Automated Generation of Abstract Web Models using QVT Relations

Ali Fatolahi Stéphane S. Somé Timothy C. Lethbridge

afato092@site.uottawa.ca ssome@site.uottawa.ca tcl@site.uottawa.ca

School of Information Technology and Engineering

University of Ottawa

Abstract

Transformations play a pivotal role in the implementation of Model-driven development (MDD) methods by providing a mechanism to express model refinement. The QVT relation language is a part of the OMG standard for formalizing model-driven transformations. In this document, three sets of QVT relations are presented. The first set transforms a high-level input model to an abstract web-specific model. The second and the third sets transform the abstract web model to specific web platform models.

1 Introduction

The three conventional levels of modeling in MDA [1] are: the Computation Independent Model (CIM), the Platform Independent Model (PIM), and the Platform Specific Model (PSM). We have added an intermediate level termed Abstract PSM (APSM) [8]. The APSM, an intermediate between the PIM and PSM in the context of Web applications, is concerned with models specified with respect to common features of different web platforms. Such features form an abstract platform [2]. The APSM is platform-specific in the sense that it describes features specific to the abstract web platform; it is also abstract since it does not contain details of specific web platforms but only their shared features. Two sets of transformations are needed from PIM to PSM: one to map PIM to APSM and the second one to map APSM to PSM.

An important aspect of our work is the usage of QVT relations [3], a standard that has not been properly discussed in the literature so far. The QVT standard suggests two ways of developing transformations: Operational and Relational. The former has received more attention because it tends to be similar to the conventional programming languages. The latter, however, has the advantage of making the development of bi-directional transformations easier [4]. The relations are visualized using Petri-Nets [5] based on the approach presented in [6]. The input model is defined using the visual language of [7]. The relations are based on the meta-model provided in [8].

This document is organized as follows. Section 2 defines the transformations from PIM to APSM. Section 3 defines the transformations from APSM to a specific platform based on AndroMDA [9]. Section 4 defines the transformations from APSM to a platform based on WebRatio [10]. Section 5 defines the transformations from APSM to Google Web platform and finally, Section 6 concludes this report.

2 PIM-to-APSM Relations

In this section, the mainstream relations that create the APSM models are listed. Code sample 1 presents a top relation, where presentation states are mapped to presentation states at the target. The input model is checked for all instances of states, termed *state1* that have one or more presentations, *presentation1*. As such, *state2* and *presentation2* are created with the same name and attributes as of *state1* and *presentation1* respectively.

Code Sample 1 – Mapping a Presentation State

```
top relation
  presentationState_presentationState {
   stateName:String;
   machineName:String;

   checkonly domain wPIM state1:
    wpim::State {
     name=stateName,
     stateMachine=
      stateMachine1:wpim::StateMachine {
       name=machineName
     },
    presentation=
     presentation1:wpim::Presentation {
     }
   };

   enforce domain aPSM state2:apsm::State{
    name=stateName,
    stateMachine=
     stateMachine2:apsm::StateMachine {
      name=machineName
     },
    presentation=
       presentation2:apsm::Presentation {
     }
   };

   when {
    statemachine statemachine(
            stateMachine1, stateMachine2);
   }
  }
```

Code Sample 2 presents another relation that maps presentations with a hyperlink. The main difference with Code Sample 1 is that the pattern of the source domain has an extra section for checking UI components contained in *presentation1*; this is named as *hyperLink1*. These are simply mapped to extra outgoing transitions, as opposed to the ones that are directly mapped from PIM, known as *outgoing2* in Code Sample 2.

Code Sample 2 – Mapping a Hyperlink

```
top relation hyperlink hyperLink {
  machineName:String;
  stateName:String;
  triggerName:String;

  checkonly domain wPIM stateMachine1:
   wpim::StateMachine {
    name=machineName,
    states=state1:wpim::State{
```

```
    name=stateName,
    presentation=
     presentation1:wpim::Presentation {
     uiComponents=
      hyperLink1:wpim::HyperLink {
       name=triggerName
      }
    },
   outgoing=outgoing1:wpim::Transition {
     target=target1:wpim::State {
     }
    }
   }
  };

 enforce domain aPSM stateMachine2:
  apsm::StateMachine {
   name=machineName,
   states=state2:apsm::State {
    stateMachine=stateMachine2,
     name=stateName,
    outgoings=
     outgoing2:apsm::Transition{
      stateMachine=stateMachine2,
       name=triggerName,
       source=state2,
       target=
        state2_1:apsm::FinalState {
         stateMachine=stateMachine2,
         incomings=addTransition(
          state2.incomings
          ,outgoing2),
      name=triggerName
     }
    }
   }
  };
 }
```

Code Sample 3 presents a chain of relations that copy the application, use cases and state machines respectively. Finally, the start state of the APSM state machine is created plus its outgoing transitions and their targets. For brevity, the details of these transitions are removed and connection points are only listed.

