la vegada està en *concepts* i el substitueix per ell.

get-structures(frame &key (maxlength 1))

Packages: frames-kernel

Parametres:

frame El frame del qual volem trobar les cadenes estructurals.

maxlength La longitut màxima de les cadenes estructurals que volem calcular.

Valor de retorn: Una llista amb totes les cadenes estructurals de longitut menor o igual a *maxlength* en les que el node està inmers (la longitut és el número de links que apareixen en la cadena).

Descripció: A més, les calcula únicament per a conceptes; bé, també per a relacions però com si fossin conceptes (és a dir, per al frame que defineix la relació). Les cadenes estructurals que retorna no són conceptualitzades.

get-conceptualised-structures(frame concepts &key (maxlength 1))

Packages: frames-kernel

Parametres:

frame El frame del qual volem trobar les cadenes estructurals.

concepts Una llista amb els únics conceptes que poden intervindre en les estructures conceptualitzades que la funció retornarà.

maxlength La longitut màxima de les cadenes estructurals que volem calcular.

Valor de retorn: Una llista amb totes les cadenes estructurals conceptualitzades de longitut menor o igual a *maxlength* en les que el node està inmers (la longitut és el número de links que apareixen en la cadena).

Descripció: La funció crida a *get-structures* per obtindre les estructures, *conceptualise-structure* per conceptualitzar-les i *generalize-structure* per que només intervinguin conceptes permesos.

6.4 Classificació de Conceptes i Instàncies

6.4.1 Events

frame-classification

Packages: frames-kernel

Descripció: Event que s'activa cada cop que es classifica un frame (activat pel mètode *integrate-into-hierarchy*.

frame-reclassification

Packages: frames-kernel

Descripció: Event que s'activa cada cop que es reclassifica un frame. Això es produeix quan es crea una nova estructura i subconceptes o instàncies dels frames que són els antecessors directes, passen a ser subconceptes o instàncies del frame acabat de crear. L'event és activat pel mètode *update-hierarchical-links*.

6.4.2 Primitives

classify(root frame parents)

Packages: frames-kernel

Parametres:

root Frame a sota del qual classificarem el node.

frame Frame que volem classificar.

parents Llista de pares coneguts del frame.

Valor de retorn: Llista de pares directes del frame després de la classificació.

Descripció: Classifica un frame a sota d'una arrel tenint en compte que és a la vegada fill d'una llista de conceptes (cosa que permet tractar casos d'herència múltiple).

La classificació consisteix en retorna els conceptes més específics que subsumeixen el frame.

El mètode no crea els links d'especialització entre els pares i el frame. Únicament retorna la llista de nodes.

classify-multiple(roots frame)

Packages: frames-kernel

Parametres:

roots Arrels sota les quals volem classificar el frame.

frame Frame que volem classificar.

Valor de retorn: Llista amb els pares directes del frame.

Descripció: Funciona igual que *classify*, però per una llista d'arrels en lloc de per una sola.

subconceptp(son parent)

Packages: frames-kernel

Parametres:

son El frame que volem comprovar si és subsumit per *parent*.

parent El frame que volem comprovar si subsumeix *son*.

Valor de retorn: t si *parent* subsumeix *son*, i nil en cas contrari.

*integrate-into-hierarchy(frame roots &key (reclassify *specialization-link*))*

Packages: frames-kernel

Parametres:

frame El frame que volem integrar a la jerarquia.

roots Una llista amb frames que coneixem que subsumeixen *frame*.

reclassify Tipus de relació jeràrquica amb la qual farem la reclassificació. Si és nil, no es fa reclassificació.

Valor de retorn: Llista de pares directes del node.

Descripció: Classifica el node a la jerarquia creant a més els links corresponents. Un cop fet això reclassifica els subconceptes i instàncies dels pares directes per si són subconceptes i instàncies del nou concepte introduït. Activa l'event *frame-classification*.

