# context\_flter.P

The comments to the code below have been highlighted by ***CAPS*** ***pink bold–italic*** fonts for easier recognition,

***% THIS PREDICATE IS USED TO CHECK IS AN EVENT CAN PASS THROUGH THE FILTER***

can\_pass\_event(***Event***,***Filter\_expression***):-

***% WE FALTTEN THE EVENT FROM A NESTED LIST TO A LIST OF KEY-VALUE PAIRS***

faltten\_to\_key\_value\_pairs(***Event***,[],***Flat\_kv\_pairs***), ***%replace\_keys\_by\_values(Context\_filter\_expression,Flat\_kv\_pairs,Logic\_filled\_expression),***

compute(***Filter\_expression***,***Flat\_kv\_pairs***).

***% THESE PREDICATES ARE USED TO FLATTEN THE NESTED LIST OF KV PAIRS INTO A FLAT LIST OF KV PAIRS***

faltten\_to\_key\_value\_pairs([***H***|[]],***I***,***Flat\_kv\_pairs***):-

faltten\_to\_key\_value\_pairs(***H***,***I***,***Flat\_kv\_pairs***).

faltten\_to\_key\_value\_pairs([***H***|***T***],***I***,***Flat\_kv\_pairs***):-

faltten\_to\_key\_value\_pairs(***H***,***I***,***O1***),

faltten\_to\_key\_value\_pairs(***T***,***O1***,***Flat\_kv\_pairs***).

faltten\_to\_key\_value\_pairs([***Key***,***Value***|[***T***]],***I***,***Flat\_kv\_pairs***):-

not(is\_list(***Key***)),

is\_list(***Value***),

faltten\_to\_key\_value\_pairs(***Value***,***I***,***O1***),

faltten\_to\_key\_value\_pairs(***T***,***O1***,***Flat\_kv\_pairs***).

faltten\_to\_key\_value\_pairs([***Key***,***Value***|[]],***I***,***Flat\_kv\_pairs***):-

not(is\_list(***Key***)),

not(is\_list(***Value***)),

***Kvp*** = [***Key***,***Value***],

append(***I***,[***Kvp***],***Flat\_kv\_pairs***).

faltten\_to\_key\_value\_pairs([***Key***,***Value***|[]],***I***,***Flat\_kv\_pairs***):-

not(is\_list(***Key***)),

is\_list(***Value***),

faltten\_to\_key\_value\_pairs(***Value***,***I***,***Flat\_kv\_pairs***).

***% THIS PREDICATE PRODUCES A BOOLEAN OUTPUT FOR THE INPUT OF FILTER EXPRESSION AND LIST OF KEY-VALUE PAIRS***

compute(***Filter\_expression***,***Flat\_kv\_pairs***):-

compute\_logic\_molecule(***Filter\_expression***,***Flat\_kv\_pairs***).

***% THIS A SHORT EXECUTION PATH FOR IF THE FILTER EXPRESSION IS NOT PROVIDED FOR A COMPONENT***

compute([],***\_***).

***% THIS PREDICATE IS USED TO COMPUTE THE FITER EXPRESSION RECURSIVELY, IT BREAKS THE FILTER EXPRESSION INTO NESTED LISTS OF 3 ELEMENTS EACH***

compute\_logic\_molecule([***Term\_1***,***Operator***,***Term\_2***|[]],***Flat\_kv\_pairs***):-

!,

***% CHECKING IF THE OPERATOR IS SUPPORTED***

member(***Operator***,[>,<,=<,>=,and,or,=\=,=]),

***% CHECKING IF EITHER THE FIRST OR THE SECOND TERM OF THE EXPRESSION IS A LIST IN ITS SELF, THIS WOULD MEAN THAT THE EXPRESSION IS A NESTED EXPRESSION, IN HTIS CASE THE PREDICATE IS CALLED ON THE INNER EXPRESSION RECURSIVELY***

(not(is\_list(***Term\_1***))->get\_value\_for\_key(***Term\_1***,***Flat\_kv\_pairs***,***R1***);(compute\_logic\_molecule(***Term\_1***,***Flat\_kv\_pairs***)->R1 = true;***R1*** = fail)),

(not(is\_list(***Term\_2***))->R2=Term\_2;(compute\_logic\_molecule(***Term\_2***,***Flat\_kv\_pairs***)->R2 = true;***R2*** = fail)),

compute\_logic(***R1***,***Operator***,***R2***).

***% THIS PREDICATE IS USED TO FETCH THE VALUE CORRESPONDING TO A KEY FROM A LIST OF KEY-VALUE PAIRS.***

get\_value\_for\_key(***R1***,[[***Key***,***Value***]|***T***],***V1***):-

(***R1==Key->V1*** = ***Value***;get\_value\_for\_key(***R1***,***T***,***V1***)).

get\_value\_for\_key(***R1***,[[***Key***,***Value***]|[]],***V1***):-

(***R1==Key->V1*** = ***Value***;fail).