The relation *application_application* maps a PIM application to an APSM one and calls for the relation *usecase_usecase* to map the use cases of the applications as well. Once use cases are mapped, the relation *statemachine_statemachine* is validated for mapping state machines. Finally, the relation *starttransition_starttransition* maps the start state of every state machine. This relation also maps the transitions outgoing the start state as well as their target states to the APSM.

Code Sample 3 – The Chain of Relations that create the General Structure of the APSM Web Application

```
top relation application application {

   checkonly domain wPIM application1:
    wpim::Application {
     useCases=usecase1:wpim::UseCase{}
    };

   enforce domain aPSM application2:
    apsm::Application {
       useCases=usecase2:apsm::UseCase{}
   };
```

```
   where {
    usecase_usecase(usecase1, usecase2);
   }
}

relation usecase_usecase {

  checkonly domain wPIM usecase1:
   wpim::UseCase{}
  stateMachine = stateMachine1:
   wpim::StateMachine {},
  };

  enforce domain aPSM usecase2 :
    apsm::UseCase {
     stateMachine = stateMachine2:
      apsm::StateMachine {},
     },
  };

  where {
    statemachine_statemachine(
            stateMachine1, stateMachine2);
  }
}

relation statemachine_statemachine {

 checkonly domain wPIM statemachine1 :
   wpim::StateMachine {};

 enforce domain aPSM statemachine2 :
   apsm::StateMachine {};

 where {
    starttransition starttransition(
          statemachine1, statemachine2);
}

relation starttransition_starttransition {

  checkonly domain wPIM stateMachine1:
   wpim::StateMachine {
    states=source1:wpim::InitialState {
     outgoing=outgoing1:
      wpim::Transition {
       target=target1:wpim::State {
     }
    }
   }
  };

  enforce domain aPSM stateMachine2:
   apsm::StateMachine {
    name=machineName,
    states=source2:apsm::StartState {
     outgoings=outgoing2:
      apsm::Transition {
       source=source2,
       target=target2:apsm::State {
       incomings=
        incoming2:apsm::Transition {
```

```
            source=source2,
            target=target2
        }
       }
     }
    }
   };
}
```

2.1 The Core Relation

For the most part, the relations listed as Code Sample 1-3are basic relations that generate one-to-one mappings of the elements of PIM in the APSM. Code Sample 4 presents a pivotal relation in the transformations that leads to the generation of the details added to the APSM compared to the PIM. This relation named *first_presentationAssociatioins_signalEvents,* verifies the data associated with the UI components and builds the behavioral states, transitions and events related to those. The relation maps every data association from PIM, *dataAssociation1*, to a new state, *target2_1* in PSM.

At the end of this relation a call event, *callEvent2* is created on the transition outgoing from the newly created state. This call event is then sent to the relation *dataAsscoiation_callEvent* along with *dataAsscoaition1* in order to create other details of the operation call that are the controller operation, its parameters, data services and the dependencies of the operation controller to services.

The relation goes through a number of logical tests to decide the target of the transitions carrying the call event. This could be 1) the final states, 2) a choice or 3) further states and transitions; more states and transitions are created in case there are more than one data associations. The first is verified by checking the target of the transition in the original PIM model, *target1*. The second and the third are verified through investigating the *next* attribute of data associations. If this is empty, then the relation *trueChoice_nextState* is checked in order to build the choice, transitions and their guards. Otherwise, the relation *next_presentationAssociations_signalEvent* is checked to repeat the same process for the next data association.

Elaborating the Relation

The *first_presentationAssociations_signalEvent* relation is elaborated in this section.

Code Sample 4-1 – Checking the PIM

```
1:top relation
2:first_presentationAssociations_signalEvents
{
3:stateName:String;
4:machineName:String;
5:orderNumber:String;

6:checkonly domain wPIM uiAssociation1              :wpim::UIComponentAssociation {
7: isFirst=true,
8: owner=
9:  operationTrigger1:
     wpim::OperationTrigger {
10:    presentation=
       presentation1:wpim::Presentation {
11:       presentationState=state1:
          wpim::State {
12:        name=stateName,
13:        outgoing=outgoing1:
           wpim::Transition {
14:          target=target1:wpim::State {}
15:        },
16:        stateMachine=
```

```
          stateMachine1:
           wpim::StateMachine {
17:          name=machineName
18:         }
19:        }
20:      }
21:     },
22:   order=orderNumber
23:};
```

Lines 1-2 declare the relation. Being a top relation means the execution of transformation starts at this relation.
Lines 3-5 declare the variables required to map the names of states and state machines as well as data associations. The variable *orderNumber* is a special one that is used to indicate the number of data associations. This number is used to label the states that are automatically added at the PSM level.
Line 6 checks the existence of the association with the UI.
Line 7 ensures that the association is the first in the list.
Lines 8-11 span the ownership hierarchy to reach the owning state.
Line 12 sets the variable *stateName.*
Lines 13-15 obtain the outgoing transitions and their targets.
Lines 16-18 reach the owning state machine and set the variable *machinName*.
Line 22 sets the variable *orderNumber*.

