Universitá Degli Studi Dell'Aquila

DIPARTIMENTO DI INGEGNERIA E SCIENZE DELL'INFORMAZIONE E MATEMATICA

"Documentation of the DALI Interpreter"

Author:
Sara Ciccolone
July 2018

Detailed Documentation of the DALI Interpreter

The first step that was done, when the work of this thesis began, was to revise the whole implementation, generating at the same time a quantity of documentation such as to adequately support future interpretations and extensions of the interpreter. It has been producted a detailed description of the UML Diagram realized in 2010 by a student in his thesis (Lanti, D.: Reingegnerizzazione dell'interprete DALI ed integrazione con il Web Semantico,(Universitá degli Studi dell'Aquila,2010)), then the following images are extracted by it. The diagrams producted are UML Diagrams, in particular Use Case Diagram and Activity Diagram. This diagrams allow to describe a system by a behavioral point of view. The Activity Diagrams have been realized at different levels of detail: the main diagrams shows all the macroscopic activity is actually an activity composed of many other activities(atomic or compound) and actions, and which are illustrated by other Activity Diagrams. In the picture 1 it is shown a tree of DALI Interpreter in which is represented the pricipal predicate that are called: the nodes represented with a circle described the predicates that are called other predicates, the nodes represented with a rectangle described the predicates that aren't called other predicates.

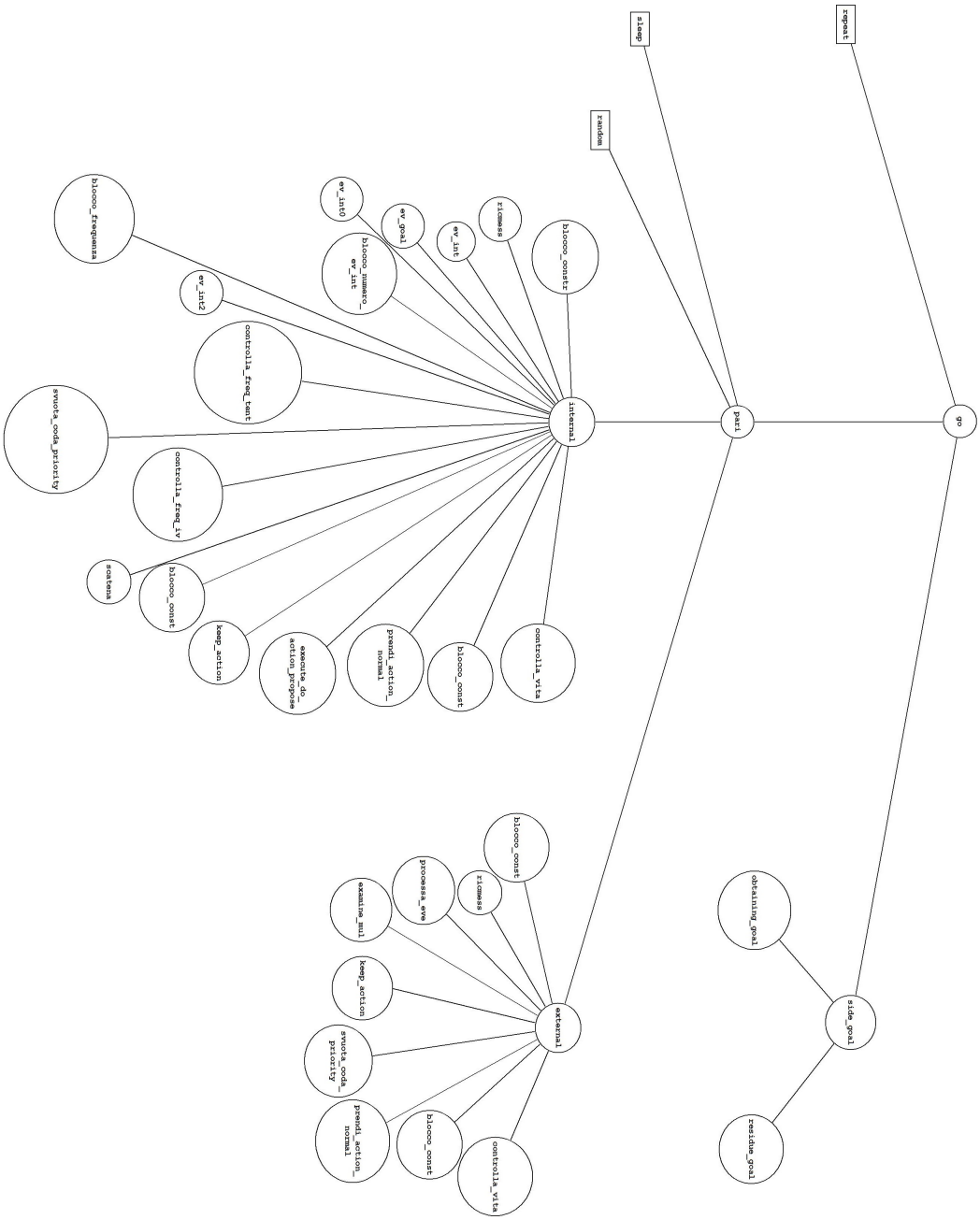
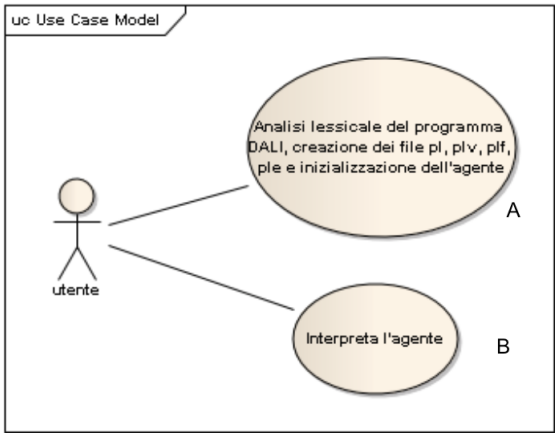


Figure 1: **Tree of DALI Interpreter**

1. Use Case Diagram: Functionality of the DALI Interpreter

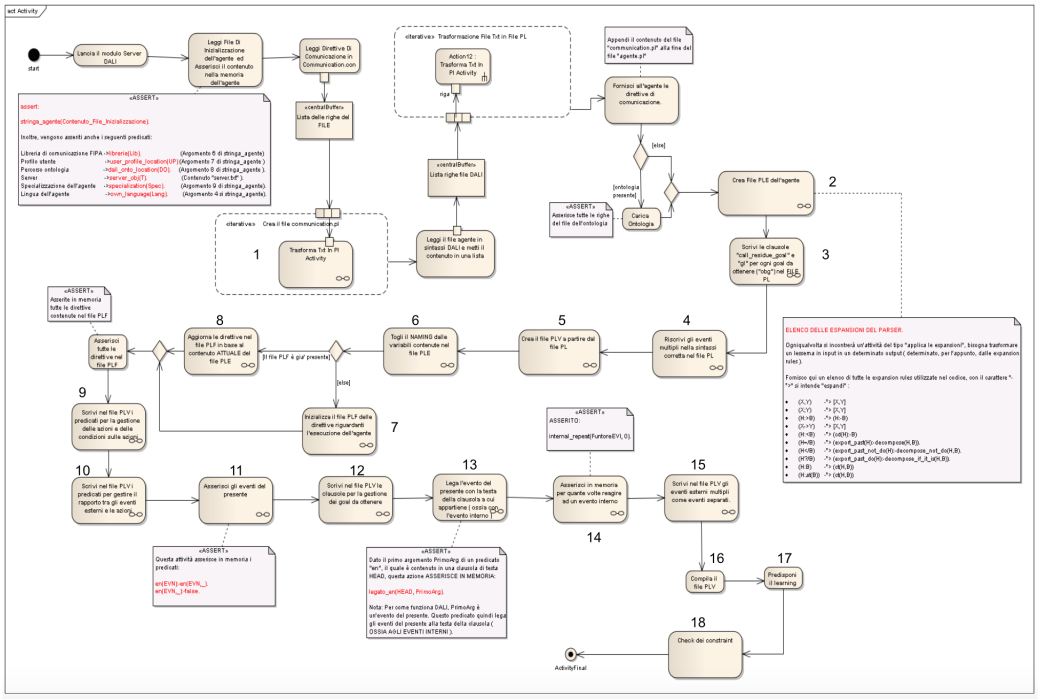


This is the Use Case of DALI Interpreter, where the only external actor to the system is the user. In this Use Case Diagram there are two many divisions that represent the funtionality of the system: the first is the lexical analysis of the DALI program, creation of files pl, plv, plf, ple and initializzation of the agent; the second is interpretations of the agent.

2. Activity Diagrams that explain the Use Case Diagram

The two macroscopic division described in the previous Use Case will be shown : **Inizializzation of the agent (A)** and **Interpret the agent (B).**

A. Initialization of the agent



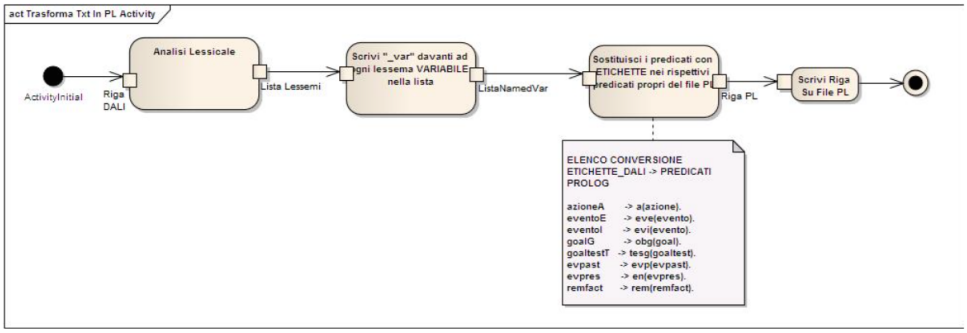
This is the Activity Diagram that explain the Use Case: **"The lexical analysis of the DALI program, creation of files pl, plv, plf, ple and initializzation of the agent"**. Once the file startmash.sh is executed, it starts the predicate *start0* from file *active _ dali _ wi.pl* in line 87. It is read the inizialization file of the agent and the content is asserted in memory of the agent. The following predicates are asserted also:

- the library of FIPA comunications (*Librerie(Lib)*, argument 6 of the string agent in line 91);
- user profile(*user _ profile _ location(UP)*, argument 7 of the string agent in line 91);
- ontology (*dali _ onto _ location(DO)*, argument 8 of the string agent in line 91);
- the server(*server _ obj(T)*, that is the content of the file server.txt);

- specializzation of the agent(*specialization(Spec)*, argument 9 of the string agent in line 91);
- language of the agent (*own _ language(Lang)*, argument 4 of the string agent in line 91).

The directive of communication is read in *communication.con* and a file *communication.pl* (see A.1) is created. The agent file in sintax DALI is read, the content of the file is saved in a list and subsequently the txt-file is transformed in pl-file. The communication directives to the agent are provided (the content of the file communication.pl is append to the end of the file agent.pl). The ontology is loaded, if it is present, then it creates the ple-file of the agent. The plv-file is created from pl-file. The NAMING of the variables is eliminated in ple-file, line 139. If the plf-file exixts (predicate *file _ exist(FilePlf)*, line 141), then the directives in plf-file are updated, based on the current content of ple-file. Else the plf-file is inizialized of the directive about the execution of the agent(line 141). All directives are asserted in plf-file (line 143). The predicates for the management of the actions and conditions of the actions are written in plv-file (line from 154 to 156). The present events are asserted in line 158. The clause for the management of goals to be obtained is written to the plv-file (line 160). The present event is bound with the head of the clause to which it belongs(namely with the internal event). How many times to react to the internal event are asserted in memory (line 164). The multiple external events are written in plv-file as separeted events (line 166). The plv-file is compiled (line 168) and the learning is predisposed (line 171), finally the constraints are checked in line 175.

A.1. Transform the txt file into pl-file



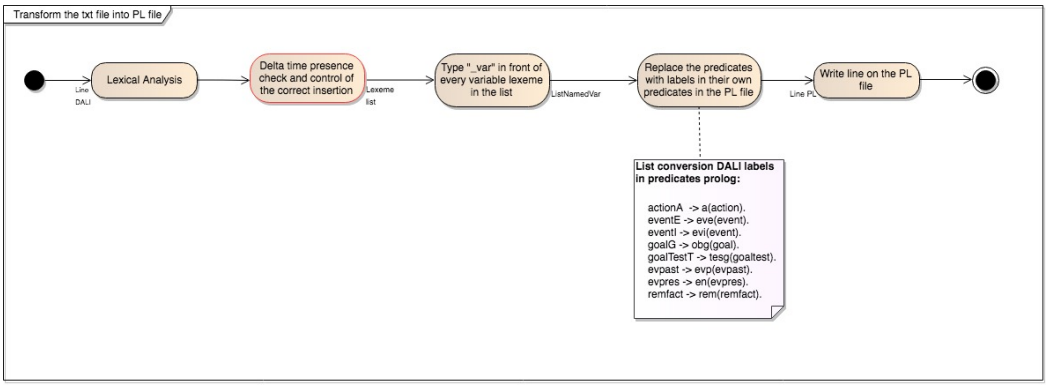
This is a sub-activity of **Inizializzation of the agent**. The txt-file, writes in DALI syntax, is transformed to the pl-file. In file *active _dali _wi.pl*, line 105, the predicate *token(File)* is called. This predicate opens a file called *tokefun.pl* in line 34. From here the lexical analysis starts in line 94: a list of ASCII code is transformed in a list of lexemes. In line 188 it is checked if the character ASCII represents a variable with the predicate *variable(Me)*. If the result is true, the predicate *examine _variable* is called in line 229, in front of each VARIABLE lexeme in the list it is written *" _var"*. Subsequently from line 239 to 256 the events label is examined and the predicates with labels are replaced with a prolog predicates. In the following the list of conversion of events label is shown :

- $azioneA \rightarrow a(azione)$.
- $eventoE \rightarrow eve(evento)$.
- $eventoI \rightarrow evi(evento)$.
- $goalG \rightarrow obg(goal)$.
- $goaltestT \rightarrow tesg(goaltest)$.
- $evpast \rightarrow evp(evpast)$.
- $evpres \rightarrow en(evpres)$.

- $remfact \rightarrow rem(remfact)$.

This predicates are written in pl file.

A.1.(Updated) Transform the txt file into pl-file



This is a sub-activity of **Inizializzation of the agent**. The txt-file, writes in DALI syntax, is transformed to the pl-file. This diagram is the updated version of the previously diagram. In this version the verification of the insertion of the Delta time and the control of its correct syntax was introduced. It is shown the changes in file *tokefun.pl*. In line 159 it is asserted *eventi _ esterni(0)* and *assert(deltaT(0))* that will be used later :

```
take_meta(L,F):-assert(eventi_esterni(0)),
assert(deltaT(0)),assert(parentesi(0)),assert(buffer([])),
name(F,Lf),append(Lf,[46,112,108],Lft),
name(Nf,Lft),if(file_exists(Nf),delete_file(Nf),true),
last(L,U),
repeat,
member(Me,L),
examine_all(Me),
Me==U,!,if(clause(residue(R),_),(name(R1,R),
examine_all(R1)),true),
if(clause(buffer(ParsedC),_),
(retractall(buffer(_)),name(Parsed,ParsedC),
open(Nf, append, Stream, []), write(Stream, Parsed),
close(Stream)),(write('Errore take_meta'),nl)),
re_file(Nf).
```

In line 186 it is inserted:

```
examine_all1(Me):-if(member(Me,['(',')']),
conta_parentesi(Me),true),
if(tempo(Me), scrittura(Me),
(if(variabile(Me),examine_variable(Me),
if(label(Me),examine_label(Me),write_NovarNolabel(Me))))).
```

It is inserted an if-construct in order to check the predicate *tempo(Me)*. If this predicate is true then it is executed the predicate *scrittura(Me)*. From lines 203 to 211 it is inserted:

```
tempo(Me):- name(Me,L), nth0(0,L,El,L1), El==116,
numbertime(L1).
numbertime(L1):- nth0(0,L1,El,L_rest),check_number(El),
scorri(L_rest).
check_number(El):- El>47, El<58.
scorri(L_rest):-if(L_rest=[],true, scorri_list(L_rest)).
scorri_list(L_rest):- nth0(0,L_rest,X,L2),
check_number(X),scorri(L2).
scrittura(Me):-name(Me,L),nth0(0,L,R,L3),
append(Parsed,L2,Parola),assert(buffer(Parola)),
clause(deltaT(X),true),retractall(deltaT(X)),
assert(deltaT(1)).
```

With the predicate *tempo(Me)*, where *Me* is the *tseconds*, it is checked if the first character is the *"t"*. If it is so it is executed *numbertime(L1)*, this predicate it verifies if the characters that following the first character *"t"* are numbers. If the syntax of *tseconds* is correct the predicate *scrittura(Me)* is executed and then it is proceed to write into files .pl and .plv ,relative to the agent, the following predicate:

deltat(seconds).

and finally it is asserted the predicate *deltaT(1)*. In line 243 it is inserted:

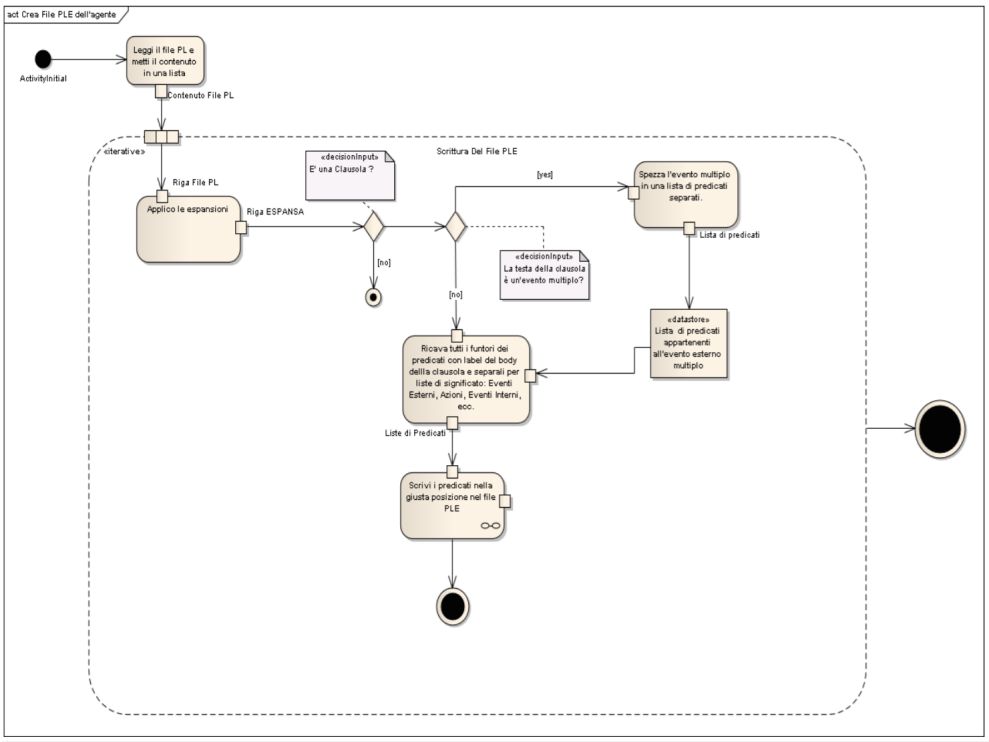
```
appE(L,U):-conta_eventi_esterni,length(L,N),
nth1(N,L,U,R),append([101,118,101,40],R,L1),
re_write(L1),assert(evento_aperto).
```

and line 258:

```
conta_eventi_esterni:- clause(eventi_esterni(X),true),
R is X+1, retractall(eventi_esterni(X)),
assert(eventi_esterni(R)).
```

The predicate *conta _ eventi _ esterni* allows to determine the number of the external event. It is introduced this predicate because if the number of the external event is greater than one, the tseconds must be present in the agent's file.

