DEVICE Compilation Algorithms

In the following we assume that the parser has transformed the DEVICE rule language expression into the following structure:

```
rule ::= [cond,action]
cond ::= {op_I}, I>0
op_I ::= pop_I | ¬pop_I
pop_I ::= [oid_var_I,class_I,{ap_I,J}], J>=0
ap_I,J ::= attr_I,J opert_I,J operd_I,J | attr_I,J ':' var_I,J rel_op_I,J operd_I,J
opert ::= ':' | rel_op
rel_op ::= '='|'>'|'>='|'=<'|'<'|'\='
operd ::= const | var
attr ::= A valid attribute of class
const ::= A Prolog constant,     var ::= A Prolog variable,     action ::= a compound Prolog goal
```

Furthermore, the various lists used in the algorithms have the following structure:

```
var_list ::= {var(var,event_oid_op_list,action_occ)}
event_oid_op_list ::= {ev(event_oid,event_oid,opert)}
top_event_list ::= {event_oid}
event_oid ::= The OID of an event object
action_occ ::= integer
```

- The following function compiles a production rule into an ECA rule and returns its OID.

```
function compile_rule(rule)
VAR_LIST := {}
OID_VAR_LIST := {}
TOP_EVENT_LIST := {}
NEGATIVE_LIST := {}
/* First Pass - Create 1-input events */
∀ op_I ∈ cond
  OID_VAR_LIST := OID_VAR_LIST ∪ {var(oid_var_I,{},0)}
  ∀ ap_I,J ∈ pop_I
    if ap_I,J=(attr_I,J:var_I,J) then
      EVENT_OID := create_p_event("put_",attr_I,J,class_I,oid_var_I,var_I,J)
      A_EVENT_OID := create_p_event("delete_",attr_I,J,class_I,oid_var_I,var_I,J)
      VAR_LIST := VAR_LIST ∪ {var(var_I,J,{ev(EVENT_OID,A_EVENT_OID,=)},0)}
      OID_VAR_LIST := (OID_VAR_LIST − {var(oid_var_I,OLD_LIST,0)}) ∪
                      {var(oid_var_I,OLD_LIST∪{ev(EVENT_OID,A_EVENT_OID,=)},0)}
      TOP_EVENT_LIST := TOP_EVENT_LIST ∪ {EVENT_OID}
    end if
    if ap_I,J=(attr_I,J rel_op_I,J const_I,J) then
      P_EVENT_OID := create_p_event("put_",attr_I,J,class_I,oid_var_I,var_I,J)
      AP_EVENT_OID := create_p_event("delete_",attr_I,J,class_I,oid_var_I,var_I,J)
      L_EVENT_OID := create_l_event(P_EVENT_OID,AP_EVENT_OID,rel_op_I,J,const_I,J)
      OID_VAR_LIST := (OID_VAR_LIST − {var(oid_var_I,OLD_LIST,0)}) ∪
                      {var(oid_var_I,OLD_LIST∪{ev(L_EVENT_OID,L_EVENT_OID,=)},0)}
      TOP_EVENT_LIST := TOP_EVENT_LIST ∪ {L_EVENT_OID}
      if op_I = ¬pop_I then