*reclassify-respect-to-concept(frame &key (link *specialization-link*))*

Packages: frames-kernel

Parametres:

frame El frame que acabem d'integrar a la jerarquia.

link Tipus de relació jeràrquica amb la qual farem la reclassificació.

Valor de retorn: Una llista amb els frames que s'han reclassificat com a fills del nou frame.

Descripció: La reclassificació només afecta els frames susceptibles de ser classificats com a fills del nou frame; és a dir, les subjerarquies donades pels fills dels pares directes del frame. Modifica els links sempre que sigui necessari.

*reclassify(frame &key (reclassify (list *specialization-link* *instantiation-link*)))*

Packages: frames-kernel

Parametres:

frame El frame que volem reclassificar.

reclassify Llista de relacions que tindrem en compte per a la reclassificació. Es canviaran aquestes relacions si *frame* és subsumit per algun fill dels conceptes que estan lligats a ell per alguna d'aquestes relacions. Típicament aquestes relacions són **specialization-link** i **instantiation-link**.

Valor de retorn: Retorna una llista amb els nous pares del frame.

Descripció: A vegades pot ser útil esperar a reclassificar un frame, per això ens serveix aquest mètode. Modifica els links corresponents a *reclassify* sempre que sigui necessari.

specific-concepts(concepts)

Packages: frames-kernel

Parametres:

concepts La llista de conceptes dels quals volem saber els més específics.

Valor de retorn: Una llista amb els conceptes més específics de la llista; és a dir, aquells conceptes que no subsumeixen cap altre de la llista.

subsumes-structure-p(strc-fill strc-pare)

Packages: frames-kernel

Parametres:

strc-fill L'estructura de la qual volem comprovar si és subsumida per l'altra estructura.

strc-pare L'estructura que volem comprovar si subsumeix l'altra.

Valor de retorn: t si *strc-pare* subsumeix *strc-fill* i nil en cas contrari.

find-subsumers&subsumees(strc frames)

Packages: frames-kernel

Parametres:

strc L'estructura de la que volem trobar els frames subsumidors més específics i els subsumits més generals.

frames Els frames que coneixem que subsumeixen l'estructura i pels quals començarem la cerca.

Valor de retorn: 3 valors retornats com a valors múltiples:

1. La llista de subsumidors.
2. La llista de subsumits.
3. El concepte que es correspon a l'estructura si aquest ja existeix a la jerarquia o nil en cas contrari.

find-subsumees&subsumers(strc frames)

Packages: frames-kernel

Parametres:

strc L'estructura de la que volem trobar els frames subsumidors més específics i els subsumits més generals.

frames Els frames que coneixem que són subsumits per l'estructura i pels quals començarem la cerca.

Valor de retorn: 3 valors retornats com a valors múltiples:

1. La llista de subsumidors.
2. La llista de subsumits.
3. El concepte que es correspon a l'estructura si aquest ja existeix a la jerarquia o nil en cas contrari.

Descripció: Aquesta funció fa la mateixa tasca que *find-subsumers&subsumees*, però en lloc de partir dels pares, parteix dels fills coneguts i va pujants per la jerarquia.

Capítol 7

Descripcions i Representació Probabilística

YAYA és un sistema que, tot i que pot funcionar perfectament com un sistema de frames normal i corrent que incorpora raonament terminològic, està pensat per a servir en tasquest d'aprenentatge automàtic.

Els elements bàsics sobre els que podem aprendre són les propietats estructurals que hem descrit en el capítol 6. I el mecanisme que ens permet aprendre és bàsicament la dualitat *definició - descripció*; és a dir, la combinació del raonament terminològic i la representació probabilística.

Abans d'anar més enllà, veiem què entenem per descripció. Anomenem *descripció* al conjunt de parells *(propietat , rati)*, on propietat és una propietat estructural que té alguna de les seves instàncies i rati és el percentatge d'instàncies que la té.