***% THIS PREDICATE HANDLES THE EQUALITY OPERATOR***

compute\_logic(***R1***,=,***R2***):-

( ***R1*** == ***R2***

-> true

; fail

).

***% THIS PREDICATE HANDLES THE > OPERATOR***

compute\_logic(***R1***,>,***R2***):-

( ***R1*** > ***R2***

-> true

; fail

).

***% THIS PREDICATE HANDLES THE < OPERATOR***

compute\_logic(***R1***,<,***R2***):-

( ***R1*** < ***R2***

-> true

; fail

).

***% THIS PREDICATE HANDLES THE =< OPERATOR***

compute\_logic(***R1***,=<,***R2***):-

( ***R1*** =< ***R2***

-> true

; fail

).

***% THIS PREDICATE HANDLES THE >= OPERATOR***

compute\_logic(***R1***,>=,***R2***):-

( ***R1*** >= ***R2***

-> true

; fail

).

***% THIS PREDICATE HANDLES THE =/= INEQUALITY OPERATOR***

compute\_logic(***R1***,=\=,***R2***):-

( \+(***R1*** == ***R2***)

-> true

; fail

).

***% THIS PREDICATE HANDLES THE AND OPERATOR***

compute\_logic(***R1***,and,***R2***):-

( ***R1***,***R2***

-> true

; fail

).

***% THIS PREDICATE HANDLES THE OR OPERATOR***

compute\_logic(***R1***,or,***R2***):-

( ***R1***;***R2***

-> true

; fail

).

# creation\_script\_db.P

***% PREDICATE TO CREATE ENVIRONMENT WITH A LIST OF CONTEXT VARIABLES***

create\_environment(***Environment\_name***, ***Context\_var\_list***):-

***%not(environment\_instance(Environment\_name)),***

assert(environment\_instance(***Environment\_name***)),

***% CREATE A COMPONENT INSTANCE FOR EACH ENVIRONMENT SO THAT THE ENVIRONMENT CAN BE NESTED INSIDE ANOTHER ENVIRONMENT***

assert(component\_instance(***Environment\_name***)),

***% ASSOSIATE CONTEXT VARIABLES TO THE NEW ENVIRONMENT***

add\_context\_variable\_list\_environment(***Environment\_name***,***Context\_var\_list***).

***% PREDICATE TO CREATE A COMPONENTWITH A LIST OF CONTEXT VARIABLES***

create\_component(***Component\_name***, ***Context\_variable\_list***):-

***%not(component\_instance(Component\_name)),***

assert(component\_instance(***Component\_name***)),

***% ASSOSIATE THE CONTEXT VARIABLES TO THE NEW CONTEXT***

add\_component\_context\_variable\_list(***Component\_name***,***Context\_variable\_list***).

***% PREDICATE TO DECLARE A NEW EVENT SCHEMA, THE EVENT SCHEMA IS REPRESENTED IN A NESTED LIST FORMAT WHERE DATA IS REPRESENTED AS KV PAIR***

create\_event\_schema(***Event\_id***,***Event\_schema\_list***):-

assert(event\_schema(***Event\_id***,***Event\_schema\_list***)).

***% PREDICATE TO CREATE AN ADVERTISEMENT FOR A COMPONENT, AN ADVERTISEMENT IS ASSOSIATED WITH A COMPONENT, AN EVENT TYPE, AN ENVIRONMENT AND A CONTEXT FILTER. AN ENVIRONMENT IS NEEDED BECAUSE ADVERTISEMENTS ARE NOT SHARED BETWEEN ENVIRONMENTS. THEREFORE IF A COMPONENT WANTS TO DISTRIBUTE AN EVENT TO MULTIPLE PARENTS IT WILL NEED MULTIPLE ADVERTISEMENTS FOR EACH PARENT ENVIRONMENT. THE ADVERTISEMENT ALSO STORES A CONTEXT FILTER EXPRESSION THAT AN EVENT NEEDS TO PASS BEFORE REACHING THE IMMEDIATE PARENT ENVIRONMENT.***

create\_advertisement(***Component\_name***,***Environment\_name***,***Event\_id***,***Context\_filter\_expression***):-

***% CHECKING IF THE COMPONENT ALREADY EXISTS***

component\_instance(***Component\_name***),

***% CHECKING IF THE ENVIRONMENT EXISTS***

environment\_instance(***Environment\_name***),

***% CHECKING IF THE COMPONENT IS CONTAINED IN THE ENVIRONMENT***

environment\_component(***Environment\_name***,***Component\_name***),

***% CHECKING IF THE EVENT TYPE EXISTS***

event\_schema(***Event\_id***,***\_***),

assert(event\_advertisement\_entry(***Component\_name***,***Environment\_name***,***Event\_id***,***Context\_filter\_expression***)).