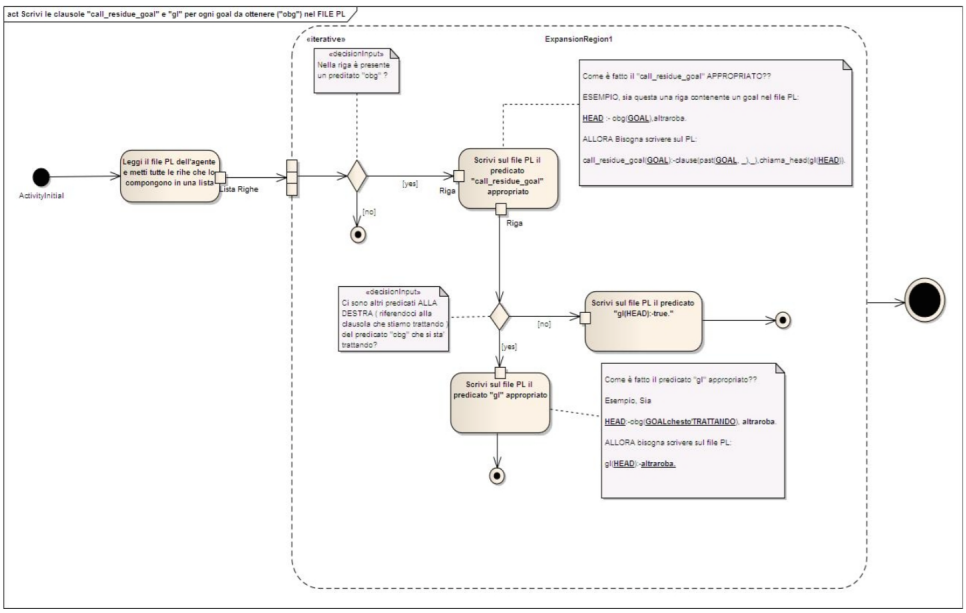
A.2. Create ple-file of the agent



This is the sub-activity of the **Inizialization of the agent**. In file *active _ dali _ wi.pl*, line 131 the predicate *aprifile(FilePl)* is called, the pl-file is read and its content is put in a list. The predicate in line 215 is called, the expansion is applied. If the head of clause isn't a multiple event, all the functors of the predicates with the label are derived and are separated by list

of meaning. Finally the predicates are written in a right position in ple-file. If the head of clause is a multiple event, first the multiple event is broken into a list of separate predicates and then all the functors of the predicates with the label are derived and are separated by list of meaning. Finally the predicates are written in a right position in ple-file.

A.3. Write the clause " call _residue _goal " and "gl" for each goal to get(" obg ") in the pl-file



This diagram represent a sub-activity of the **Inizialization of the agent**. In the file *active _ dali _ wi.pl* , line 133 the predicate *aprifile _ res(FilePl)* is called and the predicates for the manage to the "obtaining goals" are written in pl-file. The predicate is called in file *memory.pl*, line 12. The pl-file of the agent is read and all the lines that compose it are put in a list. In file *memory.pl*, line 41 it is checked if the predicate "obg" is present in the line. If this predicate is present, the predicate *"call _ residue _ goal"* is written in pl-file ,line 49. How is an appropriate *"call_ residue _ goal"* made?

Example: Let this a line containing a goal in pl-file:

$$HEAD :- obg(GOAL), other.$$

Then it must write on pl-file:
$call\_residue\_goal(GOAL) :- clause(past(GOAL, \_), \_),$
$chiama\_head(gl(HEAD)).$
Now if are there any other predicates to the right of the predicate "obg" being dealt with? If yes the predicate "gl" appropriate is written in pl-file .

How is an appropriate predicate "gl" made?

Example: Let

$$HEAD :- obg(GOALthatIsBeingDealtWith), other.$$

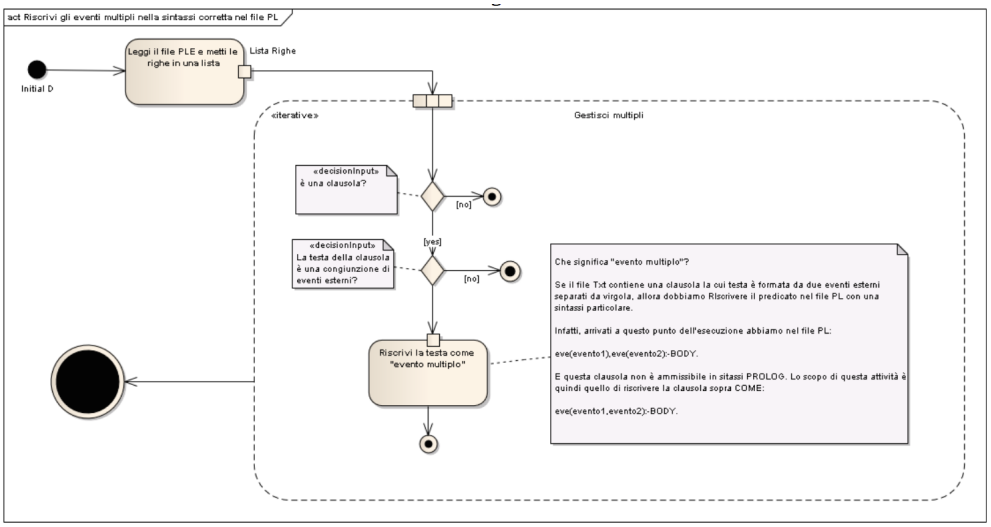
Then it must write on pl-file:

$$gl(HEAD) :- other.$$

If there aren't predicates to the right of the rule then this predicate is written in pl-file:

$$gl(HEAD) :- true.$$

A.4. Rewrite multiple events in the correct syntax in the pl-file



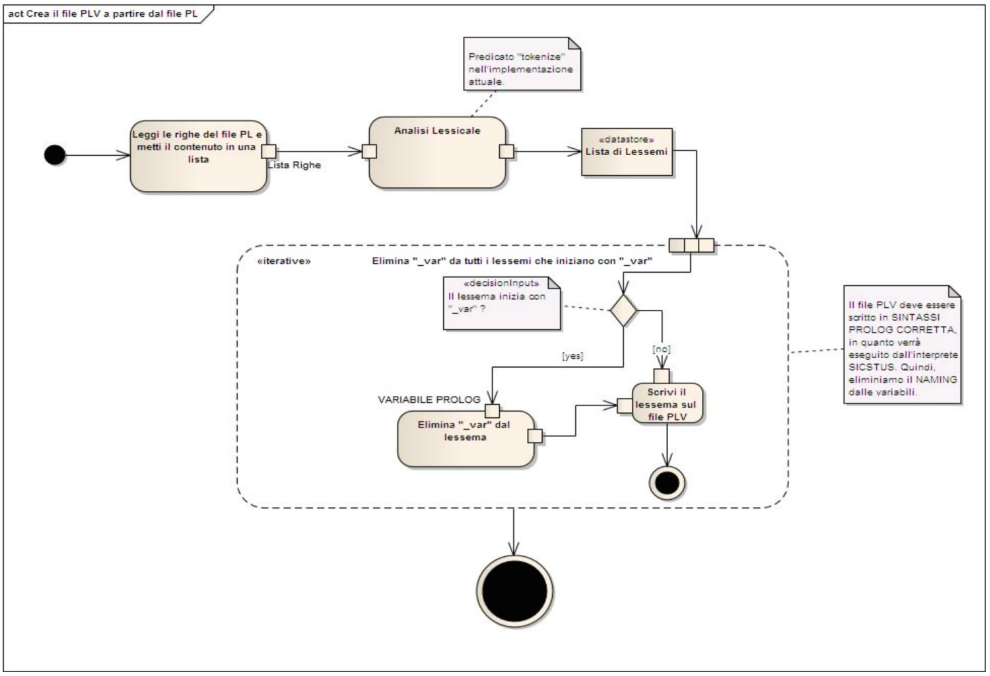
This diagram represent a sub-activity of the **Inizialization of the agent**. In the file *active _ dali _ wi.pl*, line 135 the predicate *carica _ file(FilePl)* is called and *leggi _ mul.pl* is opened the file in line 7. The ple-file is read and its content is put in a list. If it is a clause, then is the head of the clause a conjunction of external events? If yes, the head is rewritten as multiple event. What means multiple event? If the txt-file contains a clause whose head is formed by two external events, comma separeted, then it is necessary rewrites this predicate in pl-file with a particular syntax. In fact at this point of the execution in pl-file it is:

$$eve(evento1), eve(evento2) : -BODY.$$

And this clause is not admissible in PROLOG syntax. The purpose of this activity is to rewrite the above clause as :

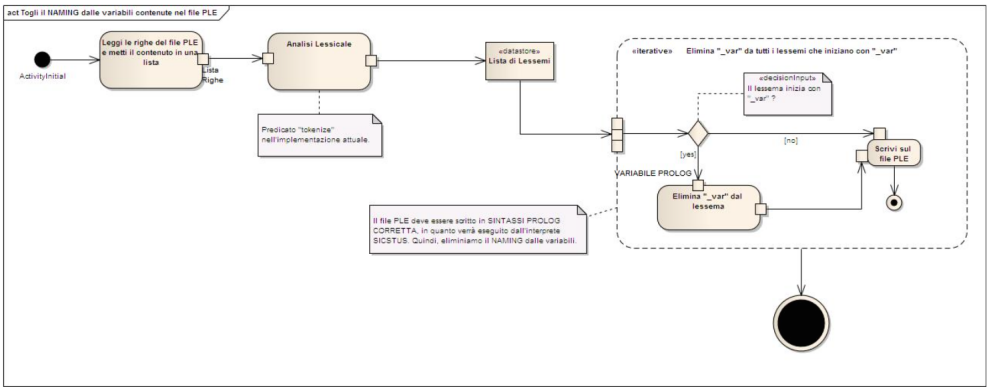
$$eve(evento1, evento2) : -BODY.$$

A.5. Create the plv-file from pl-file



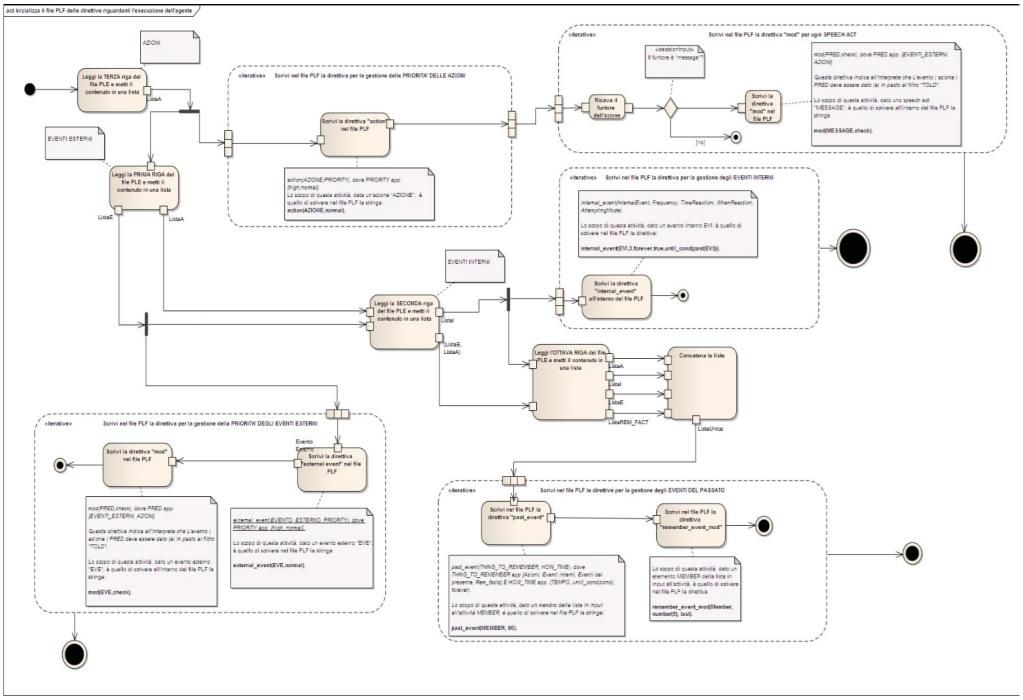
This diagram represent a sub-activity of the **Inizialization of the agent**. In the file *active _ dali _ wi.pl*, line 137 the predicate *togli _ var(F)* is called, and the file *togli _ var.pl* is opened in line 7. The ple-file is read and its content is put in a list. It is started the lexical analysis with predicate "tokenize". Given the list of lexemes, "_ var" is eliminated from all the lexemes that begin with "_ var". If the lexeme begins with "_ var", file *togli _ var.pl* line 88, then *_ var* it is eliminated from lexeme and this is written in plv-file. Else it is written directly in plv-file. The plv-file must be write in a correct Prolog syntax because it will execute by SICSTUS Interpreter. Then the NAMING of the variable is eliminated.

A.6. Eliminate the NAMING of the variable in ple-file



This diagram represent a sub-activity of the **Inizialization of the agent**. In the file *active _dali _wi.pl*, line 139 the predicate *togli _var _ple(Fe)* is called, and the file *togli _var.pl* is opened in line 20. This predicate deals to eliminating the prefix *"var _"* from variables to which the naming has been applied. The lines of ple-file is read and its contents is put in a list. It is started the lexical analysis with predicate "tokenize". Given the list of lexemes, "_var" is eliminated from all the lexemes that begin with "_var". If the lexeme begins with "_var", then _var is eliminated from lexeme and this is written in ple-file. Else it is written directly in ple-file. The plv-file must be write in a correct Prolog syntax because it will execute by SICSTUS Interpreter. Then the NAMING of the variable is eliminated.

A.7. Initialize the plf-file of the directive about the execution of the agent



This diagram represent a sub-activity of the **Inizialization of the agent**. It is in file *active _ dali _ wi.pl*, this diagram describes the else branch in the if-control in line 141. The predicate *"inizializza _ plf(FilePle)"* is called in line 454. In line 455 the first line in ple-file(external events) is read and the content is put in a list(Le), the second line in ple-file(internal events) and the content is put in a list(Li), the third line in ple-file(actions) and the content is put in a list(La) and the octave line in ple-file and puts the content in a list(Lr). This four lists are concatenated and there is only one list in output.

A.7.1. Write in plf-file the directive for the management of external events

From line 468 to 479 the directives of PAST EVENTS are written in plf-file. The directive *"past _ event"* is written in plf-file. The purpose of this activity

is to write the string in the plf file:

past _event (MEMBER, 60)

where MEMBER is a member of a list. Furthermore the directive *"remember _event _mod"* is written in plf-file. The purpose of this activity is to write the string in the plf-file:

remember _event _mod (Member, number(5), last)

where Member is a member of a list.

A.7.2. Write in plf-file the directive for the management of action's priority

In line 500 the third line of ple-file is read and it is put the content in al list, and the directive " action " is written in plf-file. The purpose of this activity is to write the string in the plf-file:

action (ACTION, priority)

where priority could be high or normal. In line 609 with the predicate *"writ _actn _event "* it is checked that if the functor is ugual to message, the directive "mod " is written in plf-file. The purpose of this activity, given a speech act "MESSAGE", is to write the string in the plf-file:

mod (MESSAGE, check).

A.7.3. Write in plf-file the directive for the management of external event's priority

In line 513 the first line of ple-file is read and it is put the content in a list, and it is written the directive " mod " in plf-file. The purpose of this activity, given an external event "EVE", is to write the string in the plf-file:

mod (EVE, check).

In line 596 with the predicate *"writ _est _event "* it is written the directive "external event ". The purpose of this activity, given an external event "EVE", is to write the string in the plf file:

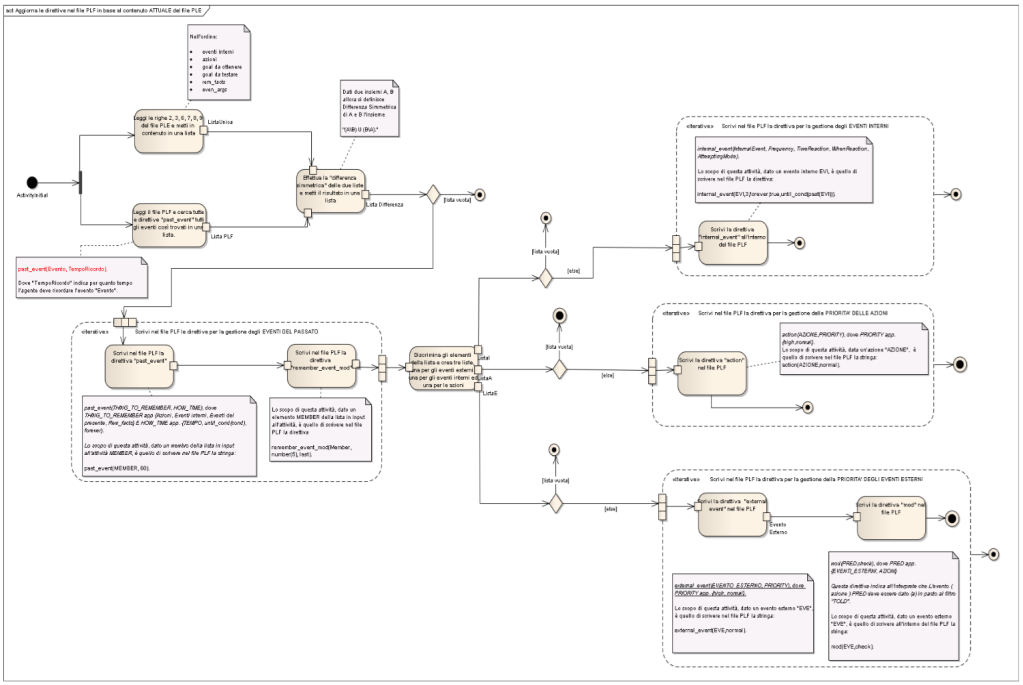
external _event (EVE, normal).

A.7.4. Write in plf-file the directive for the management of internal events

In line 459 the predicate *"smonta _ interno"* is called in line 483. The second line of ple-file is read and the content is put in al list, and the directive *" internal _ event "* is written in plf-file. The purpose of this activity, given an external event "EVE", is to write the string in the plf file:

internal _event(EVI, 3, forever, true, until _cond(past(EVI))).

A.8. Updated the directives in plf-file based on the current content in ple-file



This diagram represent a sub-activity of the **Inizialization of the agent**.It is in file **active _dali _wi.pl**, this diagram describes the then-branch in the if-control in line 141. The predicate *"controlla _ ev _ all(F)"* is called in line 529. The line 2(internal event), line 3(actions), line 6(goal to get), line 7(goal to test), line 8(rem _facts), line 9(event _args) of ple-file are written

and the content is put in a list(L6). In line 542 the plf-file is read and all the directives " past _event" is searched and the events are put in a list(Ls) in line 549. In line 550 the symmetric difference amoung the two lists is carried out and the result is put in a list(Ldf).

A.8.1. Write in plf-file the directive for the management of past events

If this list ldf is not empty the directive for the management of the past event is written in the plf-file. The directive *" past _ event"* is written. Given a member of the input list, the purpose of this activity is to write the string in the plf-file:

past _event(MEMBER, 60).

Subsequently the directive " remember _event _mod" is written in plf-file line 582. Given a member of the input list, the purpose of this activity is to write the string in the plf-file:

remember _event _mod(MEMBER, number(5),last).

Discriminate the element of the list, three lists are created: one for internal events, one for actions, one for external events.

A.8.2. Write in plf-file the directive for the management of internal events

The directive " internal _event" is written in plf-file, from line 603 to 607. Given a internal event EVI, the purpose of this activity is to write the string in the plf-file:

internal _event(EVI, 3,forever, true,until _cond _past(EVI)).

A.8.3. Write in plf-file the directive for the management of action's priority

The directive " action " is written in plf-file, line 609. Given an action, the purpose of this activity is to write the string in the plf-file:

action(ACTION, priority).

where priority could be high or normal.