        if var(I,OLD_NEG,_) ∈ NEGATIVE_LIST then
```

```
          NEGATIVE_LIST := NEGATIVE_LIST - {var(I,OLD_NEG,_)} ∪
                           {var(I,OLD_NEG∪{L_EVENT_OID},0)}
        else
          NEGATIVE_LIST := NEGATIVE_LIST ∪ {var(I,{L_EVENT_OID},0)}
        end if
      end if
    end if
    if ap_I,J=(attr_I,J:var_I,J rel_op_I,J const_I,J) then
      P_EVENT_OID := create_p_event("put_",attr_I,J,class_I,oid_var_I,var_I,J)
      AP_EVENT_OID := create_p_event("delete_",attr_I,J,class_I,oid_var_I,var_I,J)
      L_EVENT_OID := create_l_event(P_EVENT_OID,AP_EVENT_OID,rel_op_I,J,const_I,J)
      OID_VAR_LIST := (OID_VAR_LIST - {var(oid_var_I,OLD_LIST,0)}) ∪
                  {var(oid_var_I, OLD_LIST ∪{ev(L_EVENT_OID,L_EVENT_OID,=)},0)}
      VAR_LIST := VAR_LIST ∪ {var(var_I,J,{ev(P_EVENT_OID,AP_EVENT_OID,=)},0)}
      TOP_EVENT_LIST := TOP_EVENT_LIST∪{P_EVENT_OID,L_EVENT_OID}
    end if
    if ap_I,J=(attr_I,J:var_I,J rel_op_I,J var'_I,J) then
      EVENT_OID := create_p_event("put_",attr_I,J,class_I,oid_var_I,var_I,J)
      A_EVENT_OID := create_p_event("delete_",attr_I,J,class_I,oid_var_I,var_I,J)
      OID_VAR_LIST := (OID_VAR_LIST - {var(oid_var_I,OLD_LIST,0)}) ∪
                   {var(oid_var_I, OLD_LIST ∪{ev(EVENT_OID,A_EVENT_OID,=)},0)}
      VAR_LIST := (VAR_LIST-{var(var'_I,J,OLD_LIST_B,0}) ∪
                {var(var_I,J,{ev(EVENT_OID,A_EVENT_OID,=)},0),
                var(var'_I,J,OLD_LIST_B∪{ev(EVENT_OID,A_EVENT_OID,rel_op_I,J)},0)}
      TOP_EVENT_LIST := TOP_EVENT_LIST ∪ {EVENT_OID}
    end if
    if ap_I,J=(attr_I,J rel_op_I,J var_I,J) then
      EVENT_OID ::= create_p_event("put_",attr_I,J,class_I,oid_var_I,var_I,J)
      A_EVENT_OID ::= create_p_event("delete_",attr_I,J,class_I,oid_var_I,var_I,J)
      VAR_LIST := (VAR_LIST - {var(var_I,J,OLD_LIST_A,0)}) ∪
                {var(var_I,J,OLD_LIST_A∪{ev(EVENT_OID,A_EVENT_OID,rel_op_I,J)},0)}
      OID_VAR_LIST := (OID_VAR_LIST-{var(oid_var_I,OLD_LIST_B,0)}) ∪
                    {var(oid_var_I,OLD_LIST_B∪{ev(EVENT_OID,A_EVENT_OID,=)},0)}
      TOP_EVENT_LIST := TOP_EVENT_LIST ∪ {EVENT_OID}
      if op_I = ¬pop_I then
        if var(I,OLD_NEG,_) ∈ NEGATIVE_LIST then