***% PREDICATE TO CREATE AN ADVERTISEMENT FOR AN ENVIRONMENT. . THE ADVERTISEMENT ALSO STORES A CONTEXT FILTER EXPRESSION THAT AN EVENT NEEDS TO PASS BEFORE REACHING THE IMMEDIATE PARENT ENVIRONMENT.***

create\_advertisement(***Environment\_name***,***Event\_id***,***Context\_filter\_expression***):-

***% CHECKING IF THE ENVIRONMENT EXISTS***

environment\_instance(***Environment\_name***),

***% CHECKING IF THE EVENT TYPE EXISTS***

event\_schema(***Event\_id***,***\_***),

assert(event\_advertisement\_entry(***Environment\_name***,***Environment\_name***,***Event\_id***,***Context\_filter\_expression***)).

***% PREDICATE TO CREATE A SUBSCRIPTION FOR A COMPONENT, A SUSCRIPTION IS ASSOSIATED WITH A COMPONENT, AN ENVIRONMENT, AN EVENT TYPE, A CONTEXT FILTER AND A CONTENT FILTER. AN ENVIRONMENT IS NEEDED AS SUBSCRIPTIONS ARE NOT SHARD AMONG ENVIRONMENTS. THEREFORE IF A COMPONENT WANTS TO SUBSCRIBE TO AN EVENT FROM MULTIPLE PARENTS IT WILL NEED MULTIPLE SUBSCRIPTIONS FOR EACH PARENT ENVIRONMENT. A CONTENT FILTER IS EVALUATED BASED ON THE CONTENT FILTER EXPRESSION AND THE CONTENTS OF THE EVENT. A CONTEXT FILTER IS EVALUATED BASED ON THE CONTEXT FILTER EXPRESSION AND THE VALUES OF THE CONTEXT VARIABLES HOSTED BY THE PARENT ENVIRONMENTS.***

create\_subscription(***Component\_name***,***Environment\_name***,***Event\_id***,***Content\_filter\_expression***,***Context\_filter\_expression***):-

***% CHECKING IF THE COMPONENT EXISTS***

component\_instance(***Component\_name***),

***% CHECKING IF THE EVENT TYPE EXISTS***

event\_schema(***Event\_id***,***\_***),

assert(event\_subscription\_entry(***Component\_name***,***Environment\_name***,***Event\_id***,***Content\_filter\_expression***,***Context\_filter\_expression***)).

add\_component\_context\_variable\_list(***Component\_name***,[]).

***% THESE PREDICATES ARE USED TO ITERATE THE LIST OF CONTEXT VARIABLES OF COMPONENTS***

add\_component\_context\_variable\_list(***Component\_name***,[***H***|***T***]):-

add\_component\_context\_variable(***Component\_name***,***H***),

add\_component\_context\_variable\_list(***Component\_name***,***T***).

***% THIS PREDICATE IS USED TO STORE EACH CONTEXT VARIABLE***

add\_component\_context\_variable(***Component\_name***,[***Key***,***Value***|[]]):-

assert(context\_variable(***Component\_name***,[***Key***,***Value***])).

***% THIS PREDICATE IS USED TO ADD A COMPONENT TO AN ENVIRONMENT. THIS PREDICATE CAN ALSO BE USED TO NEST ENVIRONMENTS AS EACH ENVIRONMENT IS ALSO ASSERTTED AS A COMPONENT.***

add\_component\_to\_environment(***Environment\_name***, ***Component\_name***):-

***% CHECK IF ENVIRONMENT EXISTS***

environment\_instance(***Environment\_name***),

***% CHECK IF THE COMPONENT EXISTS***

component\_instance(***Component\_name***),

assert(environment\_component(***Environment\_name***,***Component\_name***)).

***% THIS PREDICATE IS USED TO ITTERATE THE LIST OF ENVIRONMENT CONTEXT VARIABLES***

add\_context\_variable\_list\_environment(***Environment\_name***,[***H***|***T***]):-

assert(context\_variable(***Environment\_name***,***H***)),

add\_context\_variable\_list\_environment(***Environment\_name***,***T***).

add\_context\_variable\_list\_environment(***Environment\_name***,[]).

***% THIS PREDICATE IS USED TO ADD A LIST OF COMPONENTS TO ENVIRONMENT***

add\_component\_list\_environment(***Environment\_name***,[***H***|***T***]):-

add\_component\_to\_environment(***Environment\_name***,***H***),

add\_component\_list\_environment(***Environment\_name***,***T***).

add\_component\_list\_environment(***Environment\_name***,[]).

***% THIS PREDIACTE IS USED FOR GETING ALL THE EVENT TYPES FOR AN ENVIRONMENTS SUBCOMPONENTS FOR THAT HAVE ADVERTISEMENTS***

get\_event\_id\_list\_advertisement\_environment(***Environment\_name***,***Event\_id\_list***):-

findall(***X***,environment\_event\_advertisement(***Environment\_name***,***X***),***Event\_id\_list***).