A.8.4. Write in plf-file the directive for the management of external event's priority

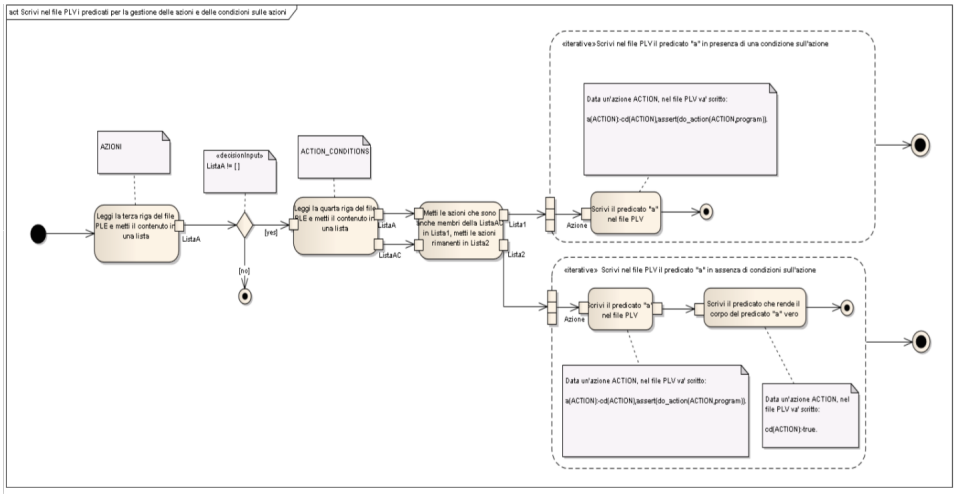
The directive " external event " is written in plf-file, line 596. Given an external event EVE, the purpose of this activity is to write the string in the plf-file:

external _event(EVE, normal).

where priority could be high or normal. The directive " mod " is written in plf-file, line 600. Given an external event EVE, the purpose of this activity is to write the string in the plf-file:

mod(EVE, check).

A.9. Write in plv-file the predicates for the managementof actions and action's conditions



In line 791 the third line of ple-file is read and the content is put in a list(X). If X is not empty then the fourth line in ple-file is read and it is put the content in a list (line 795). The actions that are also members of the list X are puts in list1 and the rimanent actions are puts in list2.

A.9.1. Write in plv-file the predicate in the presence of a condition on the action

Line 810, predicate *clause1*: given an action ACTION , it is written in plv-file:

$$a(ACTION) : -cd(ACTION), assert(do\_action(ACTION, program)).$$

A.9.2. Write in plv-file the predicate in the absence of a condition on the action

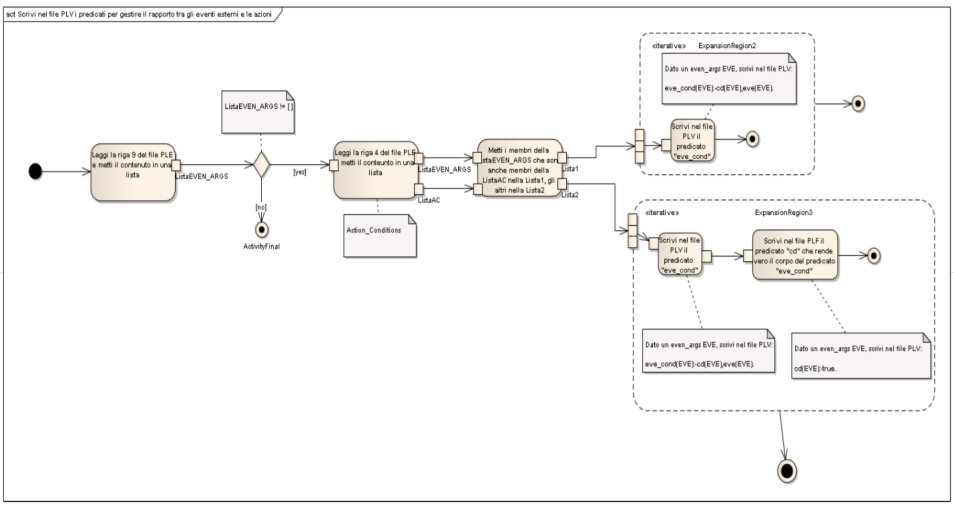
Line 815, predicate *clause2*: given an action ACTION , it is written in plv-file:

$$a(ACTION) : -cd(ACTION), assert(do\_action(ACTION, program)).$$

and it is written the predicate that makes true the body of predicate "a". Given an action ACTION , it is written in plv-file:

$$cd(ACTION) : -true.$$

A.10. Write in plv-file the predicates to manage the relationship between external events and actions



Line 825, predicate *cond _ esterni*. The line 9 of ple-file is read and the content is put in a list. If the list event _args is not empty (line 828), the fourth line in ple-file is read and the content is put in a list, action _condition list . It puts the member of list event _args that are also member of action _condition list in list 1, others in list 2. In line 846, predicate *clause1 _ esterni*. The predicate " eve _cond " is written in plv-file. Given a even _args EVE, in plv-file it is written:

$$eve\_cond(EVE) : -cd(EVE), eve(EVE).$$

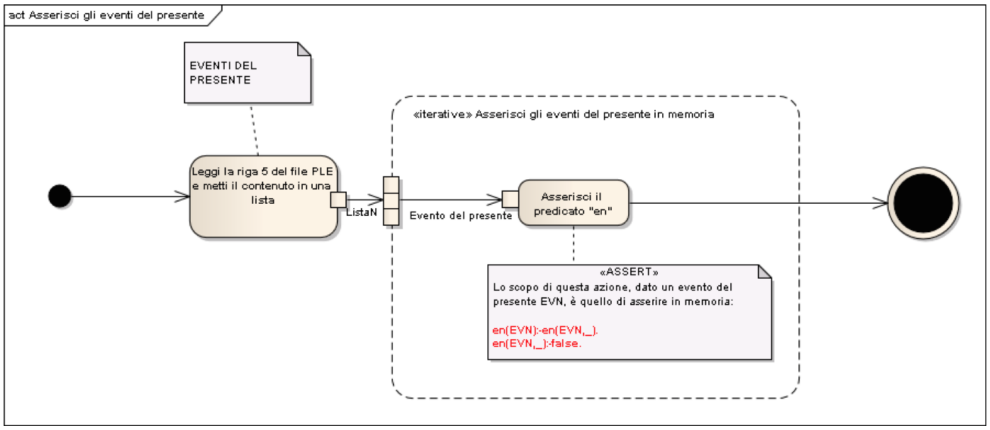
In line 851, predicate *clause2 _ esterni*. The predicate " eve _cond " is written in plv-file. Given a even _args EVE, in plv-file it is written:

$$eve\_cond(EVE) : -cd(EVE), eve(EVE).$$

Given a even _args EVE, in plv-file it is written:

$$cd(EVE) : -true.$$

A.11. Assert the present events

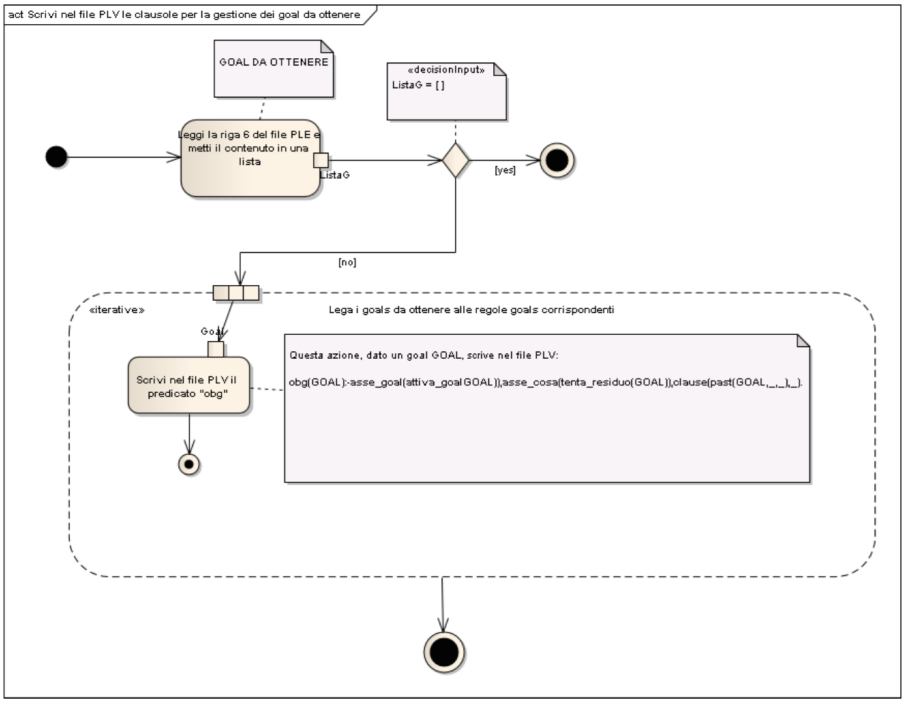


Line 860, predicate *assert _ evN*. The fifth line is read in ple-file and the content is put in a list. The predicate "en" is asserted. Given an present event EVN the purpose of this activity is to write the string in the plf-file:

en(EVN):-en(EVN,_).

en(EVN,_):-false.

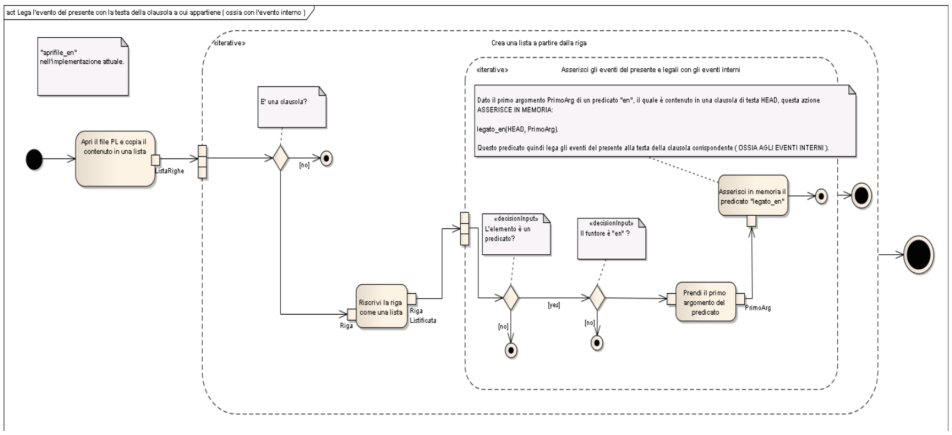
A.12. Write in plv-file the clause to manage of goals to obtain



Line 874, predicate *obg _ goal*. The line 6 of the ple-file is read and the content is put in a list, list G. If this list is not empty then it is written in plv-file the predicate " obg ". This action, given a goal GOAL, is written in plv-file, line 887:

$obg(GOAL) : -asse\_goal(attiva\_goal(GOAL)),$
$asse\_cosa(tenta\_residuo(GOAL)), clause(past(GOAL, \_, \_), \_).$

A.13. Bind the present event with the head of the clause to which it belongs

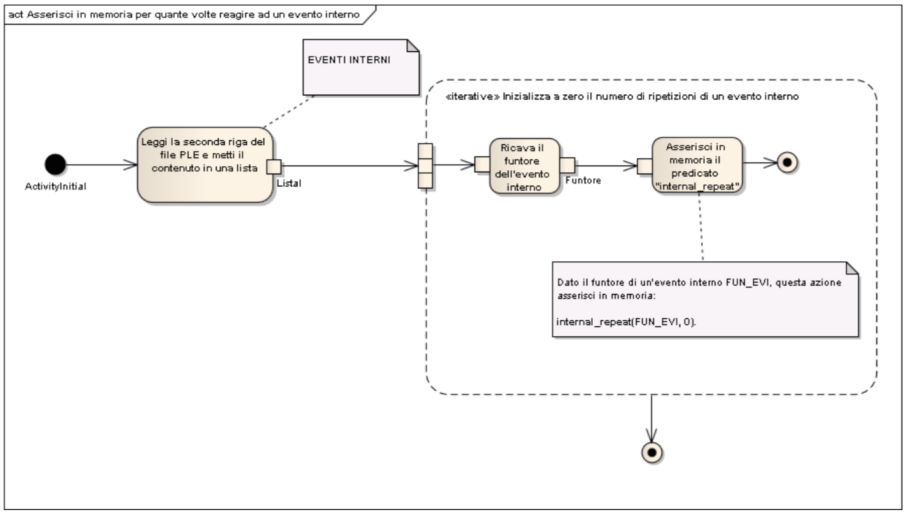


Line 162 the predicate " aprifile _en " is called in line 658. The pl-file is opened and the content is put in a list, this is a list of line. If a line is a clause , it is rewrite the line in a list. If the element is a predicate, and the functor is equal to "en" then it takes the first argument of the predicate and the predicate " legato _en " is asserted in memory . Given a first argument PrimoArg of a predicate " en ", it asserts in memory :

$$legato\_en(HEAD, PrimoArg).$$

This predicate binds the present events with the head of the corresponding clause.

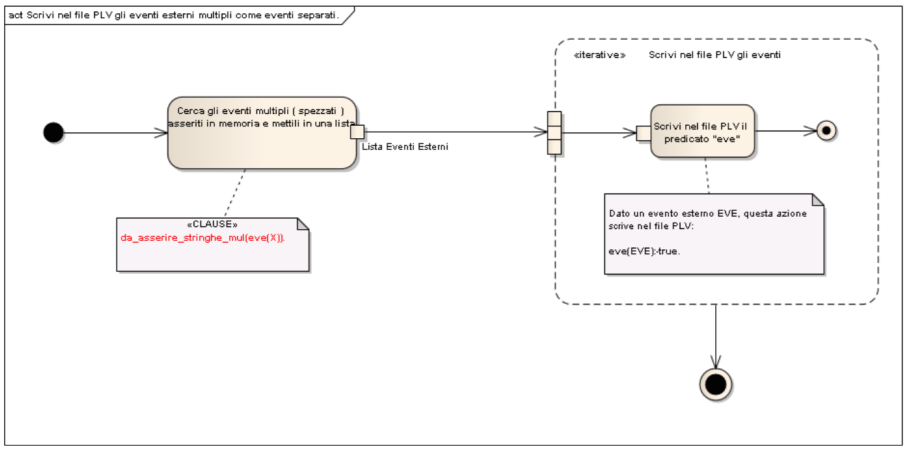
A.14. Assert in memory how many times to react to an internal event



This diagram represent a sub-activity of the **Inizialization of the agent**. Line 164, the predicates " ass _internal _repeat" is called in line 1554. The second line in ple-file is read and the content is put in a list. The functor of the internal event is obtained and the predicate " internal _repeat " is asserted in memory . Given a functor of internal event FUN _EVI, this action asserted in memory :

internal _repeat (FUN _EVI,0)

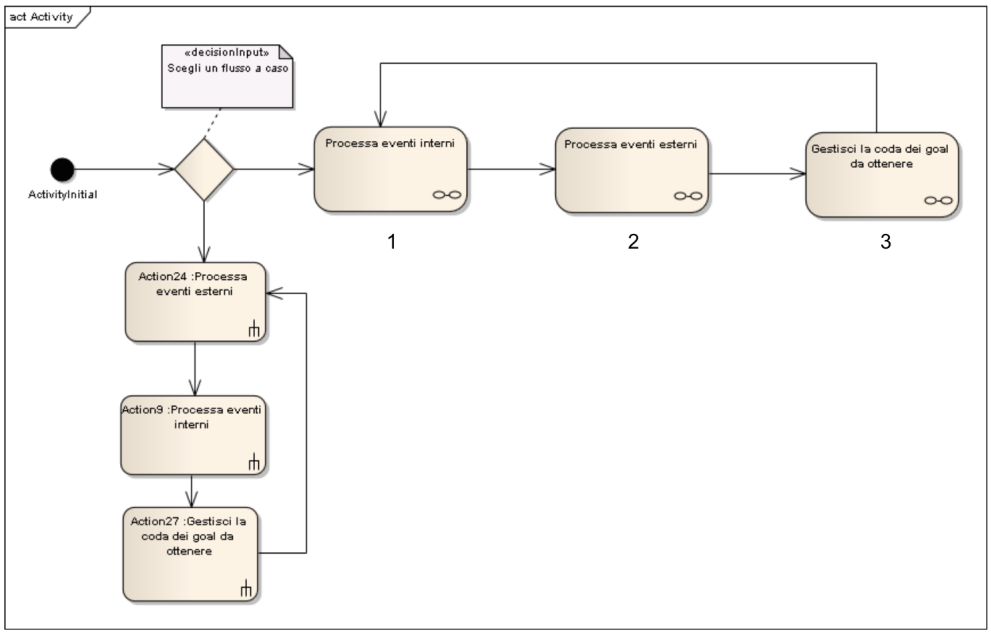
A.15. Write in plv-file the multiple external events as separate events



This diagram represent a sub-activity of the **Inizialization of the agent**. Line 166 the predicate " ass _stringhe _mul" is called in line 1566. The multiple events asserted in memory are created and it put this events in a lists (clause(da _asserire _stringhe _mul)). The predicate "eve" is written in the plv-file. Given an external event EVE, this action writes in plv-file (line 1573):

eve(EVE):- true.

B. Interpret the agent



This is a macro-activity associated at the Use Case **" Interpret the agent"**. After the initializzation the agent is active. In line 177 there is a predicate **"go"**, this predicate is called in line 1510.

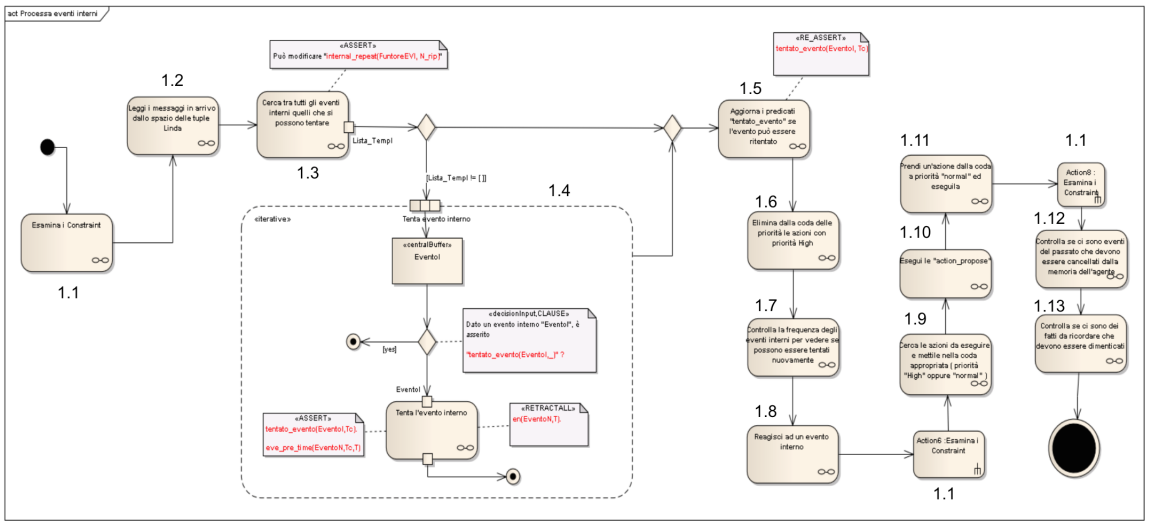
```
go:-repeat, (pari,side_goal ), fail,!.
```

The predicate " pari ", that represent this diagram substantially, is called in lines 1499-1500.

```
pari:-sleep(1), random(1,10,R), R1 is R mod 2,
if(R1=0,(internal,external), (external,internal)).
```

The branch (internal,external) will be analyzed then **" Processa eventi interni " (B.1)**, **" Processa eventi esterni " (B.2)**, **"Gestisci la coda dei goal da ottenere " (B.3)**. Each of these activities will be described with activities diagram.