Aleshores, el mecanisme per el qual aprenem és força senzill: d'alguna manera tenim una sèrie de conceptes definits, constutuïnt la nostra jerarquia (no entrem en com els hem definit). Aquests conceptes tenen una definició que és una propietat estructural i també tenen cada un una sèrie d'instàncies. A partir de les instàncies calculem la descripció dels conceptes, cosa que ens permet establir correlacions entre propietats estructurals més o menys complexes.

7.1 Classes

description

Slots:

ninstances Número d'instàncies que s'han tingut en compte per calcular la descripció. Necessari per qüestions de ratio.

properties Una taula de hash que conté les propietats com a claus i el número d'instàncies que tenen la propietat com a valor.

Descripció: Classe que serveix per encapsular una descripció.

description-frame

Pares: frame

Slots:

description Conté un valor de tipus *description* que és la descripció del frame.

Descripció: Afegeix una descripció probabilística a la definició del frame.

description-relation

Pares: relation description-frame

Descripció: Afegeix el comportament de relació als frames amb descripció probabilística.

7.2 Events

compute-description

Packages: frames-kernel, frames

Descripció: Event que s'activa cada cop que es genera una descripció per a un concepte.

generalize-instance-properties

Packages: frames-kernel, frames

Descripció: Event que s'activa cada cop que es generalitzen les propietats d'una instància cap als conceptes que el tenen com a instància.

7.3 Primitives

average-description(&rest descriptions)

Packages: frames-kernel

Parametres:

descriptions Llista de descripcions de les que volem calcular la mitja.

Valor de retorn: Una descripció nova que és la descripció mitja de *descriptions*. Calculem la descripció mitja calculant la mitja dels ratios de totes les propietats que intervenen en *descriptions*, ponderats pel número d'instàncies de cada descripció. El número d'instàncies total de la nova descripció es calcula com la suma de les instàncies de *descriptions*, cosa que només és certa quan les descripcions són disjuntes.

description-difference(d1 d2)

Packages: frames-kernel

Parametres:

d1, d2 Les descripcions de les que volem conèixer la diferència.

Valor de retorn: Una nova descripció que és la diferència de les dues descripcions *d1* i *d2*. La diferència es calcula com la diferència entre els ratios de les propietats (amb això, podem tenir ratios negatius. Aquesta funció es fa servir per a calcular descripcions de relacions, on tenen sentit els ratios negatius (veure [1], apartat referent a l'aprenentatge de relacions). El número d'instàncies es perd, mantenint-se el ratio.

similarity(d1 d2)

Packages: frames-kernel

Parametres:

d1, d2 Les descripcions de les que volem calcular la similaritat.

Valor de retorn: Un valor entre 0 i 1 que ens dóna una mesura de la similaritat entre *d1* i *d2*.

Descripció: Informalment, per calcular la similaritat entre dues descripcions, calculem la quantitat d'informació comú i la dividim per la màxima quantitat d'informació comú possible (el màxim dels dos ratios per a cada propietat). En un entorn com aquest en que poden aparèixer una gran quantitat de propietats però normalment no n'apareixen gaires, aquesta sembla una bona opció.

Aquest mètode també es pot aplicar per a *description-frames*, cas en el qual els paràmetres són dos *description-frame* en lloc de dos propietats estructurals.

show-description(frame &optional (stream t))

Packages: frames-kernel

Parametres:

frame El frame del que volem veure la descripció.

stream El stream pel que la volem veure.

Valor de retorn: t

Descripció: Imprimeix la descripció d'un *description-frame* en un stream.

compute-concept-description(frame)

Packages: frames-kernel

Parametres:

frame El concepte del qual volem calcular la descripció.

Valor de retorn: Dos valors com a valors múltiple: la llista de les instàncies del concepte i la llista de propietats que conformen la descripció.