***% THIS PREDICATE IS TRUE IS THE ENVIRONMENT OR ONE OF ITS COMPONENTS HAS AN ADVERTISEMENTS FOR THE EVENT TYPE***

environment\_event\_advertisement(***Environment\_name***,***Event\_id***):-

***% CHECK IF THE ENVIRONMENT EXISTS***

environment\_instance(***Environment\_name***),

***% CHECK IF THE EVENT TYPE EXISTS***

event\_schema(***Event\_id***,***\_***),

***% CHECK IF THE COMPONENT IS CONTAINED INSIDE THE ENVIRONMENT***

environment\_component(***Environment\_name***,***Component\_name***),

event\_advertisement\_entry(***Component\_name***,***Environment\_name***,***Event\_id***,***Context\_filter\_expression***).

***% THIS PREDICATE IS TRUE IS THE ENVIRONMENT OR ONE OF ITS COMPONENTS HAS AN SUBSCRIPTIONS FOR THE EVENT TYPE***

environment\_event\_subscription(***Environment\_name***,***Event\_id***):-

***%CHECK IF THE ENVIRONMENT EXISTS***

environment\_instance(***Environment\_name***),

***% CHECK IF THE EVENT TYPE EXISTS***

event\_schema(***Event\_id***,***\_***),

***% CHECK IF THE COMPONENT IS CONTAINED INSIDE THE ENVIRONMENT***

environment\_component(***Environment\_name***,***Component\_name***),

event\_subscription\_entry(***Component\_name***,***Environment\_name***,***Event\_id***,***Content\_filter\_expression***,***Context\_filter\_expression***).

***% PREDICATE TO CHECK IF ANY OF THE N DEGREE CHILD OF THE COMPONENT HAS A SUBSCRIPTION FOR THIS KIND OF EVENT***

component\_event\_subscription(***Component\_name***,***Event\_id***):-

event\_subscription\_entry(***Component\_name***,***Environment\_name***,***Event\_id***,***Content\_filter\_expression***,***Context\_filter\_expression***).

component\_event\_subscription(***Component\_name***,***Event\_id***):-

***% GET A CHILD CONTAINED INSIDE THE COMPONENT***

environment\_component(***Component\_name***,***Sub\_component\_name***),

component\_event\_subscription(***Sub\_component\_name***,***Event\_id***).

***% PREDICATE TO CHECK IF ANY OF THE N DEGREE CHILD OF THE COMPONENT HAS AN ADVERTISEMENT FOR THIS KIND OF EVENT***

component\_event\_advertisement(***Component\_name***,***Environment\_name***,***Event\_id***):-

event\_advertisement\_entry(***Component\_name***,***Environment\_name***,***Event\_id***,***Context\_filter\_expression***).

component\_event\_advertisement(***Component\_name***,***Environment\_name***,***Event\_id***):-

environment\_component(***Component\_name***,***Sub\_component\_name***),

component\_event\_advertisement(***Sub\_component\_name***,***Component\_name***,***Event\_id***).

***% PREDICATE TO GET ALL THE FIRST DEGREE COMPONENTS OF AN ENVIRONMENT***

get\_component\_list\_environment(***Environment\_name***,***Component\_name\_list***):-

findall(***X***,environment\_component(***Environment\_name***,***X***),***Component\_name\_list***).

***% THIS PREDICATE IS USED TO SIMULATE AN EVENT ORIGINATION FROM A COMPONENT***

simulate\_event\_component(***Component\_name***,***Environment\_name***,***Event\_id***,***Event\_data\_list***):-

component\_instance(***Component\_name***),

***% TYPE FILTER START, THIS IS A CHECK IF THE COMPONENT HAS AN ADVERTISEMENT FOR THIS TYPE OF EVENT***

event\_advertisement\_entry(***Component\_name***,***Environment\_name***,***Event\_id***,***Context\_filter\_expression***),

***% TYPE FILTER END***

***% CONTEX FILTER START, SINCE THE OUTGOING EVENT S ONLY NEED TO PASS THE CONTEXT FILTER AND NOT THE CONTENT FILTER***

findall(***X***,get\_context\_variables(***Environment\_name***,***\_***,***X***),***Environment\_context\_variable\_list***),

***% write('Context filter expression - '),write(Context\_filter\_expression),nl,***

***% write('Context filter data'),write(Environment\_context\_variable\_list),nl,***

***% THE BELOW IS USED TO TEST IF THE EVENT PASSES THE CONTEXT FILTER OF THE ADVERTISEMENT***

compute(***Context\_filter\_expression***,***Environment\_context\_variable\_list***),

***% write('Context filter passed'),nl,***

***% CONTEX FILTER END***

***% VALIDATE THE EVENT TO ONE OF THE KNOWN EVENT SCHEMAS***

[***Event\_type***|***Et***] = ***Event\_data\_list***,

event\_schema(***Event\_type***,***Event\_schema***),

***% VALIDATE IT FURTHER BY A DATA TYPE CHECK OS OF THE VALUES OF THE KV PAIRS***

!,unify\_event(***Event\_data\_list***,***Event\_schema***),

***% THE BELOW IS USED TO THE ACTIONS THAT ARE TAKEN BY THE COMPONENT ON THE ARRIVAL OF THIS TYPE OF EVENT***

!,trigger\_component\_event\_response(***Component\_name***,***Event\_type***,***Event\_data\_list***),

***% WE NOW MOVE TO THE ENVIRONMRNT THAT CONTAINS THE COMPONENT***

get\_reachable\_environments(***Event\_type***,***Event\_data\_list***,[***Environment\_name***],***Component\_name***,[***Component\_name***],***Reachable\_environment\_list***).