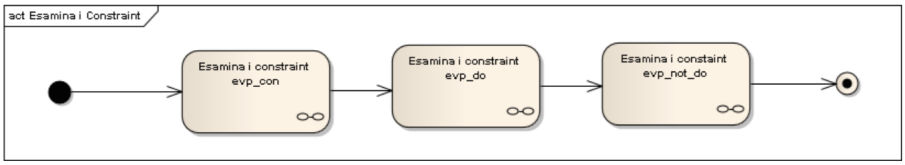
B.1. Process Internal Events



This diagram is represented by the predicate **“ internal ”** in line 1502.

```
internal:- blocco_constr, ricmess, ev_int, ev_goal,
ev_int0, blocco_numero_ev_int, blocco_frequenza, ev_int2,
controlla_freq_tent, svuota_coda_priority,
controlla_freq_iv, scatena, blocco_constr, keep_action,
execute_do_action_propose, prendi_action_normal,
blocco_constr, controlla_vita.
```

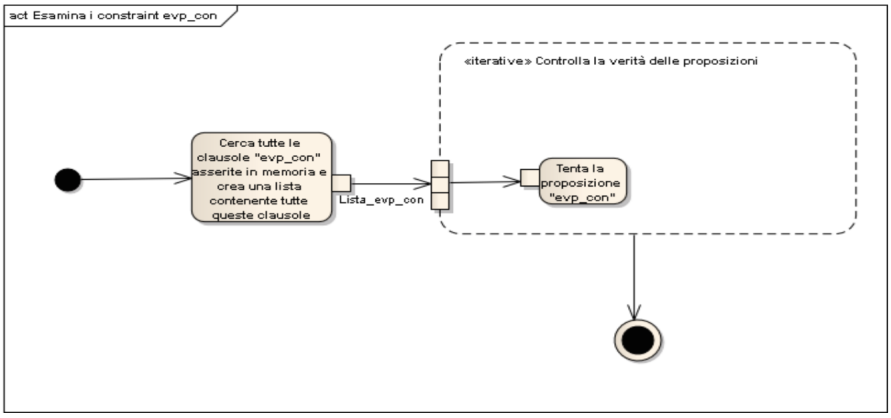
B.1.1. Examine the Constraints



This diagram is a sub-activity of the activity **“Process internal events ”** and it is represented in line 1513 by the predicate “ blocco _constr ” :

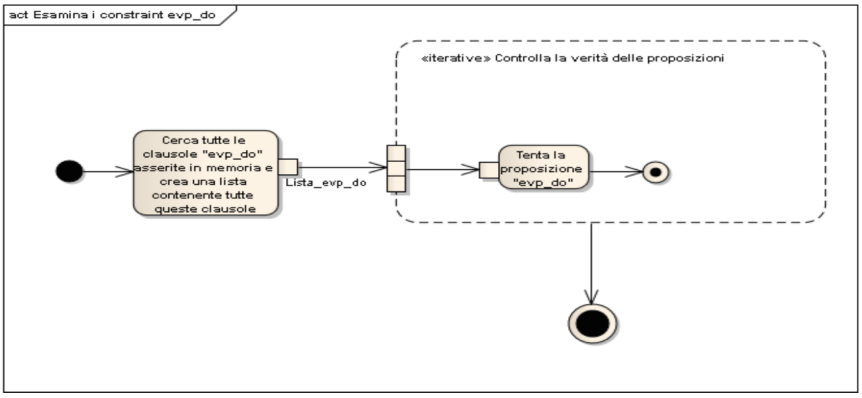
```
blocco_constr:- evaluate_evp_constr, evaluate_evp_do,
evaluate_evp_not_do.
```

B.1.1.1. Examine the Constraints evp _con



This diagram is sub-activity of **“ Examine the Constraint ”** and it represent to predicate “ evaluate _evp _constr ”. This predicate is called in file **examine _past _constraints.pl**, line 128. All the clause “evp _con”, asserted in memory, is searched and a list with all these clause is created. Subsequently it is verified the veracity of propositions and the proposition “evp _con” is attempted.

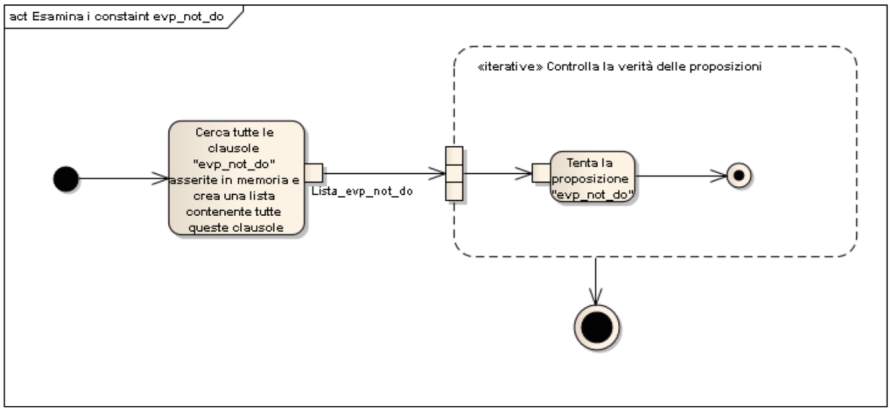
B.1.1.2. Examine the Constraints evp _do



This diagram is sub-activity of **“ Examine the Constraint ”** and it represent to predicate “ evaluate _evp _do ”. This predicate is called in

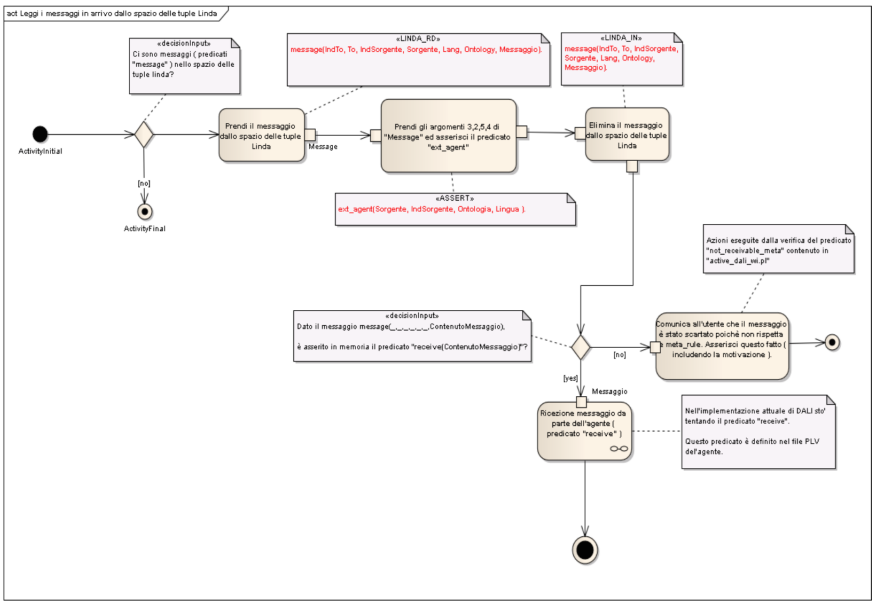
file **examine _past _constraints.pl**, line 324. All the clause “evp _do”, asserted in memory, is searched and a list with all these clause is created. Subsequently it is verified the veracity of propositions and the proposition “ evp _do ” is attempted.

B.1.1.3. Examine the Constraints evp _not _do



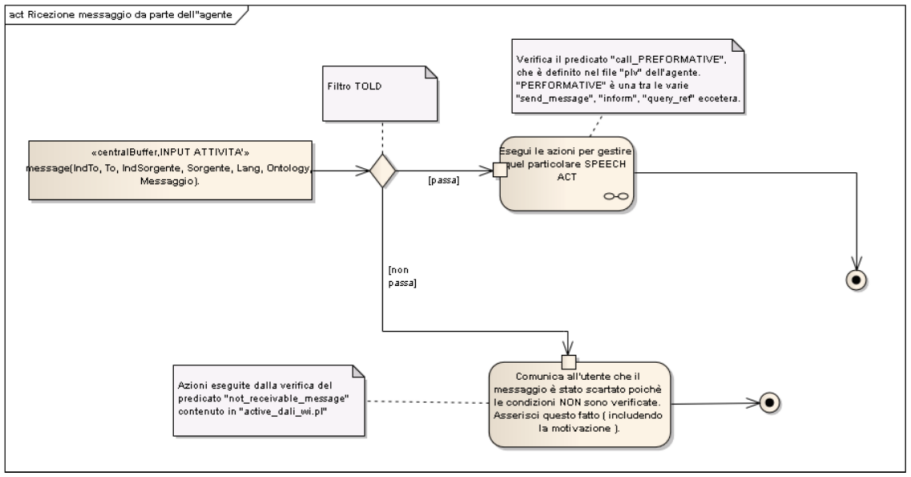
This diagram is sub-activity of **“ Examine the Constraint ”** and it represent to predicate “ evaluate _evp _not _do ”. This predicate is called in file **examine _past _constraints.pl**, line 226. All the clause “evp _not _do”, asserted in memory, is searched and a list with all these clause is created. Subsequently the veracity of propositions is verified and the proposition “evp _not _do” is attempted.

B.1.2. Read the arrived message from the space of Linda's tuple



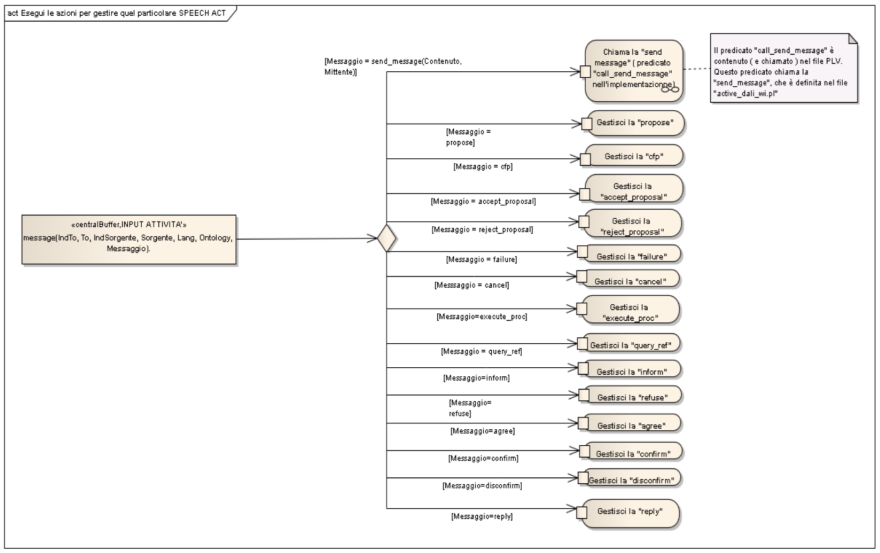
This diagram is a sub-activity of the activity **"Process internal events "** and it is represented in line 684 by the predicate " ricmess ". The question is: are there message(message predicates) in the space of Linda's Tuple? If yes, in line 685 the predicate "rd _noblock" is called in file **client.pl** line 118(under the directory PrologReference/library/linda). Subsequently the predicate "ricmess0" is called in line 689. It is taken the arguments 3,2,5,4 of Message and in line 690 calling to the predicate " assecosa(ext _agent(AgM, IndM, Ontology, Language))" where the predicate " ext _agent " is asserted. In line 691 the predicate "in _noblock" is called in file **client.pl** line 138(under the directory PrologReference/library/linda), where the message of the space of Linda's Tule is elimineted. Line 693, given a message , is the predicate " receive(ContenutoMessaggio) " asserted in memory? If yes, the agent receives the message(predicate " chiama _con ",line 695 - see diagram B.1.2.1). Else it is communicated to the agent that the message has been discarded as it does not respect the meta _rule (predicate " not _receivable _meta ", line 693-740).

B.1.2.1. Receiving the message from the agent



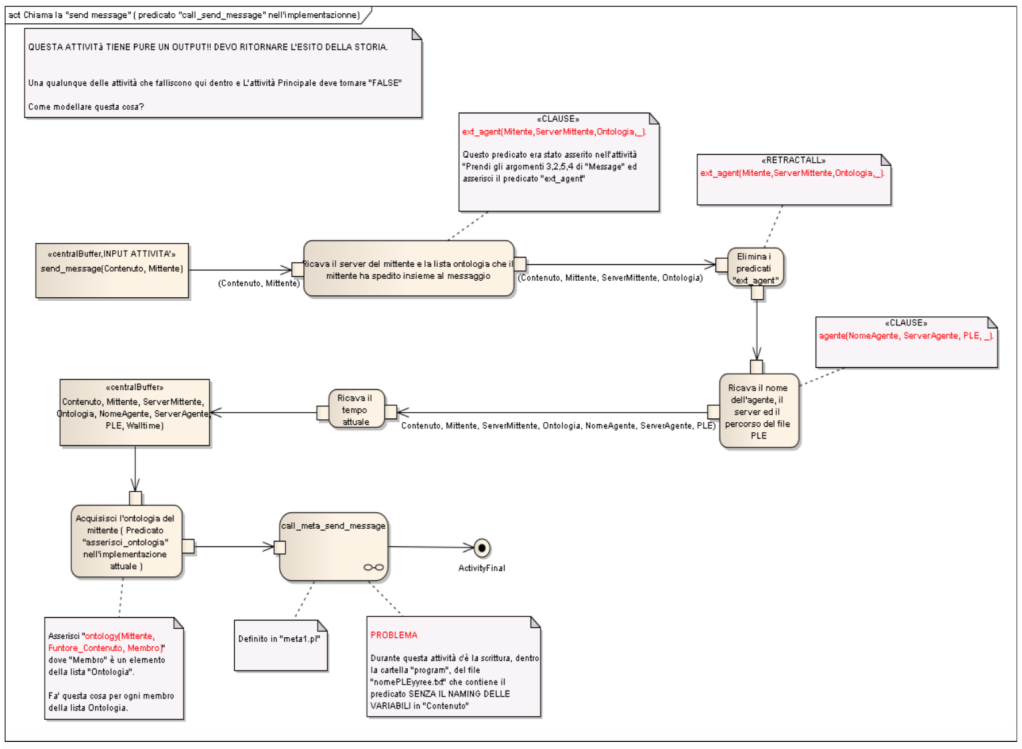
This diagram is a sub-activity of the activity **"Read the arrived message from the space of Linda's tuple "** and it is represented in line 695 by the predicate " chiama _con ". The TOLD filter is activated in the plv-file. If the filter has the positive result, the actions for manage the SPEECH ACT are executed(see diagram B.1.2.1.1.). Else it is communicated to the agent that the message has been discarded as it does not respect the meta _rule (predicate " not _receivable _meta ", line 740).

B.1.2.1.1. The actions for manage the SPEECH ACT are executed



This diagram is a sub-activity of the activity **"Receiving the message from the agent "**. The predicate received in the plv-file can be contain various types of messages: propose, cfp, accept _proposal, reject _proposal, failure, cancel, execute _proc, query _ref, inform, refuse, agree, confirm, disconfirm, reply or send _message(Content, Sender). The predicate "Call _send _message" is contained in plv-file. This predicate calls the "send _message" line 698(B.1.2.1.1.1).

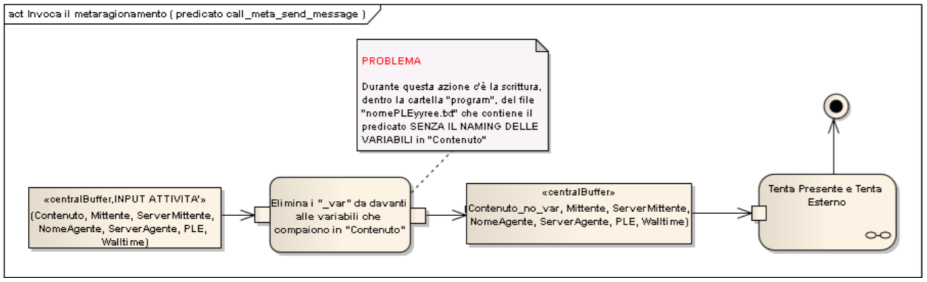
B.1.2.1.1.1 Call the "send message "



This diagram is a sub-activity of the activity **"The actions for manage the SPEECH ACT are executed"**. Line 698, predicate "send _message", the sender's server and the ontology that the sender has sent are derived(line 698). In line 699, the predicates "ext _agent" is eliminated (retractall ext _agent). Line 700, the name of the agent, the server and the path of ple-file are derived. In line 706 the time is derived with the predicate "statistics".

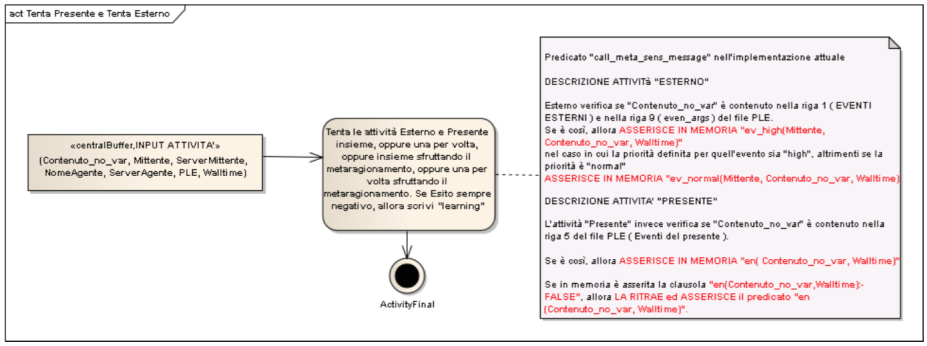
Line 702-729 the ontology of the sender is derived (predicate “asserisci _ontologia”). In line 709 the predicate “call _meta _send _message” is called. This predicate is defined in B.1.2.1.1.1.1.

B.1.2.1.1.1.1. Invoke the meta-reasoning



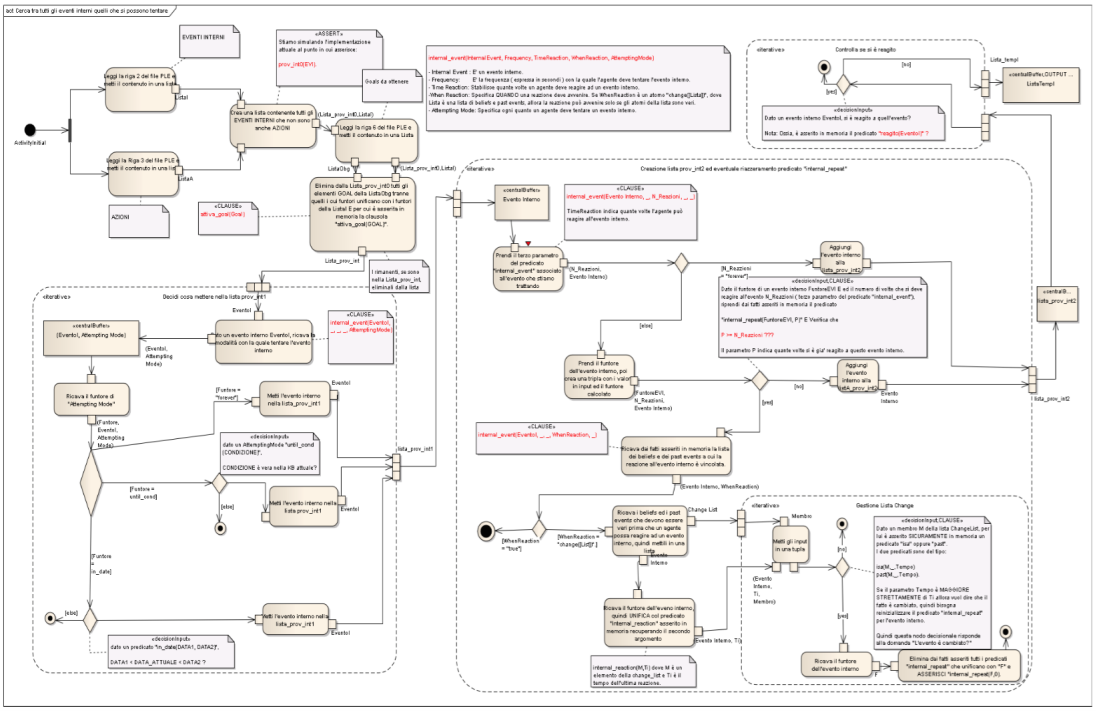
This diagram is a sub-activity of the activity **“Call the send message ”**. The predicate “call _meta _send _message ” is called in file **meta1.pl**, line 27. The “_var ” is eliminated from the front of variables that appear in “Content ”. Subsequently the predicate “chiama _send _message” is called in line 33. Finally present and external activities are attempted (see diagram B.1.2.1.1.1.1.).

B.1.2.1.1.1.1.1. Present and external activities are attempted



This diagram is a sub-activity of the activity **“Invoke the meta-reasoning”**. The present and external events are attempted togheter, or one at a time, or togheter exploiting the meta-reasoning, or one at a time exploiting the meta-reasoning. If the result is negative, it is written “learning ”(file meta1.pl, from line 29 to 53).