Code Sample 4-2 –Creating the APSM Main State

```
24:enforce domain aPSM state2
                        :apsm::State {
25:   stateMachine=
     stateMachine2:apsm::StateMachine {
26:      name=machineName
27:    },
28:    name=stateName,
    ...
```

Lines 24-28 declare the target state that matches the original presentation state in the source. Then the *name* and the *stateMachine* property of this state are set.

Code Sample 4-3 – Transition Outgoing the Main State and An Extra Logic State added here

```
29:outgoings=outgoing2:apsm::Transition {
30:stateMachine=stateMachine2,
31:source=state2,
32:target=state2_1:apsm::State {
33: incomings=
    addTransition(state2_1.incomings,
                  outgoing2),
34:     stateMachine=
     stateMachine2:apsm::StateMachine {
35:       name=machineName,
36:       states=
          stateMachine2.states
            ->append(state2_1)
37:     },
38:    name='runState1',
    …
```

Lines 29-38 handle the creation of a new state in the target without an equivalent in the source.
Line 29 creates the transition outgoing from *state2* that was created in line 24.
Lines 30-31 set the *stateMachine* and the source properties of the transition.
Line 32 declares the new state, *target2_1* as the target of the transition.
Lines 34-35 assign the *stateMachine* attribute of the new state.
Line 36 adds this state to the state machine.

Line 38 sets the name of the state, which matches a state with the same name.

Code Sample 4-4 – Transition Outgoing the Extra State and the Target Choice are added here

```
39: outgoings=
           outgoing2_1:apsm::Transition {
40: stateMachine=stateMachine2,
41: source=state2 1,
42: callEvents=
           callEvent2:apsm::CallEvent {
43: owner=outgoing2_1
44: },
45: target=choice1:apsm::Choice {
46: statMachine=
      stateMachine2:apsm::StateMachine {
47: name=machineName,
48: states=
    stateMachine2.states->append(choice1)
49: },
50: incomings=
         addTransition(choice1.incomings,
                  outgoing2_1),
51: name='choice'+orderNumber,
```

Lines 39-51 create the transition outgoing from the state, *runState1*, the call event to handle the operation and the choice that evaluates the return value of the operation:
Lines 39-41 create the outgoing transition and define its state machine and source values.
Lines 42-44 define the call event.
Line 45 defines a choice as the target of the transition.
Lines 46-49 assign the state machine value of the choice and then append the choice to the states collection of the state machine.
Line 50 adds the transition from line 39 to the incoming transitions of the choice.
Line 51 sets the name of the choice based on the value of the variable *orderNumber*.

Code Sample 4-5 – Transition Outgoing the Choice and the 'True' Outgoing

```
52: outgoings=
         outgoing2 1 1:apsm::Transition {
53: stateMachine=stateMachine2,
54: source=choice1,
55: guard=gurad21:apsm::Guard {
56:                          name='true'
57:   }
58: },
```

Lines 52-58 define a transition outgoing from the choice that carries the guard *true.* That is, this transition would be taken when the return value from the operation called by the event of lines 42-44 returns a *true* value.

Code Sample 4-6 – The 'False' Guard and the Target Error State

```
59: outgoings=
         outgoing2_1_2:apsm::Transition {
60: stateMachine=stateMachine2,
61: source=choice1,
62: guard=gurad22:apsm::Guard {
63:   name='false'
64: },
65: target=state2_1_2:apsm::State {
66: stateMachine=
       stateMachine2:apsm::StateMachine {
67:   name=machineName,
68:   states=
            stateMachine2.states->
```

```
                      append(state2_1_2)
69: },
70: incomings=
      addTransition(state2 1 2.incomings,
                    outgoing2_1_2),
71: name='Error State',
72: presentation=
      presentation2:apsm::Presentation {
73: name='Error Page',
74: uiComponents=
      errorMessage2:apsm::StaticText {
75:     name='Error Performing Operation'
76:    }
77: },
```

Lines 59-77 create another transition outgoing from the choice that is guarded by the value *false*. This transition leads to another state in line 65, which shows an error message, this error message is created in lines 74-76. This state has two outgoing transitions.