***% nl,write('Event '),write(Event\_id),write(' Visited List -->'),write(Reachable\_environment\_list),nl.***

***% THIS PREDICATE IS USED TO SIMULATE AN EVENT ORIGINATION FROM AN ENVIRONMENT***

simulate\_event\_environment(***Environment\_name***,***Event\_id***,***Event\_data\_list***):-

environment\_instance(***Environment\_name***),

***% TYPE FILTER START***

event\_advertisement\_entry(***Environment\_name***,***Environment\_name***,***Event\_id***,***Context\_filter\_expression***),

***% TYPE FILTER END***

***% VALIDATE THE EVENT TO ONE OF THE KNOWN EVENT SCHEMAS***

[***Event\_type***|***Et***] = ***Event\_data\_list***,

event\_schema(***Event\_id***,***Event\_schema***),

***% VALIDATE IT FURTHER BY A DATA TYPE CHECK OS OF THE VALUES OF THE KV PAIRS***

!,unify\_event(***Event\_data\_list***,***Event\_schema***),

***% WE NOW MOVE TO THE ENVIRONMRNT THAT CONTAINS THE ENVIRONMENT***

***% write('Event\_id - '),write(Event\_id),***

***% write('Event\_data\_list - '),write(Event\_data\_list),***

***% write('Environment\_name - '),write(Environment\_name),***

get\_reachable\_environments(***Event\_id***,***Event\_data\_list***,[***Environment\_name***],***Environment\_name***,[],***Reachable\_environment\_list***).

nl,write('Event '),write(***Event\_id***),write(' Visited List -->'),write(***Reachable\_environment\_list***),nl.

process\_event\_environment(***Environment\_name***,***Event\_id***,***Event\_data\_list***,***Calling\_component\_name***):-

event\_subscription\_entry(***Environment\_name***,***Environment\_name***,***Event\_id***,***Content\_filter\_expression***,***Context\_filter\_expression***),

***% write('processing event '),write(Event\_id),write(' for environment '),write(Environment\_name),nl,***

***% write('calling component '),write(Calling\_component\_name),nl,***

***% CONTEX VARIABLE START***

(environment\_component(***Environment\_name***,***Calling\_component\_name***)

->

findall(***X***,get\_context\_variables(***Environment\_name***,***\_***,***X***),***Environment\_context\_variable\_list***)

;

findall(***X***,get\_context\_variables(***Calling\_component\_name***,***\_***,***X***),***Environment\_context\_variable\_list***)

),

***% write('Context filter expression - '),write(Context\_filter\_expression),nl,***

***% write('Context filter data'),write(Environment\_context\_variable\_list),nl,***

compute(***Context\_filter\_expression***,***Environment\_context\_variable\_list***),

***% write('Context filter passed'),nl,***

***% CONTEX VARIABLE END***

***% CONTENT FILTER START***

can\_pass\_event(***Event\_data\_list***,***Content\_filter\_expression***),

***% write('\*\*Passed\*\* Component Content Filter - '),write(Environment\_name),nl,***

***% CONTENT FILTER END***

trigger\_component\_event\_response(***Environment\_name***,***Event\_id***,***Event\_data\_list***).

process\_event\_environment(***Environment\_name***,***Event\_id***,***Event\_data\_list***,***Calling\_component\_name***).

***% ONE OF THE DEFINITIONS OF THE A RECURSIVE PREDICATE FOR ITERATING THROUGH ALL THE COMPONENTS IN AN ENVIRONMENT***

get\_reachable\_environments(***Event\_type***,***Event\_data\_list***,[],***Calling\_component\_name***,***I***,***I***).

***% ONE OF THE DEFINITIONS OF THE A RECURSIVE PREDICATE FOR ITERATING THROUGH ALL THE COMPONENTS IN AN ENVIRONMENT***

get\_reachable\_environments(***Event\_type***,***Event\_data\_list***,[***Rh***|***Rt***],***Calling\_component\_name***,***I***,***O***):-

get\_reachable\_environments(***Event\_type***,***Event\_data\_list***,***Rh***,***Calling\_component\_name***,***I***,***O2***),

get\_reachable\_environments(***Event\_type***,***Event\_data\_list***,***Rt***,***Calling\_component\_name***,***O2***,***O***).