B.1.3. Search among all the internal events those that can be tried



This diagram is a sub-activity of the activity **“Process internal events ”** and it is represented in line 1041 by the predicate **“ ev _int ”**. The second line in ple-file is read and the content is put in a list. If this list is not empty then the third line of ple-file is read. The list that contains all the external events that do not the actions also, from line 1045 to 1064. Line 1502 the predicate **“ ev _goal ”** is called in line 1066 (clause(prov _int0(Me))). Line 1068 predicate “ev _goal(L) ”: the sixth line of ple file is read. The predicate “clause(obtgoal(X),true)” is called in line 1070. If the list X is empty the predicate “passa _a _prov _int” is executed, else the predicate “ev _goal21(X,L)” is called, where X is the list obtained from the file and L is the list containing the internal events. All the elements goal present in ListObg are eliminated from the list L, except those functors unify with functors of the ListI for which the clause “attiva _goal(GOAL)” is asserted

in memory(line 1087). Line 1502, the predicate "ev _int0" is called in line 1099. It is arrived to the line 1105 where the predicate "internal _event(Me, _, _, _,C), _)" is called (given an internal event, it is obtained the modality with which to try the internal event). In line 1106 the predicate "functor(C, F, _)" is called and the functor of "Attempting mode" is obtained and the controls on the functors are started.

If the funtor is ugual to FOREVER, line 1153 is executed and the internal event is put in list *prov _int1*.

If the functor is equal to UNTIL _COND the predicate "esamina _cd", line 1112, is executed. If the condition is true in the current KB then line 1153 is executed and the internal event is put in list *prov _int1*. Else the predicate "scarica(Me)" is executed in line 1155.

If the functor is equal to IN _DATE, the predicate "esamina _dt" is executed (from line 1115 to 1149 it checks on time is carried out) and the internal event is put in list *prov _int1*. Else a message of error is written, line 1111.

In the second block a new list,*prov _int2*, is created and there is a possible redirection to the predicate "internal _repeat".

The predicate **"blocco _numero _ev _int"** , line 1502, is called in line 1159. If the list *prov _int1* is not empty the predicate "blocco _numero _2" is called in line 1163. In line 1166 the third value of internal event, if the funtor is equal to FOREVER, the event is added to the new list *prov _int2*, line 1167. Else the predicate "check _time _rep" is executed in line 1170.

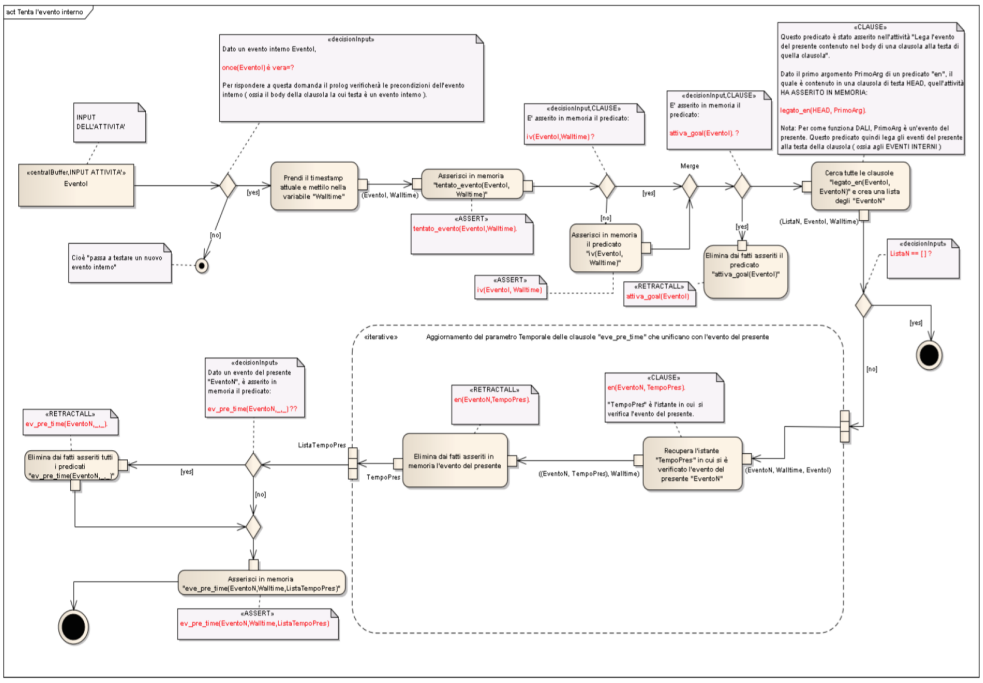
Given a funtor of internal event F and the number of times you must react to the event N _Reaction. The question is : $P \geq N$ _Reaction? , where P is the number of times that it is reacted at the internal event. If the answer is no, the event is addded to the list *prov _int2*, else the predicate "check _cond _repeat" is called in line 1177. The fouth parameter is derived (C_n, WhenReaction). It is derived the list of beliefs and past events to which the reation to the internal event refers.

If the C_n is true then end, else the predicate "check _cond _repeat1" is called, line 1180.

If the C_n is equal to change, it is derived the list of beliefs and past events that

must be true before the agent reacts to the internal event, and puts them on a list. If this list is not empty in line 1184 the predicate "check _cond _change" is called: the functor is derived by the internal event then UNIFIES with the predicate "internal _reaction(F,T_i)" (where F is an element of changeList and T_i is the time of the last reaction), recovering the second argument.
Line 1197, predicate " clause(isa(M, _, _))" is called. Given a member M of changeList, for this it is asserted in memory a predicate "isa" or "past". If the time parameter is greater than T_i, then the fact is changed and it has to initialize the predicate "internal _repeat". Line 1193 the predicate "fact _evp _changed" is called: all predicates "internal _repeat" that unifies with F are eliminated from asserted facts, and it is asserted the predicate "internal _repeat(F,0)".
In the third block the predicate "blocco _frequenza", line 1502, is called. The input is the list *prov _ int2*. In line 1212 the predicate "blocco _frequenza1" is called, if the result of the findall, L , is not empty the predicate "blocco _frequenza2" is called, and the clause(reagito(Me,_)) is called. Given an internal event I,did you react to that event? If yes it is ended, else the list ListaTempI is the output.

B.1.4. Try the Internal Event

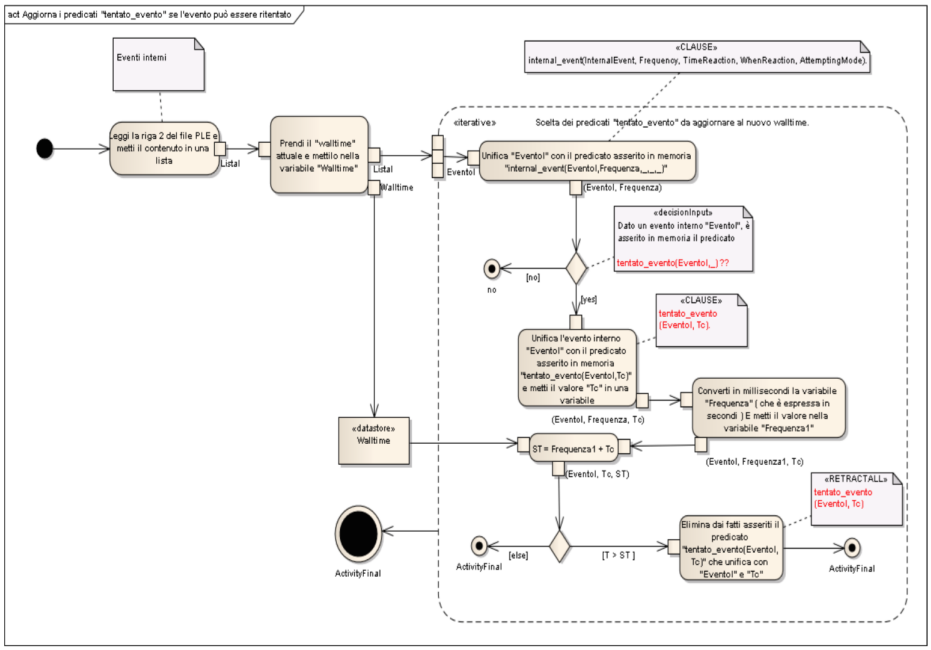


This diagram is a sub-activity of the activity **“Process internal events ”** and it is represented in line 1225 by the predicate **“ ev _int2 ”**. Given an internal event, is once(EventoI) true? If no a new internal event is tempted. Else the current time-stamp is put in variable,“walltime”. The predicate “ev _int _p _no _tent” is called, line 1235. The predicate “tentato _evento ” is asserted in memory, line 1237. The question is : is the predicate “iv(EventoI, Walltime)” asserted in memory? If no this predicate is asserted, else the question is: is the predicate “attiva _goal(EventoI)” asserted in memory? If yes this predicate is eliminated from asserted facts, line 1241.

Line 1242, the clause *legato _ en* is found and a list of “EventoN ” is created, ListN. If ListN is not empty there is an update of temporal parameter of clause “eve _pre _time ” that unifies with the present event. In line 1245 it is recovered the time in which the present event “EventoN” occurred. Line 1247 , T_1 is the instant in which the present event occurred. In line 1249 the present event is eliminated from the asserted facts. Given a present event “EventoN”, is the predicate “eve _pre _time (EventoN, _, _)” asserted in memory? If yes, all predicates “eve _pre _time (EventoN, _, _)” are

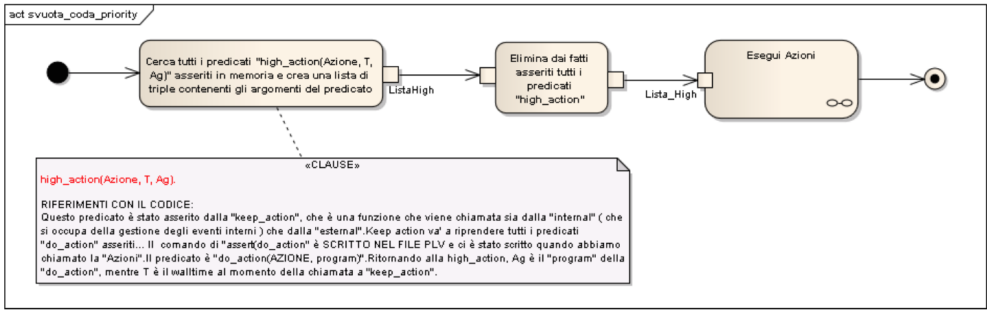
eliminated from the asserted facts, line 1255. Else the predicates “eve _pre _time (EventoN, Walltime , ListaTempoPres)” are asserted in memory, line 1256.

B.1.5. Update the predicates “ tentato _evento” if the event can be retried



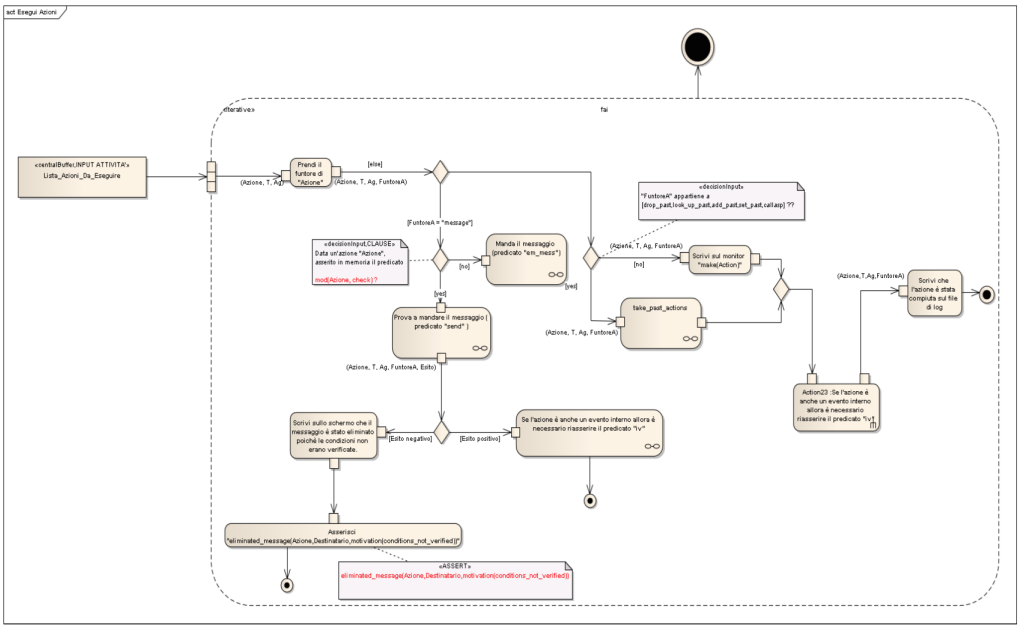
This diagram is a sub-activity of the activity **“Process internal events ”** and it is represented in line 1312 by the predicate **“ controlla _freq _tent”**. The current walltime is put in a list. The second line of ple-file is read and the content is put in a list. In line 1320 the event “EventoI” unifies with the predicate “internal _events(EventoI, Frequenza,_, _,_) ”. Given an internal event “EventoI” , is the predicate “tentato _evento(EventoI,_)” asserted in memory? If yes, in line 1324 the internal event “EventoI”unifies with the predicate asserted in memory “tentato _evento(EventoI,T_c)” and T_c is put in a variable. In line 1326 the variable “Frequenza” is converted in milliseconds and this value is put in a variable “Frequenza1”. In line 1327 it is executed ST= *Frequenza1* + T_c. If T > ST then the predicate “tentato evento(EventoI,T_c)” that unifies with “EventoI” and “ T_c” is eliminated from the asserted facts, line 1329.

B.1.6. The actions with priority “High” are deleted from priorities queue



This diagram is a sub-activity of the activity **“Process internal events ”** and it is represented in line 925 by the predicate **“ svuota _coda _priority”**. The predicate “clause(high _action(Azione, T, Ag)) ” searches all predicates “high _action ” asserted in memory and creates a list of triplet containing the arguments of predicate. In line 934, predicate “ svuota _coda _priority1” is called: all predicates “ high _action ” are eliminated from asserted facts. In line 935 the actions are executed (B.1.6.1.).

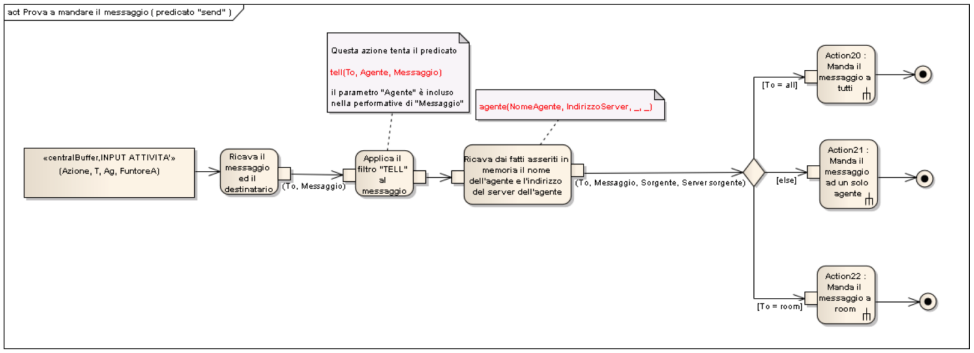
B.1.6.1. Execute Actions



This diagram is a sub-activity of the activity **"The actions with priority High are eliminated from priorities queue"** and it is represented in line 952 by the predicate "fai(X, T, Ag)" and the functor of Action is taken. If the functor is equal to message the predicate "em _mess0" in line 973. The question is: given an action "Azione", is the predicate "mod(X, check)" asserted in memory? If no the predicate "em mess0" is executed in line 977 (B.1.6.1.2). Else the predicate "check _msg" is executed and it is tried to send a message (B.1.6.1.1). If the outcome is positive the predicate "ver _az _int" in line 1030: if the action is the internal event also then it is necessary reasserted the predicate "iv" (B.1.6.1.3). If the outcome is negative the predicate "not _sendable _message " is executed: it is written that the message is eliminated because the conditions are not verificated. In line 980 the predicate "eliminated _message" is asserted.
In line 952 the predicate "choose _action" is called: the functor is obtained and it is verified if the functor allows to drop _past, look _up past, set _past, collapse. If yes the predicate "take past action" is called (B.1.6.1.4). Else it is written in the monitor *make(Action)*, line 955. Finally it is called the predicate "save _on _log _file" in line 955: it is written on the log file

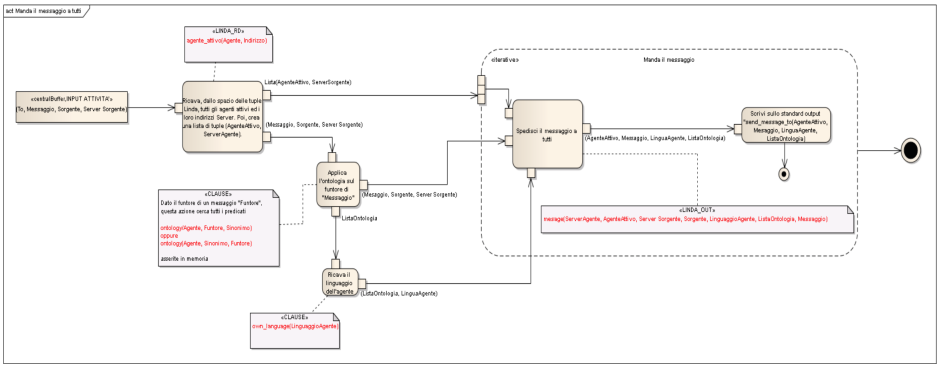
that the action is complete.

B.1.6.1.1. Try to send a message(predicate " send ")



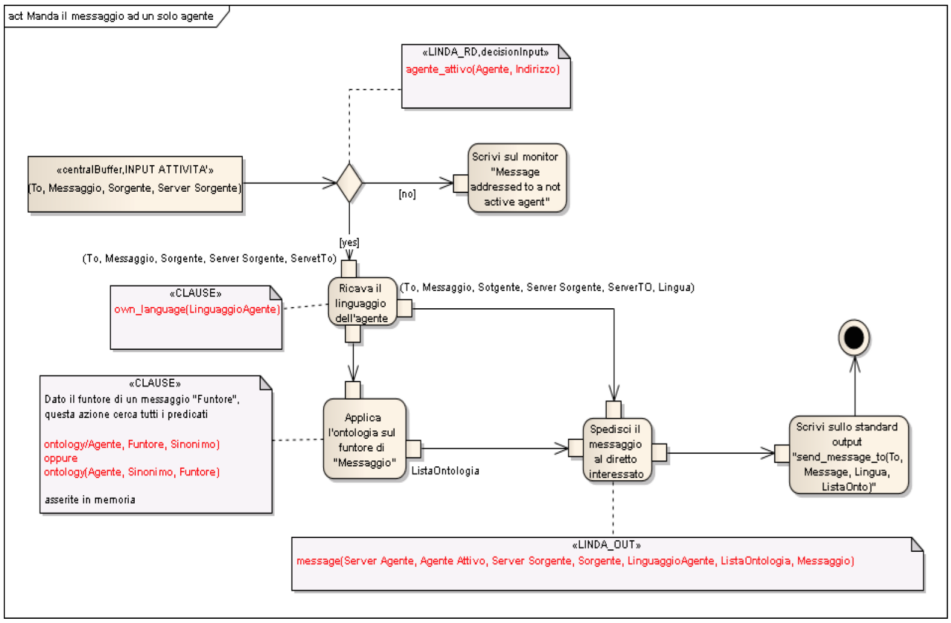
This diagram is a sub-activity of the activity "Execute Actions" and it is represented in line 974 by the predicate "check _msg". In line 974 the receiver(To) and the message(M) are obtained. The file agent.plv is opened and the Tell filter is applied to the message "send(To, send _message(X,Ag)):- tell(To, Ag, send _message(X)), send m(To, send _message(X, Ag))". The predicate "send _m " in line 984. The name of the agent and the address of the agent's server are obtained by the facts asserted in memory. In line 985 there are checks on the type of receiver.
If the receiver is ugual to all the message is sent to all(B.1.6.1.1.1.).
Else the message is sent to one agent(B.1.6.1.1.2.).
Finally if the receiver is ugual to room the message is sent to room.