Code Sample 4-7 –The Retry Links from the Error State

```
78: outgoings=
      outgoings2_1_2_1:apsm::Transition {
79:     stateMachine=stateMachine2,
80:     source=state2_1_2,
81:     signalEvent=
        signalEvent21:apsm::SignalEvent {
82:      transition=outgoings2_1_2_1,
83:       name='retry'
84:     },
85:   target=state2
86: },
```

The first transition is created in lines 78-86, where a transition with a signal event, *retry* is added. This transition leads to the main presentations state, *state2*, this is equivalent to the state.

Code Sample 4-8 –The Cancel Link from the Error State ending at the Final State

```
87: outgoings=
     outgoings2 1 2 2:apsm::Transition {
88:    stateMachine=stateMachine2,
89:    source=state2 1 2,
90:    signalEvent=
        signalEvent22:apsm::SignalEvent {
91:      transition=outgoings2_1_2_2,
92:      name='cancel'
93:     },
94: target=finalState2:apsm::FinalState {
95:   name='finalState',
96:   stateMachine=
       stateMachine2:apsm::StateMachine {
97:     name=machineName,
98:     states=stateMachine2.states->
                      append(finalState2)
99:    },
100:  incomings=
      addTransition(finalState2.incomings,
       outgoings2_1_2_2)
```

Lines 87-100 defined the second transition outgoing from the error state. This transition carries another signal event *cancel* and leads to a final state.

Code Sample 4-9 –The Post-Conditions and Aftermath Relations

```
101:  where {
102:   dataAssociation_callEvent(
```

```
                      uiAssociation1,
                      callEvent2) or
103:   fieldAssociation_callEvent(
                      uiAssociation1,
                      callEvent2);
104:  operationTrigger_signalEvent(
                    operationTrigger1,
                    outgoing2);
105:  if (not
       uiAssociation1.next->isEmpty())
        then
next_presentationAssociations_signalEvents
                     (uiAssociation1.next,
                     outgoing2 1 1)
106:  else if (not
                    target1->oclIsTypeOf(
                  wpim::FinalState)) then
            trueChoice_nextState(target1,
            stateMachine2, outgoing2_1_1)
107:  else
      presentationAssociations_finalState(
                      stateMachine1,
                      stateMachine2,
                      '', outgoing2_1_1)
108: endif
109:endif;
110:}
```

Finally, lines 101-110 check the post-conditions of the relation:
Line 102 checks the creation of the call event through the relation *dataAssociation_callEvent*.
Line 103 runs the same validation but this time for an association between data and a UI field rather than operation trigger.
Line 104 runs a relation to map the operation trigger to a signal event. This is necessary because the current relation does not create signal events at the target.
Line 105 checks if there is more than one data item associated with the operation trigger. If yes, the *next_presentationAssociations_signalEvents* relation is called to repeat the same mappings for the next association.
Lines 106-107 map the target state of the *true* transition outgoing from the choice to either a final state if the target of the main presentation state is a final state in the source model; or a regular state otherwise.

2.2 Uniqueness of the Elements

It is important to guarantee the uniqueness of the elements created at the APSM level. For example, both relations *presentationState_presenationState* and *hyperlink_hyperLink* enforce creating presentation states at the target. Without a proper mechanism, the target model would be full of unexpected redundancies resulting from different relations enforcing the same types of elements. The default mechanism of preventing such redundancies is the usage of key statements that define the uniqueness key of each element. For example, Code Sample 5 is used to define the uniqueness of a transition:

Code Sample 5 – The Key Statement of the APSM Transition

```
key apsm::Transition
          {stateMachine, source, target};
```

However, in some cases the key mechanism may not be enough. This is when more than one top relation exists. The same element might be created at some point in the chain of more than one top relation. In such cases, a relation in the *When* section can help preventing the redundancy because it imposes a precondition to the relation, which can be the uniqueness of the target elements. For example, the key of a state is de-

fined as its name and its owning state machine. In relation *presentationState_presentationState*, however, we have to impose the precondition of the uniqueness of state machines by calling the relation *stateMachine_stateMachine* in the when section. This is because the relation *presentationState_presentationState* is a top relation and therefore does not have any knowledge of possibly created state machines through other top relations such as the relation *application_application* in Code Sample 3.

Code Sample 6 – Using the 'When' Statement for Uniqueness of the Elements

```
top relation
 presentationState_presentationState {
  stateName:String;
  machineName:String;

  checkonly domain wPIM
    state1:wpim::State {
      name=stateName,
      stateMachine=
      stateMachine1:wpim::StateMachine {
       name=machineName
      },
     presentation=
     presentation1:wpim::Presentation {}
  };

  enforce domain aPSM state2:apsm::State
  {
   name=stateName,
   stateMachine=
    stateMachine2:apsm::StateMachine {
     name=machineName
    },
   presentation=
    presentation2:apsm::Presentation {}
  };

  when {
   statemachine_statemachine
    (stateMachine1, stateMachine2);
  }
 }
```

2.3 Functions

QVT allows defining functions to help the developers achieve goals that may not be possible using relations. Functions are mainly used for building values, elements or variables that occur within the target domain. Thus, in most cases creating a variable that depends only on the target domain is something to be considered done by functions. For example, in line 33 of the code in Section 2.2.1, a function *addTransition* is used to add a transition to the set of incoming transitions of *state2_1*. Following is the implementation of this function.