Descripció: Calcula la descripció d'un concepte, modificant-la.

compute-relation-description(relation concept)

Packages: frames-kernel

Parametres:

relation La relació de la qual volem calcular la descripció.

concept El concepte que ens dóna l'entorn en el qual volem calcular la descripció de la relació (veure explicació).

Valor de retorn: La relació de la qual hem calculat la descripció.

Descripció: Calcula la descripció d'una relació, modificant-la. Definim com a descripció d'una relació la diferència entre les descripcions del concepte entorn i el subconcepte més general d'aquest que a la vegada és subsumit per la propietat estructural bàsica: *(:STRC frame relation frame)* (on *relation* és la relació en qüestió).

generalize-instance-properties(frame)

Packages: frames-kernel

Parametres:

frame La instància de la que volem generalitzar les propietats.

Valor de retorn: Dos valors com a valors múltiples: La llista de conceptes sobre la que s'ha fet la generalització i una llista amb les propietats estructurals el ratio de les quals ha canviat.

Descripció: Generalitza les propietats d'una instància als conceptes que el tenen com a instància. Evidentment modifica les descripcions d'aquests conceptes.

Capítol 8

Classes de Frames

YAYA defineix una conjunt no gens despreciable de classes CLOS que defineixen diferents comportaments per a diferents tipus de frame. Per tenir les coses més clares, a la figura 8.1 tenim la jerarquia amb totes les classes definides (n'hi ha algunes que no han estat comentades al llarg del text, bé perque només combinen comportament de vàries classes més simples o bé perque s'apartaven una mica del tronc bàsic de YAYA.

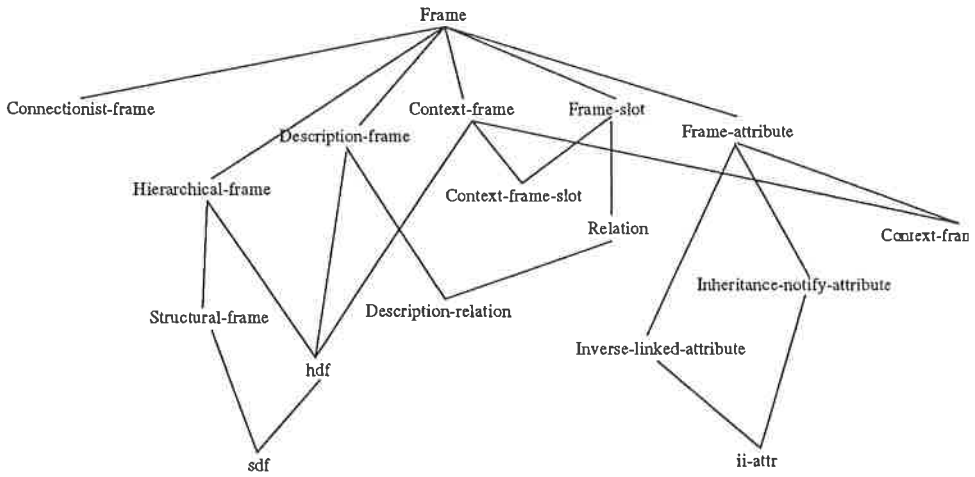


Figura 8.1: Jerarquia de les classes CLOS que defineixen el comportament dels frames.

A continuació comento breument cada una de les classes que introdueix alguna característica nova respecte dels seus pares (la resta són combinació de comportaments):

frame Defineix el comportament bàsic de tot frame.

frame-slot Defineix el comportament dels slots.

frame-attribute Defineix el comportament dels atributs.

relation Defineix el comportament de les relacions.

inheritance-notify-attribute Tipus d'atributs que notifiquen als conceptes més generals que s'ha actualitzat el valor de l'atribut.

inverse-linked-attribute Tipus d'atributs per als que es realitzen automàticament les modificacions referents a relacions inverses.

context-frame Introdueix la presència de contextes.

description-frame Introdueix la possibilitat de que els conceptes incorporin descripcions construïdes a partir de les instàncies en forma de representacions probabilístiques.

hierarchical-frame Incorpora capacitats que el fan interessants per a conceptes de jerarquia (ara per ara sol borrat transparent).

structural-frame Tipus de frame que incorpora definicions estructurals. Possibilita el raonament terminològic.

connectionist-frame Tipus de frame que modifica els valors amagatzemats als atributs per que incorporin un pes. Amb això podríem definir un model localista de xarxa neuronal, però una mica especial doncs les conexions tindrien noms.