***% THIS PREDICATE IS USED TO COMPUTE THE EFFECT OF AN EVENT ON REACHING AN ENVIRONMENT***

get\_reachable\_environments(***Event\_type***,***Event\_data\_list***,***Component\_name***,***Calling\_component\_name***,***I***,***O***):-

environment\_instance(***Component\_name***),

***% CHECKING THAT THE ENVIRONMENT HAS NOT BEEN VISITIED BEFORE***

not(member(***Component\_name***,***I***)),

***% write('Visiting Environment - '),write(Component\_name),nl,***

***% write('\_-> Already Visited - '),write(I),nl,***

***% ADD THE NAME OF THIS ENVIRONMENT IN THE LIST TO PREVENT VISITING IT AGAIN***

append(***I***,[***Component\_name***],***I2***),

process\_event\_environment(***Component\_name***,***Event\_type***,***Event\_data\_list***,***Calling\_component\_name***),

***% FIND ALL THE ENVIRONMENTS THAT CONTAIN THIS ENVIRONMENT***

***% THIS CALL IS SPREADING THE ENENT UPWARDS TOWARDS THE PARENT ENVIRONMENTS***

findall(***X***,environment\_component(***X***,***Component\_name***),***Environment\_name\_list***),

***% write('|-> Parent Environments - '),write(Environment\_name\_list),nl,***

***% THE THE METHOD RECURSIVELY TO VISIT PARENT ENVIRONMENT***

get\_reachable\_environments(***Event\_type***,***Event\_data\_list***,***Environment\_name\_list***,***Component\_name***,***I2***,***I3***),

***% write('@@'),write(I3),nl,***

***% WE ARE NOW LOOKING FOA ALL THE COMPONENTS AND ENVIRONMENTS CONTAINED IN THIS ENVIRONMENT***

findall(***X***,environment\_component\_event\_subscription(***Component\_name***,***X***,***Event\_type***),***Sub\_component\_name\_list***),

***% write('|-> Sub of '),write(Component\_name),write(' - '),write(Sub\_component\_name\_list),nl,***

***% NOW WE ARE GOING TO SPRED THE EVENT DOWNWARDS TOWARDS THIS ENVIRONMENTS CHILDRREN COMPONENTS AND ENVIRONEMENTS***

get\_reachable\_environments(***Event\_type***,***Event\_data\_list***,***Sub\_component\_name\_list***,***Component\_name***,***I3***,***O***).

***% THIS PREDICATE IS USED TO COMPUTE THE EFFECT OF AN EVENT ON REACHING A COMPONENT***

get\_reachable\_environments(***Event\_type***,***Event\_data\_list***,***Component\_name***,***Calling\_component\_name***,***I***,***O***):-

not(environment\_instance(***Component\_name***)),

component\_instance(***Component\_name***),

***% CHECKING THAT THE ENVIRONMENT HAS NOT BEEN VISITIED BEFORE***

not(member(***Component\_name***,***I***)),

append(***I***,[***Component\_name***],***O***),

***% TYPE FILTER START, CHECKING IF THE COMPONENT HAS SUBSCRIBED TO THIS TYPE OF EVENT***

event\_subscription\_entry(***Component\_name***,***Environment\_name***,***Event\_type***,***Content\_filter\_expression***,***Context\_filter\_expression***),

***% TYPE FILTER END***

***% write('##Matching## Component Found - '),write(Component\_name),nl,***

***% CONTEX FILTER START***

findall(***X***,get\_context\_variables(***Calling\_component\_name***,***\_***,***X***),***Environment\_context\_variable\_list***),

***% write('Context filter expression - '),write(Context\_filter\_expression),nl,***

***% write('Context filter data'),write(Environment\_context\_variable\_list),nl,***

***% CHECKING IF THE EVENT PASSES THE CONTEXT FILTER***

compute(***Context\_filter\_expression***,***Environment\_context\_variable\_list***),

***% write('Context filter passed'),nl,***

***% CONTEX FILTER END***

***% CONTENT FILTER START***

can\_pass\_event(***Event\_data\_list***,***Content\_filter\_expression***),

***% write('\*\*Passed\*\* Component Content Filter - '),write(Component\_name),nl,***

***% CONTENT FILTER END***

***% write('Visiting Component - '),write(Component\_name),nl,***

***% NOW WE EXECUTE THE PREDICATE THAT SIGNALS THAT THE EVENT HAS BEEN ACCEPTED BY THE COMPONENT, IT IS ALSO WHERE THE COMPONENT SERVICES THE EVENT***

trigger\_component\_event\_response(***Component\_name***,***Event\_type***,***Event\_data\_list***).

***% ONE OF THE DEFINITIONS, IT IS USED TO PREVENT THE BREAK IN THE RECURSION CHAIN IF THE COMPONENT DOES NOT SUBSCRIBE TO A EVENT TYPE***

get\_reachable\_environments(***Event\_type***,***Event\_data\_list***,***Component\_name***,***Calling\_component\_name***,***I***,***O***):-

not(environment\_instance(***Component\_name***)),

component\_instance(***Component\_name***),

not(member(***Component\_name***,***I***)),

append(***I***,[***Component\_name***],***O***),

***% write('Visiting Component - '),write(Component\_name),nl,***

not(event\_subscription\_entry(***Component\_name***,***Environment\_name***,***Event\_type***,***Content\_filter\_expression***,***Context\_filter\_expression***)).