Code Sample 7 – Adding a transition to a set of transitions using a QVT function implemented in MediniQVT

```
1:query addTransition(
2: transitions:
3:          OrderedSet(apsm::Transition),
4: transition:
5:           apsm::Transition):
6:           OrderedSet(apsm::Transition)
7: {
8: transitions->append(transition)
9:}
```

Note that our QVT tool, mediniQVT [6] uses the keyword *query* for indicating a function:

- In Line 1, the function name is declared.
- Line 2 declares one of the parameters, *transitions*. The type of this parameter is declared as a set of APSM transitions in Line 3.
- Line 4 declares another parameter, *transition* of type APSM *Transition* (line 5).
- Line 6 defines the return type of the function, which is the same as the parameter *transitions*.
- Line 8 defines the body of the function using the OCL function *append* that adds an object to an ordered set and returns the updated ordered set as a result.

2.4 Full Listing of the PIM-to-APSM Relations

```
transformation wpim_apsm(wPIM:wpim, aPSM:apsm) {
  key apsm::Application{name};
  key apsm::Service{name};
  key apsm::UseCase{name};
  key apsm::StateMachine{name};
  key apsm::State{name, stateMachine};
  key apsm::Controller{name};
  key apsm::Operation{name, class};
  key apsm::Transition{stateMachine, source, target};
  key apsm::DataComposite{name};
  key apsm::DataEntity{name};
  key apsm::SignalEvent{name, transition};
  key apsm::PageVariable{transition};
  key apsm::Parameter{name, behavior};
  key apsm::Actor{name};

  top relation application_application {
   applicationName:String;
   ucName:String;

   checkonly domain wPIM application1:wpim::Application {
    name=applicationName,
       useCases=usecase1:wpim::UseCase{name=ucName,application=application1}
   };

   enforce domain aPSM application2:apsm::Application {
    name=applicationName,
       useCases=usecase2:apsm::UseCase{name=ucName,application=application2}
   };

    where {
    usecase usecase(usecase1, usecase2);
    }
  }

  top relation baseTrigger_pageVariable {
   machineName:String;
   stateName1:String;
   stateName2:String;
   orderNumber:String;

   checkonly domain wPIM stateMachine1:wpim::StateMachine {
    name=machineName,
```

```
  states=state11:wpim::State {
   name=stateName1,
   presentation=presentation11:wpim::Presentation {
    uiComponents=operationTrigger1:wpim::OperationTrigger {
     uiAssociations=uiAssociation1:wpim::UIComponentAssociation {
      isFirst=true,
      order=orderNumber
     }
    }
   },
   outgoing=outgoing1:wpim::Transition {
    target=target1:wpim::State {
    }
   }
  },

  states=state12:wpim::State {
   name=stateName2,
   presentation=presentation12:wpim::Presentation {
    base=operationTrigger1
   }
  }
 };

 enforce domain aPSM stateMachine2:apsm::StateMachine {
  name=machineName,
  states=state2:apsm::State {
   stateMachine=stateMachine2,
   name=stateName1,
   outgoings=outgoing2:apsm::Transition {
    signalEvent=pageVariable2:apsm::PageVariable {
     transition=outgoing2
    },
    stateMachine=stateMachine2,
    source=state2,
    target=state2_1:apsm::State {
     stateMachine=stateMachine2,
     name='runState1',
     outgoings=outgoing2 1:apsm::Transition {
      stateMachine=stateMachine2,
      source=state2_1,
      signalEvent=pageVariable2_1:apsm::PageVariable {
       transition=outgoing2_1
      },
      target=choice1:apsm::Choice {
       stateMachine=stateMachine2,
       name='choice'+orderNumber,
       outgoings=outgoing2_1_1:apsm::Transition {
        stateMachine=stateMachine2,
        source=choice1,
        signalEvent=pageVariable2_1_1:apsm::PageVariable {
         transition=outgoing2_1_1
        },
        guard=gurad21:apsm::Guard {
         name='true'
        },
        target=target2:apsm::State {
         stateMachine=stateMachine2,
         name=stateName2
        }
       }
      }
     }
```

```
      }
     }
    }
   };

   where {
    if (not uiAssociation1.next->isEmpty()) then
     true
    else if (not target1->oclIsTypeOf(wpim::FinalState)) then
       if (target1=state12) then
        pageVariable_trueChoice_nextState(state12, stateMachine2, choice1, outgoing2_1_1)
       else
        pageVariable_trueChoice_nextState(target1, stateMachine2, choice1, outgoing2_1_1)