Capítol 9

Nucli de la Base de Coneixement

A més de tot el conjunt de primitives per operar amb la base de coneixement, YAYA proporciona un conjunt de frames necessaris per començar a treballar amb la base de coneixement. Aquest conjunt de frames està constituït per un conjunt de frames que defineixen conceptes, slots, relacions i atributs bàsics per:

- Guardar valors en slots i passar valors per defecte dels conceptes a les seves instàncies.
- Guardar la transitivitat de les relacions.
- Slots bàsics per a definir restriccions de domini i rang (sistema de tipus).
- Tractament de relacions inverses, herència i construcció de jerarquies, instanciació.
- Un sistema de tipus bàsic en el qual tots els frames són instància d'algun concepte i per tant tenen certes restriccions imposades.

9.1 Elements Bàsics

Els elements bàsics de la base de coneixement de YAYA estan definits al contexte *kernel*. A continuació distingim entre aquells que denoten slots, relacions i atributs; però evidentment tots ells estan definits com a frames a la base de coneixement.

9.1.1 Slots

transitivity

Classe CLOS: context-frame-slot

Slots:

instance slot-type

domain relation-type

Descripció: Defineix el slot que té com a contingut la definició de la transitivitat d'una relació. Els slots de la seva definició fan referència al sistema de tipus incipient que s'explica en la següent secció.

description

Classe CLOS: context-frame-slot

Slots:

instance slot-type
domain type

Descripció: Serveix per donar una descripció informal del frame en el que està situat.

9.1.2 Relacions

domain

Classe CLOS: context-relation

Slots:

instance restriction-slot-type
domain slot-type
range type2
inverse domain-of

Descripció: Aquest slot, quan està present en un frame (que a la vegada ha de ser un slot), ens determina el domini de frames en el qual el slot pot estar definit.

range

Classe CLOS: context-relation

Slots:

instance restriction-slot-type
domain slot-type
range type2
inverse range-of

Descripció: Aquest slot ens determina, en la definició d'un slot, el rang de valors que poden estar associats al slot.

needs

Classe CLOS: context-relation

Slots:

instance relation-type
domain type2
range slot-type
inverse needed-by

Descripció: Ens diu que un frame determinat ha de tnidre necessàriament un slot.

domain-of

Classe CLOS: context-relation

Slots:

instance relation-type
domain type2
range slot-type
inverse domain

Descripció: Inversa de *domain*.

range-of

Classe CLOS: context-relation

Slots:

instance relation-type
domain type2
range slot-type
inverse range

Descripció: Inversa de *range*.

needed-by

Classe CLOS: context-relation

Slots:

instance relation-type
domain slot-type
range type2
inverse needs

Descripció: Inversa de *needs*.

inverse

Classe CLOS: context-relation

Slots:

instance relation-type
domain relation-type
range relation-type
inverse inverse

Descripció: Donada una relació, ens diu quina és la seva inversa.

is-a

Classe CLOS: context-relation

Slots:

instance relation-type
domain type
range type
inverse is-a+inv
transitivity (repeat is-a 0 inf)

Descripció: Relació entre conceptes que ens diu que el frame que està com a domini és més específic que el frame que està com a rang. Normalment, **generalization-link** està associat a aquest frame.

is-a+inv

Classe CLOS: context-relation

Slots:

instance relation-type
domain type
range type
inverse is-a
transitivity (repeat is-a+inv 0 inf)