          NEGATIVE_LIST := NEGATIVE_LIST - {var(I,OLD_NEG,_)} ∪
                           {var(I,OLD_NEG∪{EVENT_OID},0)}
        else
          NEGATIVE_LIST := NEGATIVE_LIST ∪ {var(I,{EVENT_OID},0)}
        end if
      end if
    end if
  end ∀
end ∀
/* Check variables that appear in action */
∀ LIST ∈ {OID_VAR_LIST,VAR_LIST}
  ∀ var(VAR,EVENT_LIST,0) ∈ LIST
    if appears(VAR,action) then
      LIST := (LIST-{var(VAR,EVENT_LIST,0)}) ∪{var(VAR,EVENT_LIST,1)}
    end if
  end ∀
end ∀
/* Second pass - Create 2-input events (first stage) */
∀ LIST ∈ {OID_VAR_LIST,VAR_LIST}
  ∀ var(VAR,EVENT_LIST,ACTION) ∈ LIST
    while length(EVENT_LIST) > 1
      ev(EV1,AEV1,OPER1) := head(EVENT_LIST)
```

```
      ev(EV2,AEV2,OPER2) := head(tail(EVENT_LIST))
      if var(I,{EV2},_) ∈ NEGATIVE_LIST then
        EVENT_TYPE := negative_event
        NEGATIVE_LIST := NEGATIVE_LIST - {var(I,{EV2},_)}
      else
        EVENT_TYPE := positive_event
      end if
      C_EVENT_OID := create_j_event(EVENT_TYPE,VAR,EV1,AEV1,OPER1,
                                                    EV2,AEV2,OPER2,ACTION)
      ∀ LIST1 ∈ {OID_VAR_LIST,VAR_LIST}
        LIST1 := replace(LIST1, {ev(EV1,AEV1,OPER1),ev(EV2,AEV2,OPER2)},
                                  ev(C_EVENT_OID,C_EVENT_OID,=))
      end ∀
      if EVENT_TYPE = positive_event then
        NEGATIVE_LIST := replace(NEGATIVE_LIST,{EV1,EV2},C_EVENT_OID)
      end if
      TOP_EVENT_LIST := TOP_EVENT_LIST - {EV1,EV2} ∪ {C_EVENT_OID}
    end while
    if ACTION = 0 then EVENT_LIST := {}
  end ∀
end ∀
/* Second pass/second stage - create join-less 2-input events
while length(TOP_EVENT_LIST) > 1
  EV1 := head(TOP_EVENT_LIST)
  EV2 := head(tail(TOP_EVENT_LIST))
  if var(I,{EV2},_) ∈ NEGATIVE_LIST then
    EVENT_TYPE := negative_event
    NEGATIVE_LIST := NEGATIVE_LIST - {var(I,{EV2},_)}
  else
    EVENT_TYPE := positive_event
  end if
  C_EVENT_OID := create_c_event(EVENT_TYPE,EV1,EV2)
  TOP_EVENT_LIST := TOP_EVENT_LIST - {EV1,EV2} ∪ {C_EVENT_OID}
  if EVENT_TYPE = positive_event then
    NEGATIVE_LIST := replace(NEGATIVE_LIST,{EV1,EV2},C_EVENT_OID)
  end if
end while
/* Subscribe top-level event to ECA rule
RULE_OID := create_rule(head(TOP_EVENT_LIST),action)
return(RULE_OID)
```