B.1.6.1.1.1. The message is sent to all



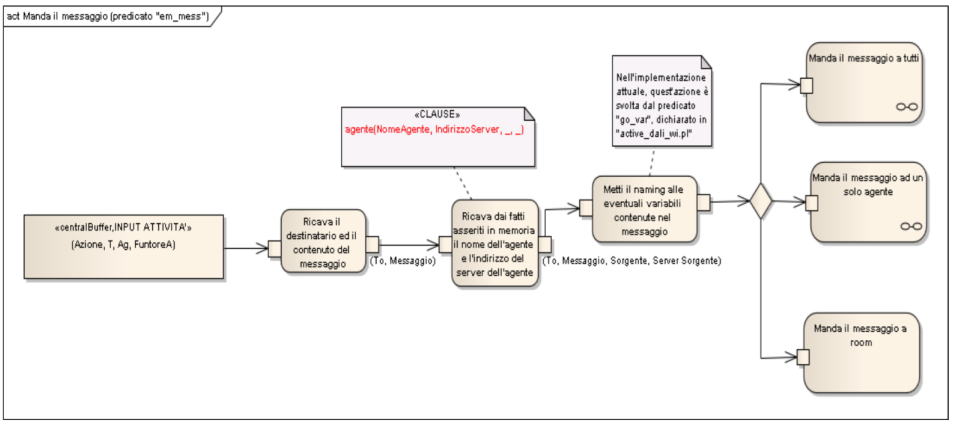
This diagram is a sub-activity of the activity **"Try to send a message(predicate " send ") "** and it is represented in line 1005 by the predicate "manda _a _all(M, S, IndS)", where M is equal to message, S is the name of the agent and IndS is the address of the Server. From line 1005 to 1008, all the active agents and their server addresses are obtained by the space of the Linda's tuple. After a list of tuples(activeAgent, ServerAgent) is created. In line 998 the predicate "invia _terms _ontology(L)" is called: given a functor of message, all predicates " ontology(Agent, Functor, Synonymous)" or " ontology(Agent, Synonymous, Functor)",asserted in memory, are searched. In line 1010, the predicate "own language(AgentLanguage)" are called: the language of the agent is obtained.
In line 1017, the predicate "out(message(IndTo, To, IndS, S, Lang, L, M)" is called (open the file client.pl) and the message is sent to all. In line 1018 it is printed the message "send _message _to(To, M, Lang, L)".

B.1.6.1.1.2. The message is sent to one agent



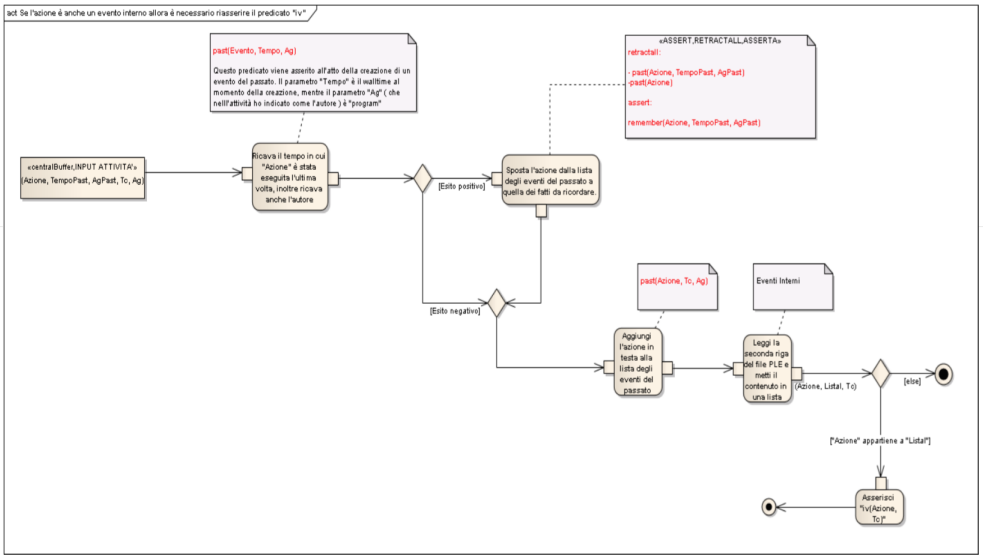
This diagram is a sub-activity of the activity **"Try to send a message(predicate " send ")"** and it is represented in line 987 by the predicate "manda _a(To,M, S, IndS)", where To is the receiver M is equal to message, S is the name of the agent and IndS is the address of the Server. If the agent is active, the predicate "rd _noblock(agente _attivo(To,IndTo))" is called. If the result fail it is printed: "Message addressed to a not active agent". Else the predicate "emetti _ms" is called in line 994: the language of the agent is obtained. The ontology is applied on the functor "message". The message is sent to the interested agent(open the file client.pl, line 996). The predicate "save _on _log _file" is called in line 1993.
If the receiver is equal to room it is executed line from 1022 to 1027.

B.1.6.1.2. Send the message(predicate " em _mess ")



This diagram is a sub-activity of the activity **"Execute Actions"** and it is represented in line 977 by the predicate "em _mess". The content of message and the receiver of the message is obtained and the predicate "send _m0" is called in line 982. The name of the agent and the address of the server are obtained by the facts asserted in memory. The predicate "go _var" is called (file meta1.pl, line 104): the NAMING is put into the variables in the message. The predicate "send _m" is called in line 984. In line 985 there are checks on the type of receiver.
If the receiver is ugual to all the message is sent to all(B.1.6.1.1.1.).
Else the message is sent to one agent(B.1.6.1.1.2.).
Finally if the receiver is ugual to room the message is sent to room.

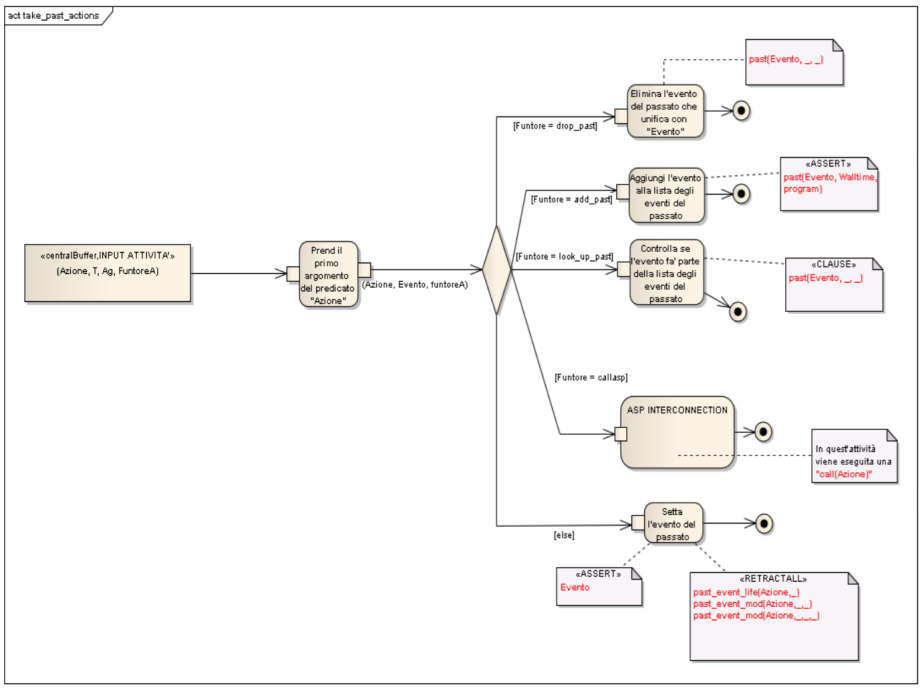
B.1.6.1.3. If the action is the internal event also then it is necessary reasserted the predicate "iv"



This diagram is a sub-activity of the activity **"Execute Actions"** and it is represented in line 976-1030 by the predicate "ver _az _int". The predicate "divP" is called in line 1358. The predicate "past(Event, T_r, Ag)" is asserted when the past event is created. The parameter T_r is the walltime at the moment of creations, while A is program. It is obtained the time in which "Azione" was last performed and also derived the author. If the clause has the positive result the predicate "sposta _in _rem" is called in line 1360: the actions is moved from the list of the past events to the list of facts to remember. If the clause has the negative result the predicate "asserta(past(Me,T_r, A_g))" is called in line 1358: the action is added in the head of the past events.

The predicate "ver _az _int" is called in line 1030. The second line in ple-file is called and the content is put in a list, L. If L is not empty the predicate "ver _az _int1" is called in line 1035 and the "iv(X, T_c)" is asserted, where X is Action.

B.1.6.1.4. Take _past _actions



This diagram is a sub-activity of the activity **"Execute Actions"** and it is represented in line 955-957 by the predicate "Take _past _actions". The first argument(X=Action) is taken and the predicate "caso0 _past(F,E,X)" is called in line 958.

If the functor F is equal to drop _past the predicate "drop _evento(X)" is called in line 967: the past event that unified with " Evento " is eliminated (retractall(past(X,_, _))).

If the functor F is not equal to drop _past the predicate "caso1 _past(F,E,X)" is called in line 959.

If the functor F is equal to add _evento the predicate "add _evento(X)" is called in line 968: the event is added to the list of the past event.

If the functor F is not equal to add _evento the predicate "caso2 _past(F,E,X)" is called in line 960.

If the functor F is equal to look _up _past the predicate "look _up _evento(X)" is called in line 969: it is checked if the events belong to the list of past events.

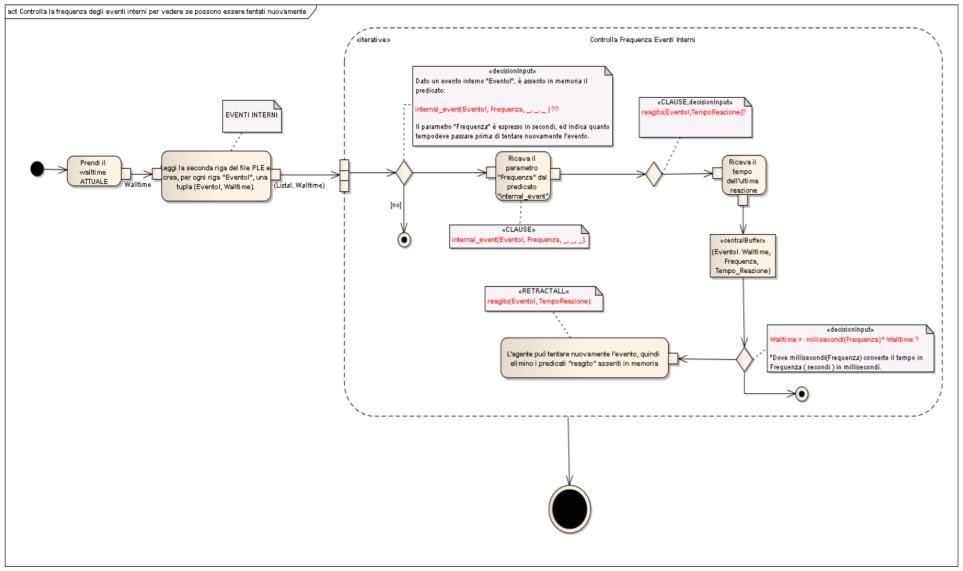
If the functor F is not equal to look _up _past the predicate "caso3 _past(F,E,X)"

is called in line 963.

If the functor F is equal to callaps the predicate “callasp” is called in line 1747: ASP-INTERCONNECTION. The predicate “call(Y)”, where Y is Action , is called.

Else the predicate “set _past _evento” is called in line 970. The past event is set: retractall of the predicates “past _event _life(X,_)”, “past _event _mod(X, _, _)”, “past _event _mod (X, _, _, _))”, where is Action.

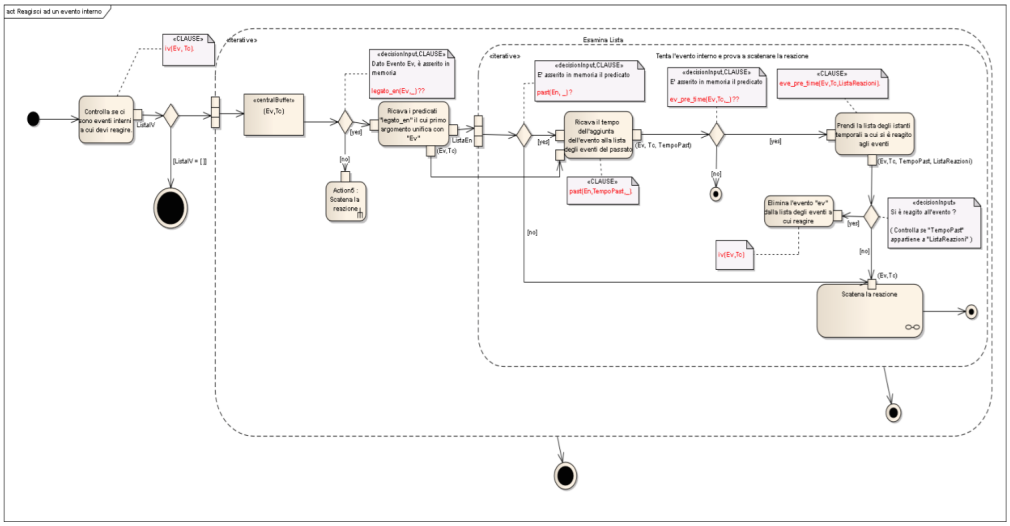
B.1.7. Check the frequency of the internal events to see if they can be retried



This diagram is a sub-activity of the activity **“Process internal events ”** and it is represented in line 1331 by the predicate **“ controlla _freq _iv”**. The current walltime is taken in line 1331. The second line of ple-file is read and for each line “EventoI” is created a tuple(EventoI, Walltime). If L is not empty the predicate “ contr _freq _iv0 ” is called in line 1336. Given an internal event “EventoI”, the question is: is the predicate “internal _event (EventoI, Frequency, _,_,_)” asserted in memory? The Frequency parameter is expressed in seconds and indicates how much time must pass before attempting the event again. If yes, the parameter Frequency is ob-

tained from the “clause(internal _event(X, Tp, _, _, _))”, line 1339. The predicate “controlla _freq _iv1” is called in line 1340. If “clause(reagito(X, Tc))” is true the predicate “controlla _freq _int2” is called in line 1349: the time is transformed into milliseconds and ST = Tp1+Tc. If T>ST then the predicate “canc _e _reagito” is called in line 1352: the agent can try the event again the the predicates “reagito”, asserted in memory, are eliminated .

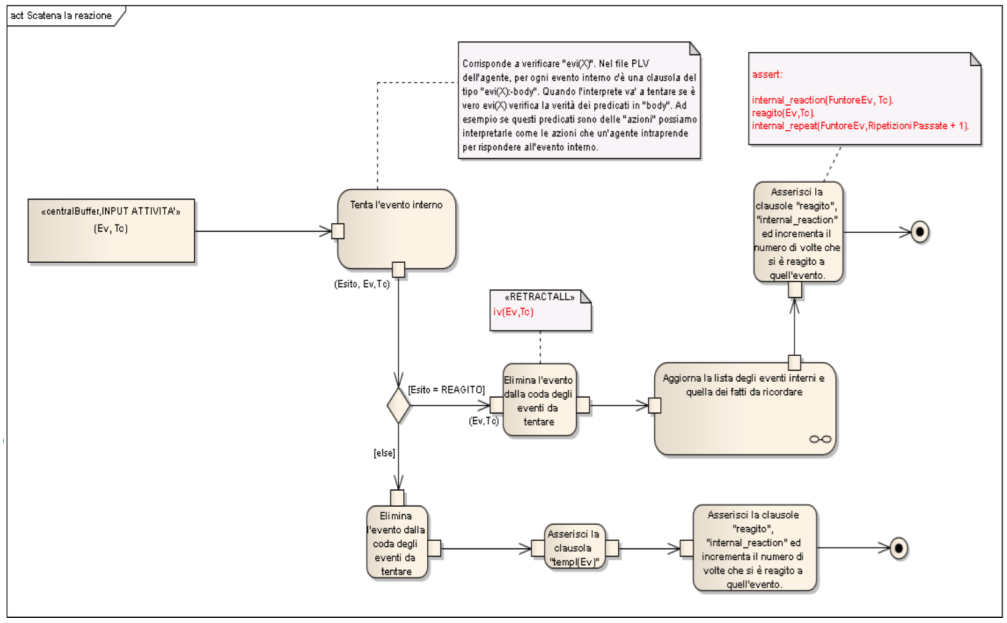
B.1.8. React to the Internal Event



This diagram is a sub-activity of the activity **“Process internal events ”** and it is represented in line 1267 by the predicate **“ scatena ”**. It is checked if there are Internal events to react to and the predicate “ manage list int(L) ”, where L is a list of internal events, is called in line 1268. If L is not empty the predicate “ scat0 ” is called in line 1275. Given an event Ev, is the predicate “ legato _en(Ev, _) ” asserted in memory? If no the predicate “ scat1 ” is called in line 1290(B.1.8.1.).
If yes the predicate “ scat0 _en ” is called in line 1277: it is derived the predicates “ legato _en ” whose first argument unifies with Ev. The question is: is the predicate “ past(X, T1, _) ” asserted in memory? If no it is called line 1290(B.1.8.1.).

If yes it is obtained the time of adding the event to the list of past events and the predicate “ check _app ” is called in line 1284. The question is : is the predicate “ eve _pre _time ” is asserted in memory? If yes the list of time instants to which it has reacted is obtained in line 1284. The predicate “ check _member ” is called in line 1284.
If no the predicate “ scat1 ” is called in line 1290(B.1.8.1.).
If yes the predicate “ritrai _ev _pres” is called in line 1288: the event is eliminated from the list of internal event to which it has reacted.

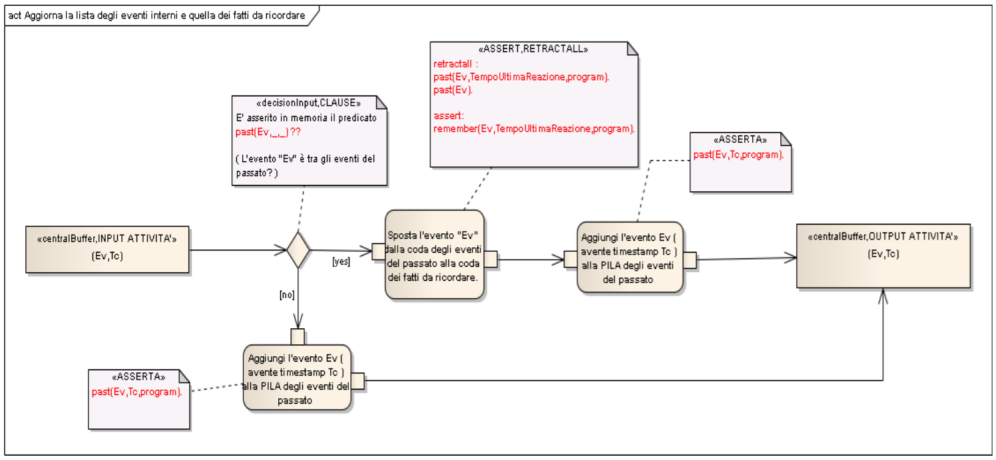
B.1.8.1. Trigger the reaction



This diagram is a sub-activity of the activity **“React to the Internal Event ”** and it is represented in line 1290 by the predicate **“ scat1 ”**. The predicate “evi(X)” is verified and then the internal event is triggered. If the result is equal to REAGITO, the predicate “scatena _reagisci” is called in line 1296: the event is eliminated from the queue of the events to try. The predicate “divP” is called in line 1296: the list of the internal events and the list of the facts to remember are updated (B.1.8.1.1.). The clause “reagito” and

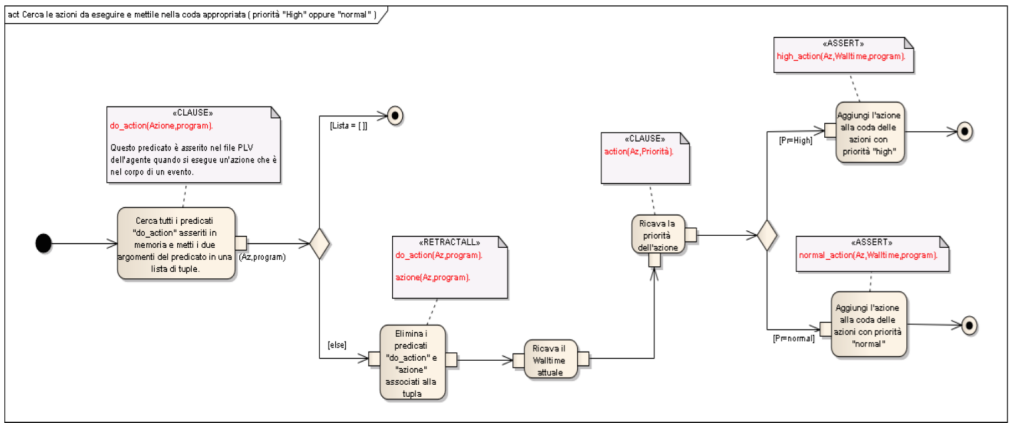
"internal reaction" are asserted and the number of times it has reacted, is increased line 1297-1299.
If the result is not equal to REAGITO, the predicate "no _scatena" is called in line 1291: the event is eliminated from the queue of the events to try. The clause "reagito" and "internal reaction" are asserted and the number of times it has reacted, is increased line 1293.