       endif
     else
      true
     endif
    endif;
   }
}


relation pageVariable_trueChoice_nextState {
 stateName:String;

 checkonly domain wPIM target1:wpim::State {
  name=stateName,
  stateMachine=stateMachine1:wpim::StateMachine {}
 };

 primitive domain stateMachine2:apsm::StateMachine;
 primitive domain source2:apsm::State;

 enforce domain aPSM transition2:apsm::Transition {
  stateMachine=stateMachine2,
  source=source2,
  target=target2:apsm::State {
   name=stateName,
   stateMachine=stateMachine2
  },
  signalEvent=pageVariable2:apsm::PageVariable {
   transition=transition2
  }
 };
}

relation usecase_usecase {
  usecaseName : String;
  actorName:String;
  machineName:String;

  checkonly domain wPIM usecase1 : wpim::UseCase {
    name = usecaseName,
    stateMachine = stateMachine1:wpim::StateMachine {
     name=machineName,
     useCase=usecase1
    },
    actors=actor1:wpim::Actor {name=actorName}
  };

  enforce domain aPSM usecase2 : apsm::UseCase {
    name = usecaseName,
    stateMachine = stateMachine2:apsm::StateMachine {
```

```
      name=machineName,
      useCase=usecase2
     },
     actors=actor2:apsm::Actor {
      name=actorName,
      useCases=addUseCase(actor2, usecase2)
     },
     controller=controller2:apsm::Controller{name=usecaseName+'Controller'}
   };
   where {
    statemachine_statemachine(stateMachine1, stateMachine2);
   }
}

relation statemachine statemachine {
  statemachineName : String;
  stateName:String;
  transitionName:String;

   checkonly domain wPIM statemachine1 : wpim::StateMachine {
     name = statemachineName,
     states=initialState1:wpim::InitialState {
      stateMachine=statemachine1
     },
     states=state1:wpim::State {
      name=stateName,
      stateMachine=statemachine1
     },
  states=finalState1:wpim::FinalState {
   incoming=transition1:wpim::Transition {
    name=transitionName
   }
  }
   };

   enforce domain aPSM statemachine2 : apsm::StateMachine {
     name = statemachineName,
     states=initialState2:apsm::StartState {
      name='startState',
      stateMachine=statemachine2
     },
     states=state2:apsm::State {
      name=stateName,
      stateMachine=statemachine2
     },
  states=finalState2:apsm::FinalState {
   name=transitionName,
   stateMachine=statemachine2
  },
  states=finalState3:apsm::FinalState {
   name='finalState',
   stateMachine=statemachine2
  }
   };

   where {
    starttransition_starttransition(statemachine1, statemachine2);
   }
}

top relation presentationState_presentationState {
  stateName:String;
  machineName:String;
```

```
 checkonly domain wPIM state1:wpim::State {
  name=stateName,
  stateMachine=stateMachine1:wpim::StateMachine {
   name=machineName
  },
  presentation=presentation1:wpim::Presentation {
  }
 };

 enforce domain aPSM state2:apsm::State {
  name=stateName,
  stateMachine=stateMachine2:apsm::StateMachine {
   name=machineName
  },
  presentation=presentation2:apsm::Presentation {
  }
 };

 when {
  statemachine_statemachine(stateMachine1, stateMachine2);
 }
}

relation starttransition starttransition {
 targetName:String;
 machineName:String;

 checkonly domain wPIM stateMachine1:wpim::StateMachine {
  name=machineName,
  states=source1:wpim::InitialState {
   outgoing=outgoing1:wpim::Transition {
    target=target1:wpim::State {
     name=targetName
    }
   }
  }
 };

 enforce domain aPSM stateMachine2:apsm::StateMachine {
  name=machineName,
  states=source2:apsm::StartState {
   name='startState',
   stateMachine=stateMachine2,
   outgoings=outgoing2:apsm::Transition {
    stateMachine=stateMachine2,
    source=source2,
    target=target2:apsm::State {
     stateMachine=stateMachine2,
     name=targetName,
     incomings=incoming2:apsm::Transition {
      stateMachine=stateMachine2,
      source=source2,
      target=target2
     }
    }
   }
  }
 };
}

relation transition transition {
 targetName:String;
```

```
  machineName:String;

  checkonly domain wPIM source1:wpim::State {
   stateMachine=stateMachine1:wpim::StateMachine {
    name=machineName
   },
   outgoing=outgoing1:wpim::Transition {
    target=target1:wpim::State {
     name=targetName
    }
   }
  };

  primitive domain stateMachine2:apsm::StateMachine;

  enforce domain aPSM source2:apsm::State {