- The following function creates a primitive event, with the specified prefix, and returns its OID.

```
function create_p_event(PREFIX,ATTR,CLASS,OID_VAR,VAR)
  if ∃ OID ∈ extension(primitive_event) ∧ method(EID)=PREFIX+ATTR ∧
       class(EID)=CLASS ∧ when(EID)=after then
    EID := OID
  else
    EID := new_object(primitive_event, [method([PREFIX+ATTR]), class([CLASS]),
                            when([after]), out_token([[OID_VAR,VAR]])])
  end if
return(EID)
```

- The following function creates a logical event and returns its OID.

```
function create_l_event(P_EVENT_OID,AP_EVENT_OID,REL_OP,CONST)
  IN_TOKEN := out_token(P_EVENT_OID)
  IN_TOKEN = [OID_VAR,VAR]
  EID := new_object(logical_event,[in_token([IN_TOKEN]),out_token([[OID_VAR]]),
             condition([VAR REL_OP CONST]),signal_by([P_EVENT_OID]),
             anti_signal_by([AP_EVENT_OID])])
  signal(P_EVENT_OID) := EID
```

```
  anti_signal(AP_EVENT_OID) := EID
return(EID)
```

- The following function creates a “join” event and returns its OID.

```
function create_j_event(EVENT_TYPE,VAR,EV1,AEV1,OPER1,EV2,AEV2,OPER2,ACTION)
  LEFT_IN_TOKEN := out_token(EV1)
  RIGHT_IN_TOKEN := out_token(EV2)
  LVAR := (VAR ∈ LEFT_IN_TOKEN)
  RVAR := (VAR ∈ RIGHT_IN_TOKEN)
  if ACTION = 1 then
    OUT_TOKEN := (LEFT_IN_TOKEN - {VAR}) ∪ RIGHT_IN_TOKEN
  else
    OUT_TOKEN := (LEFT_IN_TOKEN ∪ RIGHT_IN_TOKEN) - {VAR}
  end if
  LR_OPER := comparison(OPER1,OPER2)
  RL_OPER := comparison(OPER2,OPER1)
  EID := new_object(EVENT_TYPE, [left_in_token([LEFT_IN_TOKEN]),
      right_in_token([RIGHT_IN_TOKEN]),lr_join([[var(VAR,LVAR LR_OPER RVAR)]]),
      rl_join([[var(VAR,RVAR RL_OPER LVAR)]]),out_token([OUT_TOKEN]),
      left_signal_by([EV1]), left_anti_signal_by([AEV1]), right_signal_by([EV2]),
      right_anti_signal_by([AEV2])])
  signal(EV1) := EID
  signal(EV2) := EID
  anti_signal(AEV1) := EID
  anti_signal(AEV2) := EID
return(EID)
```

- The following function creates a “cartesian-product” event and returns its OID.

```
function create_c_event(EVENT_TYPE,EV1,EV2)
  LEFT_IN_TOKEN := out_token(EV1)
  RIGHT_IN_TOKEN := out_token(EV2)
  OUT_TOKEN := LEFT_IN_TOKEN ∪ RIGHT_IN_TOKEN
  EID := new_object(EVENT_TYPE, [left_in_token([LEFT_IN_TOKEN]),
           right_in_token([RIGHT_IN_TOKEN]), out_token([OUT_TOKEN]),
           left_signal_by([EV1]), left_anti_signal_by([EV1]),
           right_signal_by([EV2]), right_anti_signal_by([EV2])])
  signal(EV1) := EID
  signal(EV2) := EID
  anti_signal(EV1) := EID
  anti_signal(EV2) := EID
return(EID)
```

- The following function creates an ECA rule and returns its OID.

```
function create_rule(EVENT,ACTION)
  EID := new_object(production_rule, [signal_by([EVENT]),
                anti_signal_by([EVENT]), condition([true]), action([ACTION])])
  signal(EVENT) := EID
  anti_signal(EVENT) := EID
return(EID)
```

- The following function provides the comparison operator for a left-to-right comparison.

```
function comparison(OPER1,OPER2)
  if OPER1="=" and OPER2="=" then OPER := "="
    else if OPER1="=" then OPER := inverse(OPER2)
```

```
     else OPER := OPER2
return(OPER)
```

- The following function inverses the direction of a comparison operator.

```
function inverse(OPER)
select case OPER
  case OPER=">" : INV := "<"
  case OPER=">=" : INV := "=<"
  case OPER="<" : INV := ">"
  case OPER="=<" : INV := ">="
  case OPER="\=" : INV := "\="
end select
return(INV)
```

- The following function replaces in a variable list all occurrences of old events with a new event.

```
function replace(VAR_LIST,OLD_EVENTS,NEW_EVENT)
  NEW_VAR_LIST := VAR_LIST
  ∀ var(VAR,EVENT_LIST,ACTION) ∈ VAR_LIST
    NEW_EVENT_LIST := (EVENT_LIST - OLD_EVENTS) ∪ {NEW_EVENT}
    NEW_VAR_LIST := (NEW_VAR_LIST - {var(VAR,EVENT_LIST,ACTION)}) ∪
                                        {var(VAR,NEW_EVENT_LIST,ACTION)}
  end ∀
return(NEW_VAR_LIST)
```

- The function `new_object(`*`class`*`,`*`attributes`*`)` creates an instance of *`class`*, with attributes in *`attributes`* and returns its OID. This function is provided by the core active OODB system.
- The function `extension(`*`class`*`)` returns the set of instances of *`class`*.
- The function `appears(`*`var`*`,`*`prolog_code`*`)` checks if variable *`var`* appears anywhere in *`prolog_code`*. This function is straight forward to implement in Prolog.