B.1.8.1.1 The list of the internal events and the list of the facts to remember are updated



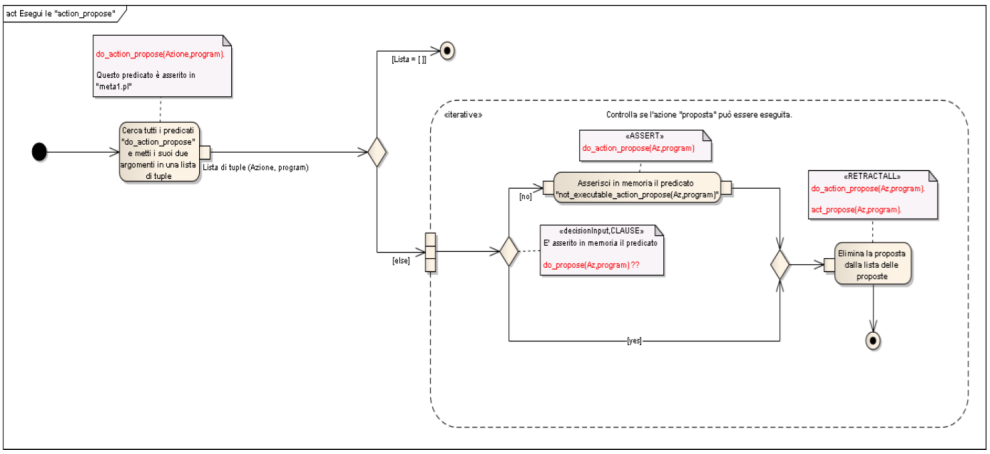
This diagram is a sub-activity of the activity **"Trigger the reaction "** and it is represented in line 1296 by the predicate **" scatena _agisci "**. The question is: is the predicate " past(Ev, _, _) " in line 1358 asserted in memory? If yes the predicate "sposta _in _rem" is called in line 1360: the event Ev is moved from queue of past events to the queue of the facts to remember and the event Ev is added to the stack of the past events.
If no the predicate "asserta(past(Ev, Tc,program))" is called in line 1358: the event Ev is added to the stack of the past events.

B.1.9. Search for the actions to be performed and place them in the appropriate queue



This diagram is a sub-activity of the activity **"Process internal events "** and it is represented in line 906 by the predicate **" keep _action "**. All predicates "do _action " asserted in memory, are found and the two arguments of the predicate are put in a list of tuples(line 906). If the list is not empty the predicate "keep _action1" is called in line 908: the predicates "do _action" and "action" associated with the tuples, are eliminated. The current Walltime is obtained in line 918. The predicate "clause(action(Az,Mod), _)" is called in line 919: it is obtained the priority of the actions(Mod).
If Mod is equal to high then the action is added to the queue of the actions with priority high, line 920.
If Mod is equal to normal then the action is added to the queue of the actions with priority high, line 920.

B.1.10. Execute the action _propose

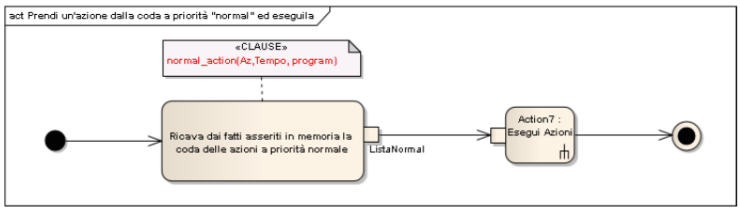


This diagram is a sub-activity of the activity **"Process internal events "** and it is represented in line 711 by the predicate **" execute _do _action _propose "**. All predicates "do _action _propose" are found and the two arguments of the predicate are put in a list of tuples(line 906). This predicate is asserted in file meta1.pl. Is the predicate "do _propose(Az, Ag)" asserted in memory?

If yes the retractall of "do _action _propose(Az, Ag)" and "act _propose(Az, Ag)" are executed, line 722-723.

If no it is asserted in memory "not _executable _action _propose ", line 721.

B.1.11. Take an action from the queue with "normal " priority

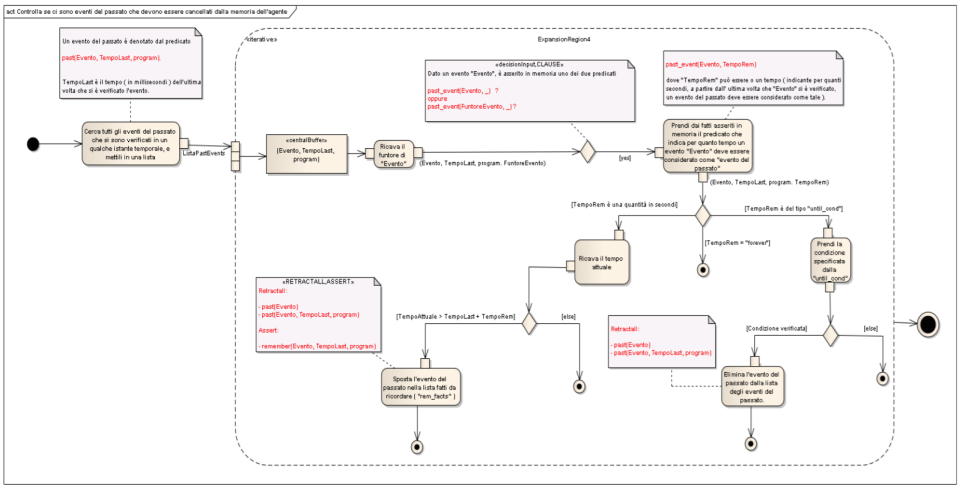


This diagram is a sub-activity of the activity **"Process internal events "** and it is represented in line 940 by the predicate **" prendi _action _normal"**. The queue of actions of normal priority is obtained from the facts asserted in memory. And subsequently the predicate "fai" is called in line 947 (see subsection B.1.6.1.).

B.1.12. Predicate controlla _vita

The predicate **“ controlla _vita”** is described in line 1515 and is a sub-activity of the activity **“Process internal events ”**. This predicate is composed by two predicate: **“ controlla _vita _past _base”** (B.1.12.1.) and **“ controlla _vita _remember”** (B.1.12.2.).

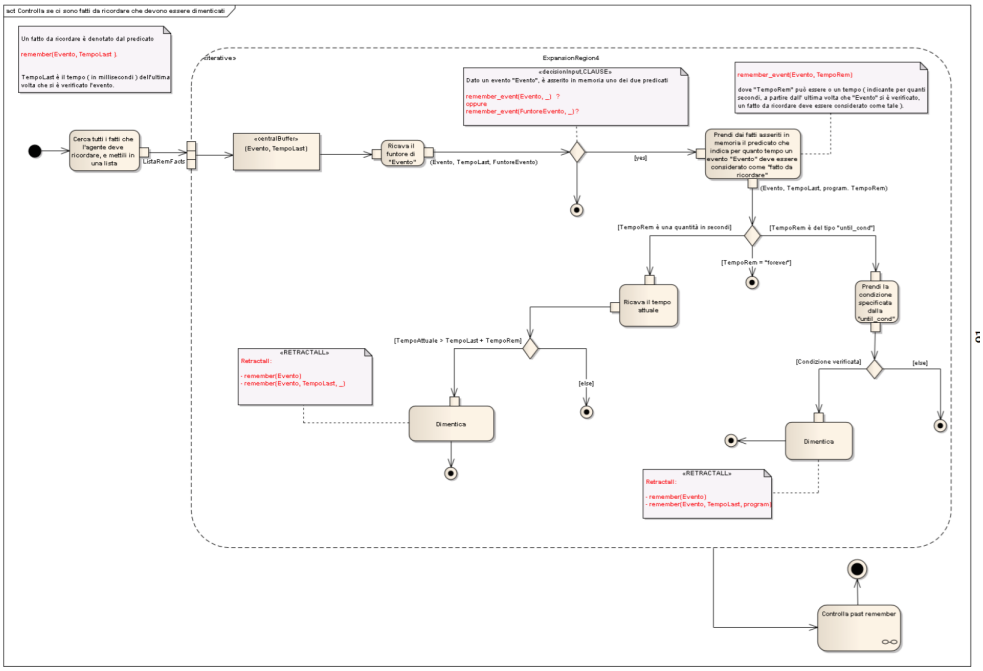
B.1.12.1. Check if there are past events which be deleted from agent's memory.



This diagram is represented in line 1366 by the predicate **“ controlla _vita _past _base”**. All the past events that have occurred at same point in time are searched for and put them in a list. The predicate “ controlla _vita _past ” is called in line 1368 and the functor of event is obtained in line 1371. The question is: given an event is one of the two predicates, “ past _event(Event, _)” or “ past _event(FunctorEvent,_)”, asserted in memory? If yes the predicate “ contrr _past _event1” is called in line 1376 and the predicate “controlla _vita _past1” is called in line 1377. If the functor is equal to forever then end, else the predicate “controlla _vita _past2” is called in line 1378: if the functor is equal to until the predicate “processa _cd _past” is called in line 1390. The past event is eliminated from the list of past events, line 1394. Else the predicate “processa _tempo _past” is called

in line 1380: the past event is moved to the list of facts to remember("rem _fact").

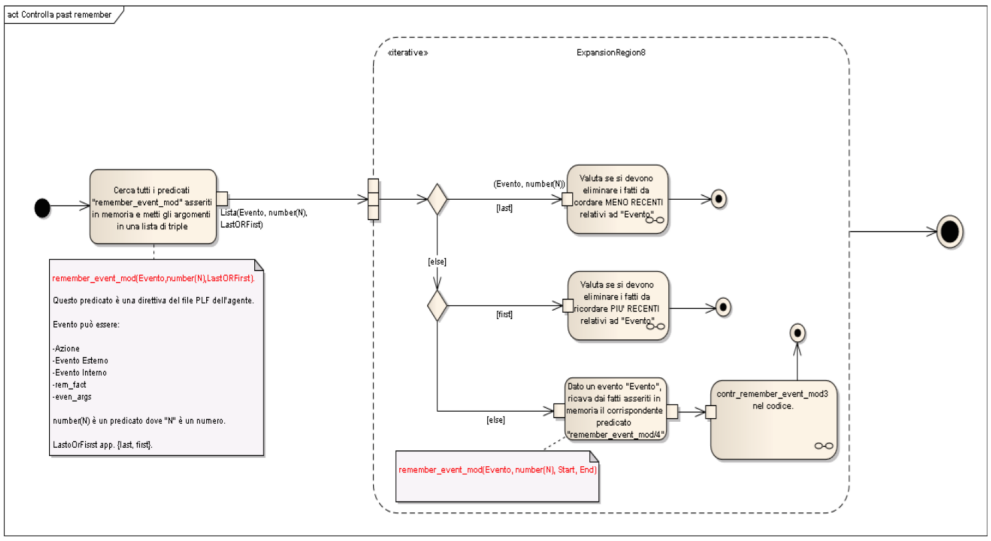
B.1.12.2. Check there are any facts to remember that must be forgotten



This diagram is represented in line 1400 by the predicate **"controlla _vita _remember"**. The predicate "controlla _vita _remember _base" is called in line 1402: all the facts that the agent must be remember are searched for and put them in a list. The functor of "Event" is obtained in line 1410. The question is: given an event "Event" is one of the two predicates, "remember _event(Event, _)" or "remember _event(FunctorEvent,_)", asserted in memory? If yes, the predicate indicating how long an event "Event" must be considered to "fact to remember", is taken from the facts asserted in memory. The predicate "contrr _remember _event1" is called in line 1411-1412 and the predicate "controlla _vita _remember1" is called in line 1416. If the functor is equal to forever the end else the predicate "controlla _vita _remember2" is called in line 1417. If the functor is equal to until the predicate

"processa _cd _remember" is called in line 1427(retractall remember(Event), retractall remember(Event, TempLast, _)). The predicate "processa _tempo _rem" is called in line 1419. If the current time is greater than of the sum between TempoLast plus TempoRem then the predicate "processa _tempo _canc item _rem" is called in line 1424.
The predicate "controlla _past _remember" is called in line 1435(B.1.12.2.1.).

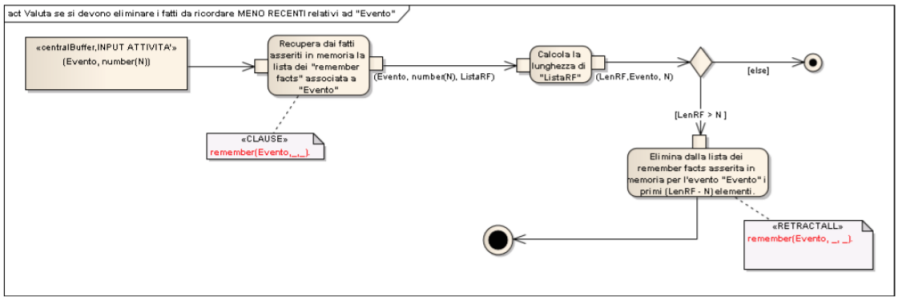
B.1.12.2.1. Controlla _past _remember



This diagram is a sub-activity of the activity **"Check there are any facts to remember that must be forgotten "** and it is represented in line 1435 by the predicate **" controlla _past _remember"**. All predicates "remember _event _mod " asserted in memory are searched and put the arguments in a list of tuples. This predicates is a directive of plf-file of the agent. If the list is not empty then the predicate "controlla _past _mod _rem0" is called in line 1442.
The predicate "clause(remember event mod(Me, number(N), last),)" is true then the predicate "contrr _remember _event _mod1" is called in line 1451: it is evalueted if you need to eliminate the oldest events to remember about "Event"(B.1.12.2.1.1.).

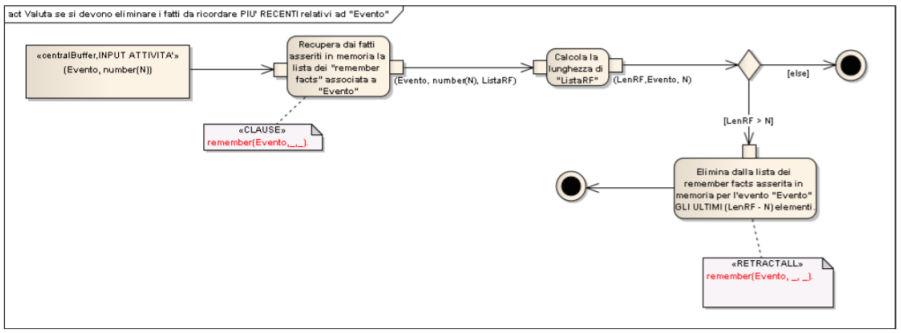
The predicate “clause(remember _event _mod(Me, number(N), first), _)” is true then the predicate “contrr _remember _event _mod2” is called in line 1469: it is evalueted if you need to eliminate the younger events to remember about “Event”(B.1.12.2.1.2.).
Else the predicate “clause(remember event mod(Me, number(N), T1, T2),)” is true then given an event “Event”, it obtains from the facts asserted in memory the corrisponding predicate “remember event mod/4”. The predicate “contrr _remember _event _mod3” is called in line 1487.

B.1.12.2.1.1. Evaluete if you need to delete the oldest events to remember about “Event”



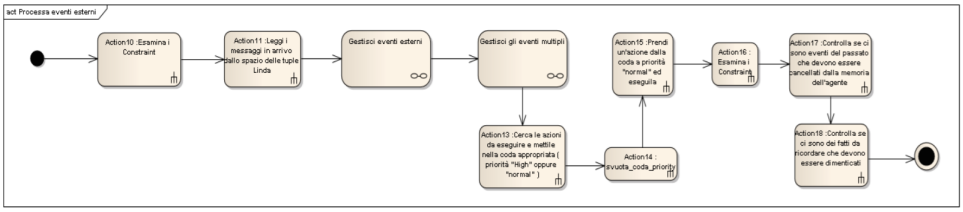
This diagram is a sub-activity of the activity **“Controlla _past _remember”** and it is represented in line 1451 by the predicate **“contrr _remember _event _mod1”**. The list of the “remember facts” associated to “Event” is obtained from facts asserted in memory(clause (remember(P,T,S),_), line 1451). The length of ListRF(List od remember facts) is calculated, line 1452. If the length of ListRF is greater than N then the predicate “eliminate _not _last _items _remember” is called in line 1456: it is eliminated from the remember-facts’ list, asserted in memory for the event “Event”, the first LenRF - N elements.

B.1.12.2.1.2. Evaluete if you need to delete the younger events to remember about "Event"



This diagram is a sub-activity of the activity **"Controlla _past _remember"** and it is represented in line 1469 by the predicate **"contrr _remember _event _mod2"**. The list of the "remember facts" associated to "Event" is obtained from facts asserted in memory(clause (past(P,T,S),_), line 1469). The length of ListRF(List od remember facts) is calculated, line 1470. If the length of ListRF is greater than N then the predicate "eliminate _not _first _items _remember" is called in line 1471: it is eliminated from the remember-facts' list, asserted in memory for the event "Event", the last LenRF - N elements.

B.2. Process External Events



This diagram is represented by the predicate **" external "** in line 1505.

```
internal:- blocco_constr, ricmess,processa_eve,
examine_mul, keep_action, svuota_coda_priority,
prendi_action_normal, blocco_constr, controlla_vita.
```

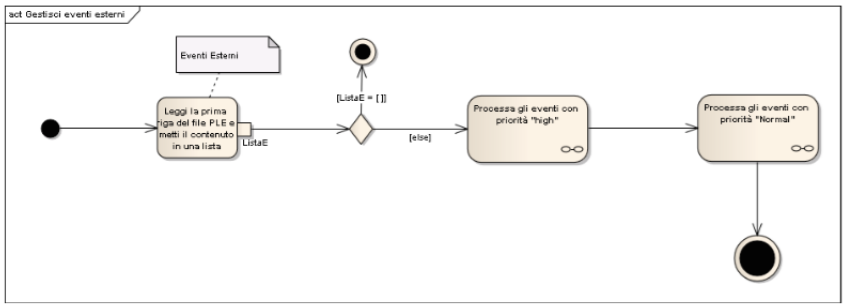
B.2.1. Predicate blocco _constr

The predicate **"blocco _constr"** is the same predicate defined in section B.1.1.