Descripció: Relació inversa a *is-a*. Normalment associada a **specialization-link**.

instance

Classe CLOS: context-relation

Slots:

instance relation-type
domain frame
range type
inverse instance+inv
transitivity (list (repeat is-a 0 inf) instance (repeat is-a 0 inf))

Descripció: Modelitza la relació entre instàncies i conceptes. Ens indica que el frame que està com a domini és una instància del frame que està com a rang. Normalment és la relació associada a **conceptualization-link**.

instance+inv

Classe CLOS: context-relation

Slots:

instance relation-type
domain type
range frame
inverse instance
transitivity (list (repeat is-a+inv 0 inf) instance (repeat is-a+inv 0 inf))

Descripció: És la relació inversa de *instance*. Normalment associada a **instantiation-link**.

9.1.3 Atributs

default

Classe CLOS: ii-attr

Slots:

instance context-frame-attribute

Descripció: Atribut que pot estar present en un concepte, i indica el valor que el atribut *value* té per defecte en el mateix slot en les instàncies del concepte.

value

Classe CLOS: ii-attr

Slots:

instance attribute-type

Descripció: Representa el valor del slot en el que està present.

9.2 Sistema de Tipus

Apart de les facilitats per comprovar certes restriccions sobre les instàncies dels conceptes, YAYA proporciona un conjunt de conceptes o tipus bàsics que modelitzen el mateix sistema de representació. Aquests frames estan definits al contexte *type-system*.

frame

Classe CLOS: **default-frame-class**

Slots:

instance type

Descripció: És l'element més genèric de la base de coneixement. Qualsevol frame és instància de frame. Ens serveix sobretot per a tenir clar que estem en un sistema de representació; i cadascun dels frames és un element de representació.

type

Classe CLOS: **default-frame-class**

Slots:

instance type2

is-a frame

slot-type type

Descripció: Aquesta classe preten fer la distincio token-type. Tots els objectes presents a la nostra base de coneixement haurien de ser tokens (es a dir, haurien de ser instancia d'algu). Pero no tots ens defineixen un tipus d'objecte, una classe.

type2

Classe CLOS: **default-frame-class**

Slots:

instance type3

is-a type

slot-type2 type2

Descripció: Defineix el comportament dels tipus. Els slots que siguin *slot-type* presents en la definició d'un tipus, han de tindre valors que siguin tipus.

type3

Classe CLOS: **default-frame-class**

Slots:

instance type3

is-a type2

Descripció: Anem una mica més enllà de *type2* per a definir el domini i el rang dels *slot-type2*. Tanquem el cicle dels tipus fent que *type3* sigui una instància d'ell mateix (en realitat ho hauria de ser de *type4*, que no definim; però com *type4* seria una subclasse de *type3*; d'aquesta manera perdem la mínima informació possible.

Això introdueix la recursivitat necessaria en tot sistema de tipus que tracti als tipus com a objectes. En ultima instancia, Alguna classe ha de tancar el cicle i ser instància d'ella mateixa.

slot-type

Classe CLOS: frame-slot

Slots:

instance slot-type2
is-a type
domain value type2
default type
range value type
default type

Descripció: La classe de tots els slots. Defineix domini i rang per ell i per les seves instàncies (atribut *default*).

relation-type

Classe CLOS: frame-slot

Slots:

is-a slot-type

Descripció: La classe de totes les relacions.

attribute-type

Classe CLOS: **default-frame-class**

Slots:

is-a type

Descripció: La classe dels atributs. Únicament té dues instàncies definides a *kernel*, que són *value* i *default*.

slot-type2

Classe CLOS: frame-slot

Slots:

instance type3 slot-type2
is-a slot-type
instance+inv slot-type
restriction-slot-type type2
domain type3
range type3

Descripció: Definicio de la classe de slots que a la vegada son classes de slots. Ens serveix per definir els slots que defineixen restriccions sobre els slots concrets

A la figura 9.1 podem veure les relacions d'inclusió i instanciació entre els diferents tipus definits en el sistema de tipus.