   stateMachine=stateMachine2,
   outgoings=outgoing2:apsm::Transition {
    stateMachine=stateMachine2,
    source=source2,
    target=target2:apsm::State {
     stateMachine=stateMachine2,
     name=targetName,
     incomings=incoming2:apsm::Transition {
      stateMachine=stateMachine2,
      source=source2,
      target=target2
     }
    }
   }
  };
}

top relation hyperlink hyperLink {
 machineName:String;
 stateName:String;
 triggerName:String;

 checkonly domain wPIM stateMachine1:wpim::StateMachine {
  name=machineName,
  states=state1:wpim::State{
   name=stateName,
   presentation=presentation1:wpim::Presentation {
    uiComponents=hyperLink1:wpim::HyperLink {
     name=triggerName
    }
   },
   outgoing=outgoing1:wpim::Transition {
    target=target1:wpim::State {
    }
   }
  }
 };

 enforce domain aPSM stateMachine2:apsm::StateMachine {
  name=machineName,
  states=state2:apsm::State {
   stateMachine=stateMachine2,
   name=stateName,
   outgoings=outgoing2:apsm::Transition {
    stateMachine=stateMachine2,
    name=triggerName,
    source=state2,
```

```
      target=state2_1:apsm::FinalState {
       stateMachine=stateMachine2,
       incomings=addTransition(state2.incomings, outgoing2),
       name=triggerName
      }
     }
    }
   };
}

relation transition_finalState {
 finalName:String;

 checkonly domain wPIM transition1:wpim::Transition {
  source=state1:wpim::State {
   presentation=presentation1:wpim::Presentation {
    uiComponents=hyperLink1:wpim::HyperLink {
     name=finalName
    }
   }
  }
 };

 primitive domain stateMachine2:apsm::StateMachine;

 enforce domain aPSM transition2:apsm::Transition {
  name=finalName,
  target=state2:apsm::FinalState {
   stateMachine=stateMachine2,
   incomings=addTransition(state2.incomings, transition2),
   name=finalName
  },
  stateMachine=stateMachine2
 };
}

top relation first presentationAssociations signalEvents {
 stateName:String;
 machineName:String;
 orderNumber:String;

 checkonly domain wPIM uiAssociation1:wpim::UIComponentAssociation {
  isFirst=true,
  owner=operationTrigger1:wpim::OperationTrigger {
   presentation=presentation1:wpim::Presentation {
    presentationState=state1:wpim::State {
     name=stateName,
     outgoing=outgoing1:wpim::Transition {
      target=target1:wpim::State {}
     },
     stateMachine=stateMachine1:wpim::StateMachine {
      name=machineName
     }
    }
   }
  },
  order=orderNumber
 };

 enforce domain aPSM state2:apsm::State {
  stateMachine=stateMachine2:apsm::StateMachine {
   name=machineName
  },
```

```
name=stateName,
outgoings=outgoing2:apsm::Transition {
 stateMachine=stateMachine2,
 source=state2,
 target=state2_1:apsm::State {
  incomings=addTransition(state2_1.incomings, outgoing2),
  stateMachine=stateMachine2:apsm::StateMachine {
   name=machineName,
   states=stateMachine2.states->append(state2_1)
  },
  name='runState1',
  outgoings=outgoing2_1:apsm::Transition {
   stateMachine=stateMachine2,
   source=state2 1,
   callEvents=callEvent2:apsm::CallEvent {
    owner=outgoing2 1
   },
   target=choice1:apsm::Choice {
    stateMachine=stateMachine2:apsm::StateMachine {
     name=machineName,
     states=stateMachine2.states->append(choice1)
    },
    incomings=addTransition(choice1.incomings, outgoing2_1),
    name='choice'+orderNumber,
    outgoings=outgoing2 1 1:apsm::Transition {
     stateMachine=stateMachine2,
     source=choice1,
     guard=gurad21:apsm::Guard {
      name='true'
     }
    },
    outgoings=outgoing2_1_2:apsm::Transition {
     stateMachine=stateMachine2,
     source=choice1,
     guard=gurad22:apsm::Guard {
      name='false'
     },
     target=state2_1_2:apsm::State {
      stateMachine=stateMachine2:apsm::StateMachine {
       name=machineName,
       states=stateMachine2.states->append(state2_1_2)
      },
      incomings=addTransition(state2_1_2.incomings, outgoing2_1_2),
      name='Error State',
      presentation=presentation2:apsm::Presentation {
       name='Error Page',
       uiComponents=errorMessage2:apsm::StaticText {
        name='Error Performing Operation'
       }
      },
      outgoings=outgoings2_1_2_1:apsm::Transition {
       stateMachine=stateMachine2,
       source=state2_1_2,
       signalEvent=signalEvent21:apsm::SignalEvent {