B.2.2. Predicate ricmess

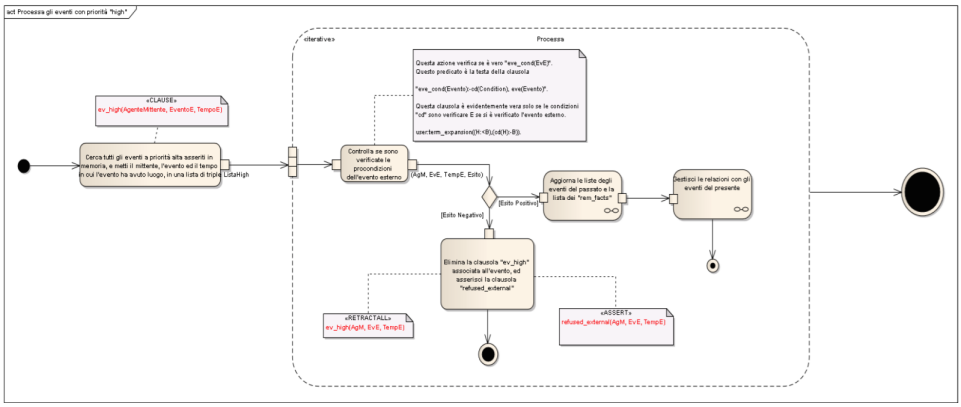
The predicate **"ricmess"** is the same predicate defined in section B.1.2.

B.2.3. Manage the External events



This diagram is a sub-activity of the activity **"Process External Events "** and it is represented in line 746 by the predicate **" processa _eve "** : the first line in ple-file is read and it is put them in a list, Es. If Es is not empty the predicate "processa _eve _high "(B.2.1.1.) and "processa _eve _normal" (B.2.1.2.) are executed .

B.2.3.1. Process the events with high priority

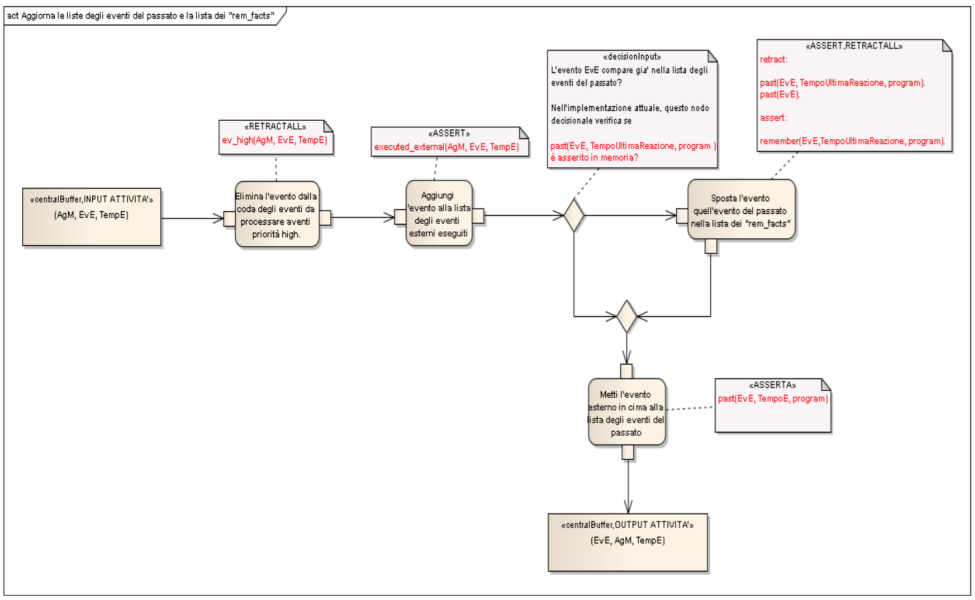


This diagram is a sub-activity of the activity **"Manage External events "** and it is represented in line 749 by the predicate **" processa _eve _high "** . All events with high priority asserted in memory are searched and it puts the event, the sender, and the time in which it took place, in a list of tuples, ListHigh. The predicate "processa _eve _high1" is called in line 750.

In line 755 it checked if the condition of external event is verified: this action verifies if the condition "eve _cond(EVE)" is true (this predicate is in plv-file). If the result is negative the predicate "no _proc _eve _high" is called in line 757: it asserts the clause "refused _external".

If the result is positive the predicate "processa _eve _high/3" is called in line 758: it updates the list of past events and the list of "rem _facts" (B.2.3.1.1.) and it manages the relationship with the present events(B.2.3.1.2.).

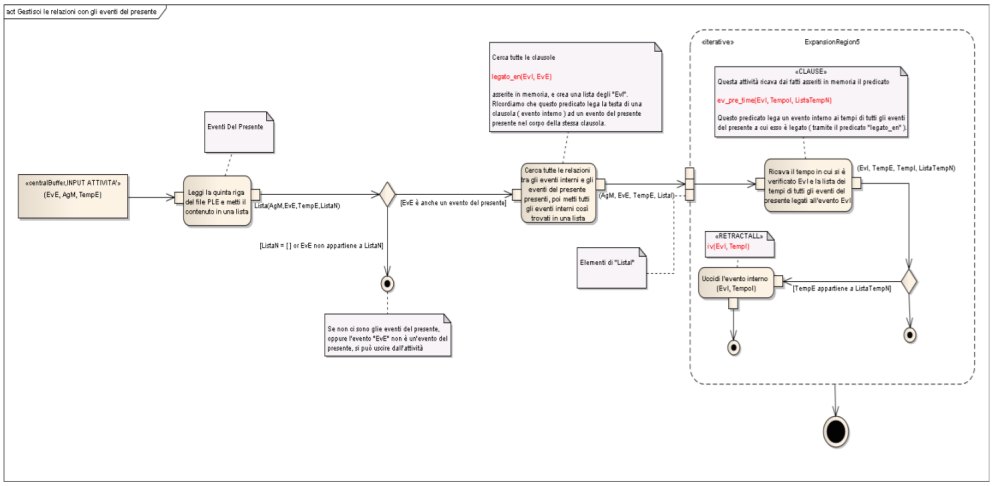
B.2.3.1.1. Updated the list of past events and the list of "rem _facts"



This diagram is a sub-activity of the activity **"Process the events with high priority"** and it is represented in line 758 by the predicate **" processa**

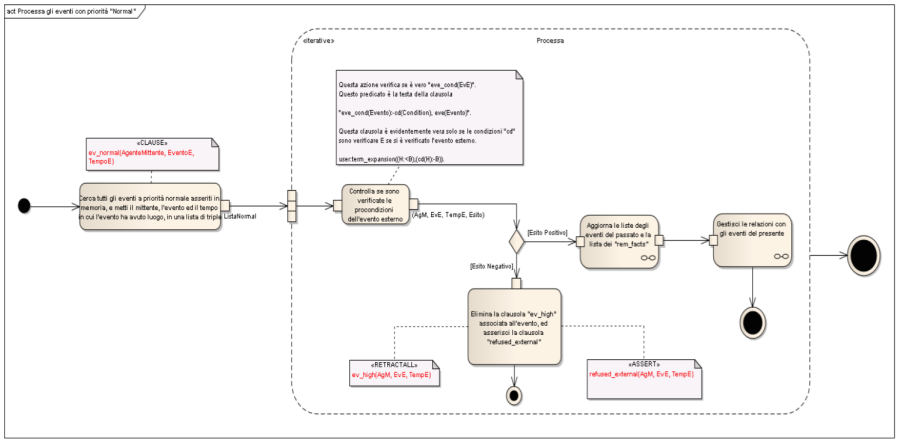
_eve _high/3 ”. The event is eliminated from the queue of the event with priority high to be processed , line 758. The event is added to the list of external event executed, line 758. The predicate “divP” is called in line 1358. The question is: is the event already on the list? If yes the predicate “sposta _in _rem” is called in line 1360: it moves this past event from the list of “rem _facts”. If no the external event is put on the top of the past event’s list, line 1358.

B.2.3.1.2. Manage relationship with the present events



This diagram is a sub-activity of the activity **“Process the events with high priority”** and it is represented in line 758 by the predicate **“ processa _eve _high/3 ”** . The fifth line of ple-file is read and it puts the content in a list. If this list is empty or the event doesn’t belong to the list, the activity ends. If the event is also a present event the predicate “ processa _eve3” is called in line 781. It searched all relations between the internal events and the present events. It puts all internal events in a list. If the “clause(ev _pre _time)” is true in line 783, the predicate “processa _eve4” is called in line 784: the internal event is killed(uccidi _iv/2), line 786.

B.2.3.2. Process the events with normal priority

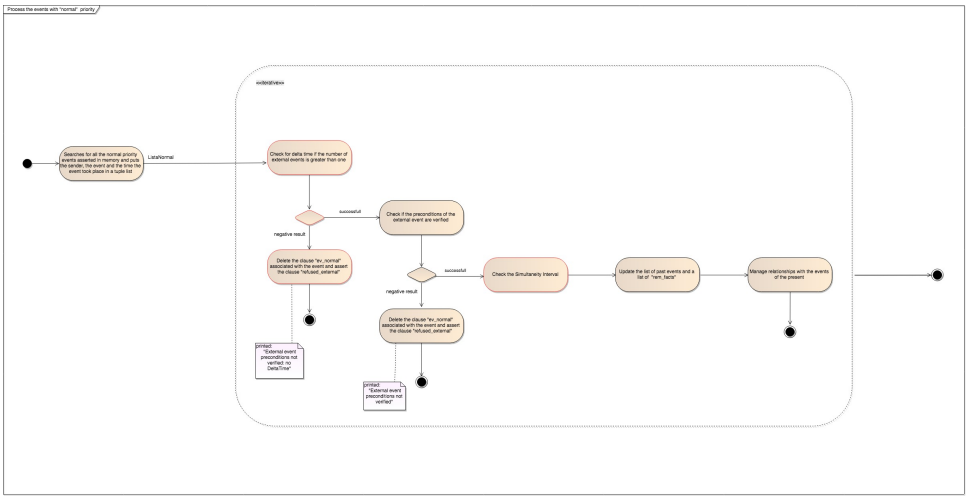


This diagram is a sub-activity of the activity **"Manage External events "** and it is represented in line 763 by the predicate **" processa _eve _normal "** . All events with normal priority asserted in memory are searched and it puts the event, the sender, and the time in which it took place,in a list of tuples, ListNormal. The predicate "processa _eve _normal1" is called in line 763.

In line 766 it checked if the condition of external event is verified: this action verifies if the condition "eve _cond(EVE)" is true(this predicate is in plv file). If the result is negative the predicate "no _proc _eve _normal" is called in line 768: it asserts the clause "refused _external".

If the result is positive the predicate "processa _eve _normal/3" is called in line 775: it updates the list of past events and the list of "rem _facts" and it manages the relationship with the present events.

B.2.3.2.(Updated) Process the events with normal priority



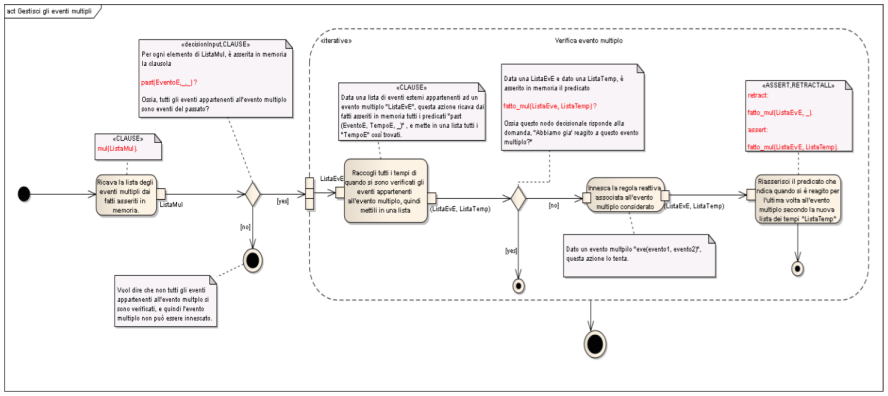
This diagram is a sub-activity of the activity **"Manage External events "** and it is represented in line 763 by the predicate **" processa _eve _normal "** . All events with normal priority asserted in memory are searched and it puts the event, the sender, and the time in which it took place,in a list of tuples, ListNormal. The predicate "processa _eve _normal1" is called in line 764.

In line 765 the predicate *"check _presence"* is called. If the result is false then the predicate "no _proc _eve _normal _no _time" is called in line 771. Else in line 766 it checked if the condition of external event is verified: this action verifies if the condition "eve _cond(EVE)" is true(this predicate is in plv file).

If the result is negative the predicate "no _proc _eve _normal" is called in line 768: it asserts the clause "refused _external".

If the result is positive the predicate "processa _eve _normal/3" is called in line 775: the predicate **"simultaneity_interval"** is called in line 2167 and then it updates the list of past events and the list of "rem _facts" and it manages the relationship with the present events.

B.2.4. Manage multiple events



This diagram is a sub-activity of the activity **"Process External Events"** and it is represented in line 232 by the predicate **"examine _mul"** . The predicate "examine1 _mul" is called in line 233: the list of the multiple events is obtained from the facts asserted in memory(clause(ListaMul)). The predicate "keep _past" is called in line 239: for each element of ListaMul, is the clause " past(EventE, _, _)" asserted in memory? If yes it is collected all time when the events belonging to the multiple event have occurred. The predicates "chiama _cong" in line 245, "controlla _time _ep/2" in line 247 and "cont _tep1/2" in line 251 are called. Given a list ListEVE and a list ListTemp is the predicate "fatto _mul/2" asserted in memory? That is, did you react to this multiple event? If the answer is no, it triggers the corrisponding reactive rule and finally it reassertes the predicate that indicates when the last time you reacted to the multiple event according to the new time list "ListTemp".

B.2.5. Predicate keep _action

The predicate **"keep _action"** is the same predicate defined in section B.1.9.

B.2.6. Predicate svuota _coda _priority

The predicate **"svuota _coda _priority"** is the same predicate defined in section B.1.6.

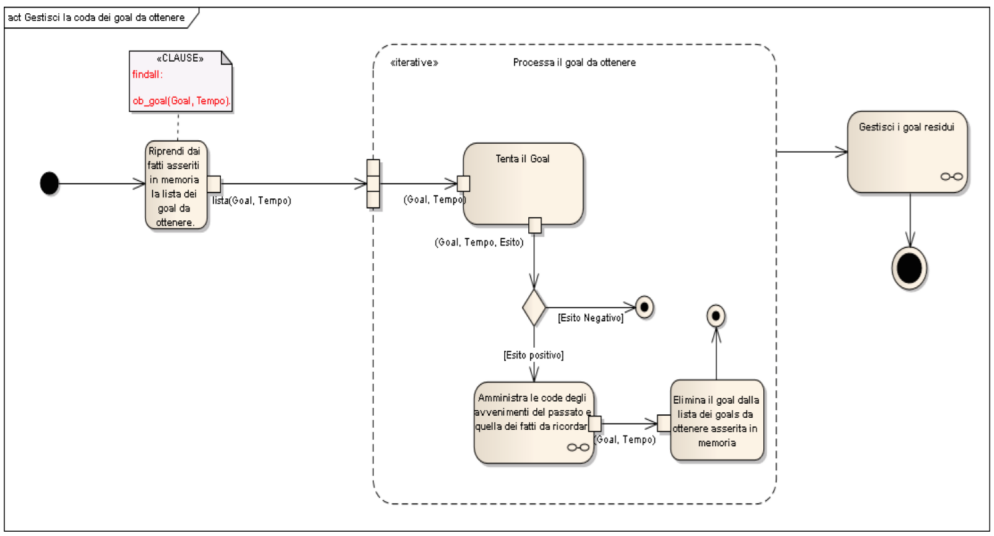
B.2.7. Predicate prendi _action _normal

The predicate **"prendi _action _normal"** is the same predicate defined in section B.1.11.

B.2.8. Predicate controlla _vita

The predicate **"controlla _vita"** is the same predicate defined in section B.1.12.

B.3. Manage queue of the goals to get



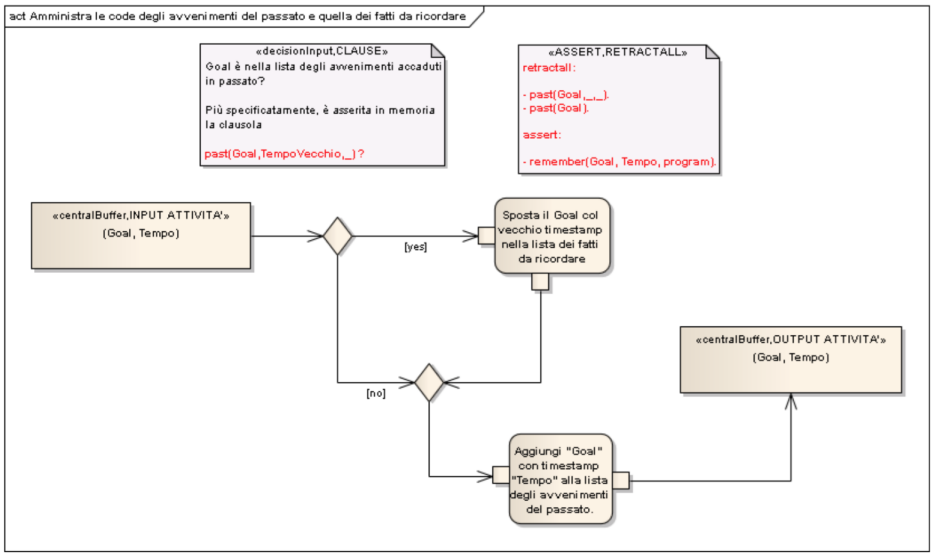
This diagram is represented by the predicate **" side _goal "** in line 1507.

```
side_goal:- obtaining_goals, residue_goal.
```

The predicate " obtaining _goals " is called in line 1259: the list of goal to obtain is recovered from the facts asserted in memory. The predicate " process _ob _goal " is called in line 1260: the goal is tempted. If the predicate "once(El)" has a positive result the predicate "fai _div _evp " is called in line 1261 (B.3.1.) : manage the queue of past events and the facts to

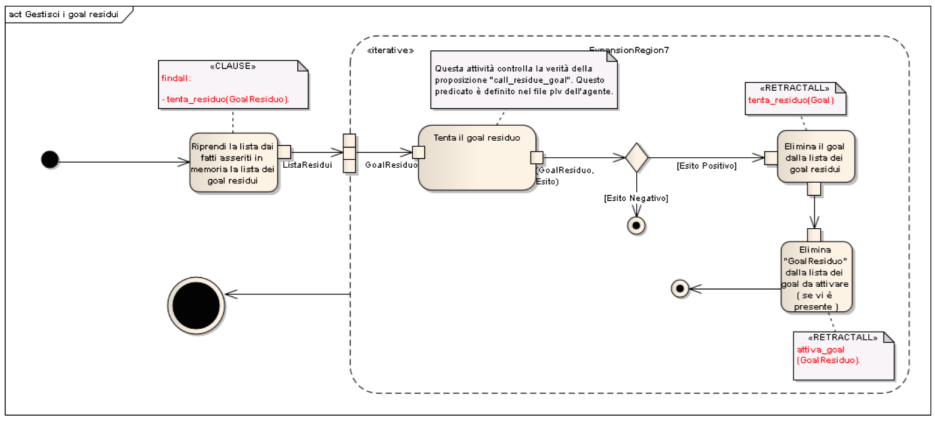
remember. The list of goals asserted in memory is eliminated: “retractall(ob _goal(El,T))”, line 1261. Finally the predicate “residue _goal” is called in line 1533(B.3.2.).

B.3.1. Manage queue of past events and the facts to remember



This diagram is a sub-activity of the activity **“Manage the queue of the goals to get”** and it is represented in line 1261 by the predicate **“ fai _div _evp ”** . The predicate “divP/3 ” is called in line 1358. Is the clause “ clause(past(Goal,Time,_)) ” asserted in memory?
If no the predicate “ asserta(past(Goal, Time, _)) ” is executed in line 1358. If yes the predicate “sposta _in _rem/5 ” is called in line 1360: the goal with timestamp is added to the list of past events.

B.3.2. Manage residual goals



This diagram is a sub-activity of the activity **"Manage the queue of the goals to get"** and it is represented in line 1533 by the predicate **" residue _goal "** . The predicate " residue _goal1 " is called in line 1534: the list of facts asserted in memory is obtained from the list of residual goals. The truth of the preposition " call _residue _goal" is checked in line 1538 (this predicate is defined in pl file of the agent).
If the result is positive then the predicate "resid _goal" is called in line 1540: the goal is eliminated from the list of residual goals. The predicate "canc _goal/1" is called in line 1363: the residual goal is eliminated from the list of goals to activate.