***% THIS PREDICATE IS USED AS AN ALL PASS DEFINITION FOR THE RECURSIVE PREDICATE***

get\_reachable\_environments(***Event\_type***,***Event\_data\_list***,***Component\_name***,***Calling\_component\_name***,***I***,***I***).

get\_context\_variables(***Environment\_name***,***Component\_name***,***Environment\_context\_variable***):-

***Component\_name*** = ***Environment\_name***,

context\_variable(***Environment\_name***,***Environment\_context\_variable***).

environment\_component\_event\_subscription(***Environment\_name***,***Component\_name***,***Event\_id***):-

environment\_component(***Environment\_name***,***Component\_name***),

event\_subscription\_entry(***Component\_name***,***Environment\_name***,***Event\_id***,***\_***,***\_***).

environment\_component\_event\_subscription(***Environment\_name***,***Component\_name***,***Event\_id***):-

environment\_component(***Environment\_name***,***Component\_name***),

environment\_instance(***Component\_name***).

***% THIS PREDICATE IS USED TO FETCH THE CONTEXT VARIABLES OF AN ENVIRONMENT, AND THE COMPONENTS CONTAINED WITHIN IT***

get\_context\_variables(***Environment\_name***,***Component\_name***,***Environment\_context\_variable***):-

environment\_instance(***Environment\_name***),

ancestor(***Environment\_name***,***Component\_name***),

context\_variable(***Component\_name***,***Environment\_context\_variable***).

***% THIS PREDICATE IS USED TO GET A CONTEXT VARIABLE CREATED FOR AN ENVIRONMENT***

get\_context\_variables(***Environment\_name***,***Component\_name***,***Environment\_context\_variable***):-

context\_variable(***Environment\_name***,***Environment\_context\_variable***).

get\_context\_variables(***Environment\_name***,***Component\_name***,***Environment\_context\_variable***):-

environment\_instance(***Environment\_name***),

ancestor(***Environment\_name***,***Sub\_environment\_name***),

context\_variable(***Sub\_environment\_name***,***Environment\_context\_variable***).

***% USED TO FIND THE ACESTORY OF A COMPONENT OR ENVIRONMENT***

ancestor(***Environment\_name***,***Component\_name***):-

environment\_component(***Environment\_name***,***Component\_name***).

***% USED TO FIND THE ACESTORY OF A COMPONENT OR ENVIRONMENT***

ancestor(***Environment\_name***,***Component\_name***):-

environment\_component(***Sub\_environment\_name***,***Component\_name***),

ancestor(***Environment\_name***,***Sub\_environment\_name***).

***% THE PREDICATE IS USED TO CAHNGE THE VALUE OF A CONTEXT VARIABLE***

change\_context\_variable\_value(***Context\_variable\_name***,***Context\_variable\_new\_value***):-

context\_variable(***Component\_name***,[***Context\_variable\_name***,***Old\_value***]),

retract(context\_variable(***Component\_name***,[***Context\_variable\_name***,***\_***])),

***% write('Setting new value '),write(Context\_variable\_new\_value),write(' for context variable '),write(Context\_variable\_name),nl,***

assert(context\_variable(***Component\_name***,[***Context\_variable\_name***,***Context\_variable\_new\_value***])).

***% USED TO COMPUTE THE DISTANCE BETWEEN THE 2 PLANAR POINTS***

distance(***X1***,***Y1***,***X2***,***Y2***,***D***):-

***D*** is sqrt((***X2-X1***) + (***Y2-Y1***)).

# nested\_list\_approach.P

***% THIS PREDICATE IS USED TO CREATE NESTED EVENT SCHEMAS, IT GENERATES SCHEMAS FOR EVENTS THAT CONTAIN OTHER EVENTS. THIS IS NO LONGER DIRECTLY USED NOW BUT IT IS A USEFULT UTILITY WHEN THINKING ABOUT LARGE EVENT SCHEMAS.***

lower\_event([***Key***,***Value***|[]],***I***,***O***):-

***% CHECK IF THE KEY IS AN DEFINED EVENT TYPE***

not(event\_schema([***Key***|***SchemaTail***])),

***% CHECK IF THE VALUE IS AN DEFINED EVENT TYPE***

not(event\_schema([***Value***|***SchemaTail***])),

not(is\_list(***Key***)),

not(is\_list(***Value***)),

***%nl,write(Key),write('-'),write(Value),nl,***

***Kvp*** = [***Key***,***Value***],

append(***I***,[***Kvp***],***O***).

lower\_event([***Head***|[]],***I***,***O***):-

lower\_event(***Head***,***I***,***O***).