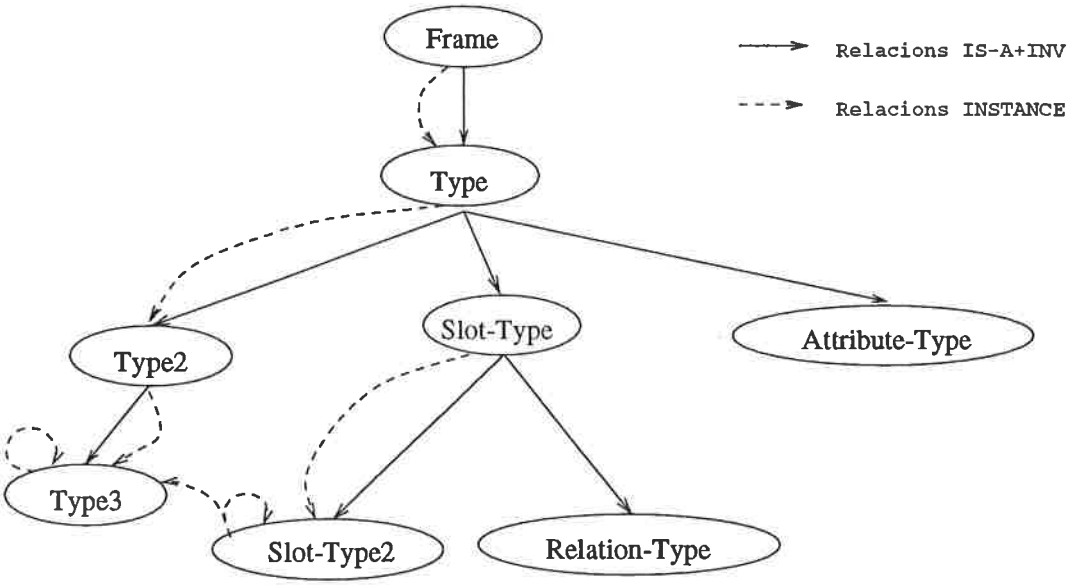


Figura 9.1: Jerarquia del sistema de tipus definit per YAYA.

Bibliografia

[1] J. Alvarez and N. Castell. Knowledge based techniques for software requirements validation. Technical Report LSI-96-R, Universitat Politècnica de Catalunya, 1996.

[2] R. Brachman and J. Schmolze. An overview of the KL-ONE knowledge representation system. *Cognitive Science*, 9:171–216, August 1985.

[3] David Brill. *Loom Reference Manual, Version 2.0.* University of Southern California, December 1993.

[4] Carnegie Group. *Knowledge Craft Reference Guide*, 1992.

[5] D. Fisher. Knowledge acquisition via incremental conceptual clustering. *Machine Learning*, 2:139–172, 1987.

[6] R. MacGregor. The evolving technology of classification-based knowledge representation systems. In Sowa [7].

[7] John F. Sowa, editor. *Principles of Semantic Networks. Explorations in the Representation of Knowledge.* Morgan Kaufman Publishers, 1991.

[8] W. A. Woods. Understanding subsumption and taxonomy: A framework for progress. In Sowa [7], chapter 1, pages 45–94.