lower\_event([***Head***|***Tail***],***I***,***O***):-

lower\_event(***Head***,***I***,***Os1***),

lower\_event(***Tail***,***Os1***,***O***).

lower\_event(***Type***,***I***,***O***):-

***% FETCH THE SCHEMA DEFINITTION OF THE SCHEMA TYPE***

event\_schema([***Type***|***SchemaTail***]),

***%write(Type),write('->'),***

append(***I***,[***Type***],***Os1***),

***%write([SchemaTail]),nl,***

***% RESOLVE THE COMPONENTS OF THE SCHEMA DEFINITION FOR OTHER NESTED TYPES***

lower\_event(***SchemaTail***,[],***Os2***),

append(***Os1***,[***Os2***],***O***).

lower\_event(***Type***,***I***,***O***):-

not(event\_schema([***Type***|***SchemaTail***])),

***%write(Type),write('\*-'),nl,***

append(***I***,[***Type***],***O***).

step\_forward([***Head***|[]]).

step\_forward([***Head***|***Tail***]):-

***%write(Head),***

lower\_event(***Head***),

step\_forward(***Tail***).

***% THIS IS THE ENTRY PREDICATE FOR THE EVENT SCHEMA GENERATOR***

top\_event(***Type***,***O***):-

***% write('Resolving Schema'),nl,***

lower\_event(***Type***,[],***O***).

***% THIS PREDICATE IS USED TO CHECK IF THE PROVIDED EVENT AND THE SCHEMA UNIFY***

unify\_event(***Event***,***Schema***):-

***% write('Start'),nl,***

[***Eh***|***Et***] = ***Event***,

***% For computing the schema internally, this is no longer needed but can be a helpful utility.***

***% top\_event(Eh,Schema),***

***% write('Schema Decided'),nl,***

***% EASIEST CHECK FIRST, CHECK IF THE LENGHTS OF THE 2 ARE SAME, IF NOT WE DO NOT NEED TO CHECK FURTHER***

!,unify\_length(***Event***,***Schema***),

***% write('length compared'),nl,***

***% IF THE LENGHS MATCH, WE MOVE ON THE ELEMENTS OF THE EVENT AND COMPARE THE KEYS AND THE TYPE OF VALUES***

!,compare\_elements(***Event***,***Schema***).

***% write('Schema compared'),nl.***

***% THIS PREDICATE IS USED TO ITERRATE THE EVENT AND THE SCHEMA TOGATHER***

compare\_elements([***Eh***|[]],[***Sh***|[]]):-

***%write('End'),nl,***

!,compare\_elements(***Eh***,***Sh***).

compare\_elements([***Eh***|***Et***],[***Sh***|***St***]):-

***%write('middle'),nl,***

!,compare\_elements(***Eh***,***Sh***),

compare\_elements(***Et***,***St***).

***% CHECK IF THE EVENT ELEMENT IS AN INTEGER***

compare\_elements(***Eventelement***,integer):-

***%write('comparing-integer => '),write(Eventelement),write('='),write(integer),nl,***

!,integer(***Eventelement***).

***% CHECK IF THE EVENT ELEMENT IS A FLOAT***

compare\_elements(***Eventelement***,float):-

***%write('comparing-float => '),write(Eventelement),write('='),write(float),nl,***

!,float(***Eventelement***).

***% CHECK IF THE EVENT ELEMENT IS A STRING***

compare\_elements(***Eventelement***,string):-

***%write('comparing-string => '),write(Eventelement),write('='),write(string),nl,***

!,atom(***Eventelement***).

***% THIS DEFINITION OF THE PREDICATE ENSURES THAT ALL THE KEYS ARE THE SAME BETWEEN THE EVENT AND THE SCHEMA***

compare\_elements(***Sameelement***,***Sameelement***).

***% THIS PREDICATE IS SUCESSFULL IF THE LENGHT OD THE 2 NESTED LISTS ARE THE SAME***

unify\_length(***Event***,***Schema***):-

***% FLATTEN THE SCHEMA NESTED LIST TO A FLAT LIST***

flatten2(***Schema***,***FlatSchema***),

***% FLATTEN THE EVENT NESTED LIST TO A FLAT LIST***

flatten2(***Event***,***FlatEvent***),

***% FIND THE LENGHT OF THE SCHEMA FLAT LIST***

length(***FlatSchema***,***Lengthschema***),

***% FIND THE LENGHT OF THE EVENT FLAT LIST***

length(***FlatEvent***,***Lengthevent***),

***% UNIFY THE 2 LENGHTS***

***Lengthschema=Lengthevent.***

***% UTILITY PREDICATE TO FLATTEN A NESTED LIST***

flatten2([], []) :- !.

flatten2([***L***|***Ls***], ***FlatL***) :-

!,

flatten2(***L***, ***NewL***),

flatten2(***Ls***, ***NewLs***),

append(***NewL***, ***NewLs***, ***FlatL***).

flatten2(***L***, [***L***]).