Índex de matèries

conceptualization-link, 12
context, 28
default-frame-class, 10
default-structural-frame-class, 35
generalization-link, 24
instantiation-link, 12
inverse-link, 24
special-context-relations, 29
specialization-link, 24
structural-relations, 35
transitivity-slot, 24
value-attribute, 14
:AND, 33
:NOT, 33
:OR, 33
:STRC, 33

activate-event, 18
add-attribute-value, 14
add-attribute-values, 14
add-value, 16
add-values, 16
attribute-type, 51
attributep, 12
average-description, 41

cadena estructural, 32
cadena estructural conceptualitzada, 32
classificador, 4
classify, 37
classify-multiple, 37
CLOS, 3
compute-concept-description, 42
compute-description, 41
compute-relation-description, 42
conceptualise-structure, 35
connectionist-frame, 45
context-frame, 28, 44
context-frame-attribute, 28
context-frame-slot, 28
context-relation, 28
contexte, 4
create-attribute, 12
create-context, 29
create-default-attributes, 13
create-frame, 8, 10
create-slot, 10
create-structure, 35

daemons, 14, 15, 17
deactivate-event, 18
default, 49
define-context, 29
define-event, 19
define-frame, 8
definició, 40
delete-after-daemons, 17
delete-any-attribute-values, 15
delete-any-values, 16
delete-attribute, 13
delete-attribute-value, 14
delete-before-daemons, 17
delete-context, 29
delete-frame, 9, 10
delete-slot, 11
delete-value, 16
dereference-frame, 8
descripció, 40
    de tipus 1, 3, 33
description, 40, 46
description-difference, 41
description-frame, 40, 45
description-relation, 40
domain, 16, 47
domain-of, 47

entrada/sortida, 19
event, 18
events, 18

filter-visible-frames, 31
find-frame, 8
find-subsumees&subsumers, 39
find-subsumers&subsumees, 39
frame, 7, 44, 50
    nom, 27
    nom complet, 27
frame-attribute, 12, 44
frame-classification, 37
frame-precedence-list, 24

frame-reclassification, 37
frame-slot, 10, 44
framep, 9

generalize-instance-properties, 41, 42
generalize-structure, 36
get-after-daemons, 17
get-attribute-value, 15
get-attribute-values, 15
get-attributes, 13
get-before-daemons, 17
get-conceptualised-structures, 36
get-direct-attribute-value, 15
get-direct-attribute-values, 15
get-direct-value, 16
get-direct-values, 16
get-local-attributes, 13
get-local-relations, 24
get-local-slots, 11
get-relations, 25
get-relatives, 25
get-slots, 11
get-structures, 36
get-value, 16
get-values, 16

hierarchical-frame, 34, 45

inheritance-notify-attribute, 44
instance, 49
instance+inv, 49
instancep, 25
integrate-into-hierarchy, 38
inverse, 48
inverse-linked-attribute, 23, 44
is-a, 48
is-a+inv, 48

lògica descriptiva, 32
local-attributep, 13
local-slotp, 11
LOOM, 3, 32

mètodes
    after, 17
    before, 17
mapframes, 9
missatges, 18

needed-by, 48
needs, 47
nom
    frame, 27

operadors lògics, 33

PATH, 34
pop-context, 30
pp, 20
print-frame, 19
push-context, 29
put-after-daemons, 17
put-attribute-value, 15
put-attribute-values, 15
put-before-daemons, 17
put-value, 16
put-values, 16

range, 16, 47
range-of, 48
raonament terminològic, 3, 32
reclassify, 38
reclassify-respect-to-concept, 38
reference-frame, 7
relacions, 22
relatedp, 25
relation, 23, 44
relation-type, 51
relationp, 25
remove-context, 29
representació probabilística, 3

save-frame, 21
save-frames, 20
save-kb, 20
save-taxonomies, 20
save-taxonomy, 20
send-event, 19
set-context, 31
show-description, 42
similarity, 41
sistema de tipus, 16
sistema modular, 3
slot-type, 51
slot-type2, 51
slotp, 11
specific-concepts, 39
structural-frame, 34, 45
subconceptp, 38
subsumes-structure-p, 39

transitivitat, 22
transitivity, 46
type, 50
type2, 50
type3, 50

unificació de nodes, 33
UNIFY, 33
unlink, 26

unuse-contexts, 30
use-contexts, 30

valid-slot, 16
valid-value, 17
value, 49
visible-contexts, 30
visiblep, 30