        transition=outgoings2_1_2_1,
        name='retry'
       },
       target=state2
      },
      outgoings=outgoings2_1_2_2:apsm::Transition {
       stateMachine=stateMachine2,
       source=state2 1 2,
       signalEvent=signalEvent22:apsm::SignalEvent {
```

```
              transition=outgoings2_1_2_2,
              name='cancel'
             },
             target=finalState2:apsm::FinalState {
              name='finalState',
              stateMachine=stateMachine2:apsm::StateMachine {
               name=machineName,
               states=stateMachine2.states->append(finalState2)
              },
              incomings=addTransition(finalState2.incomings, outgoings2 1 2 2)
             }
            }
           }
          }
         }
        }
       }
     },
     incomings=addTransition(state2.incomings, outgoings2_1_2_1)
   };
   when {
    statemachine statemachine(stateMachine1, stateMachine2);
   }
   where {
    dataAssociation callEvent(uiAssociation1, callEvent2) or fieldAssocia-
tion_callEvent(uiAssociation1, callEvent2);
    operationTrigger signalEvent(operationTrigger1, outgoing2);
    if (not uiAssociation1.next->isEmpty()) then
     next_presentationAssociations_signalEvents(uiAssociation1.next, outgoing2_1_1)
    else if (not target1->oclIsTypeOf(wpim::FinalState)) then
       trueChoice_nextState(target1, stateMachine2, outgoing2_1_1)
     else
      presentationAssociations finalState(stateMachine1, stateMachine2, '', out-
going2 1 1)
     endif
    endif;
   }
  }

  relation operationTrigger_signalEvent {
   triggerName:String;
   componentName:String;

   checkonly domain wPIM operationTrigger1:wpim::OperationTrigger {
    name=triggerName,
    uiComponents=uiComponent1:wpim::UIComponent {
     name=componentName
    }
   };

   enforce domain aPSM transition2:apsm::Transition  {
    signalEvent=signalEvent2:apsm::SignalEvent {
     transition=transition2,
     name=triggerName,
     parameters=parameter2:apsm::Parameter {
      name=componentName,
      type=type2:apsm::Type {
       primitiveType=getParameterType(uiComponent1)
      }
     }
    }
   };
  }
```

```
relation trueChoice nextState {
 stateName:String;

 checkonly domain wPIM target1:wpim::State {
  name=stateName
 };

 primitive domain stateMachine2:apsm::StateMachine;

 enforce domain aPSM transition2:apsm::Transition {
  target=target2:apsm::State {
   name=stateName,
   stateMachine=stateMachine2,
   incomings=addTransition(target2.incomings, transition2)
  }
 };
}

relation presentationAssociations finalState {
 machineName:String;

 checkonly domain wPIM stateMachine1:wpim::StateMachine {
  name=machineName
 };

 primitive domain stateMachine2:apsm::StateMachine;

 primitive domain finalName:String;

 enforce domain aPSM transition2:apsm::Transition {
  target=state2:apsm::FinalState {
   stateMachine=stateMachine2,
   incomings=addTransition(state2.incomings, transition2),
   name=finalName
  },
  stateMachine=stateMachine2
 };

 when {
 }
}

relation next_presentationAssociations_signalEvents {
 eventName:String;
 stateName:String;
 machineName:String;
 orderNumber:String;

 checkonly domain wPIM uiAssociation1:wpim::UIComponentAssociation {
  owner=operationTrigger1:wpim::OperationTrigger {
   name=eventName,
   presentation=presentation1:wpim::Presentation {
    presentationState=state1:wpim::State {
     name=stateName,
     outgoing=outgoing1:wpim::Transition {
     },
     stateMachine=stateMachine1:wpim::StateMachine {
      name=machineName
     }
    }
   }
  },
```

```
  order=orderNumber
};

enforce domain aPSM transition2:apsm::Transition {
 target=state2:apsm::State {
  incomings=addTransition(state2.incomings, transition2),
  name='runState'+orderNumber,
  stateMachine=stateMachine2:apsm::StateMachine {
   name=machineName,
   states=stateMachine2.states->append(state2)
  },
  outgoings=outgoing2:apsm::Transition {
   stateMachine=stateMachine2,
   source=state2,
   callEvents=callEvent2:apsm::CallEvent {
    owner=outgoing2
   },
   target=choice2:apsm::Choice {
    stateMachine=stateMachine2:apsm::StateMachine {
     states=stateMachine2.states->append(choice2)
    },
    name=eventName+'?',
    incomings=addTransition(choice2.incomings, outgoing2),
    outgoings=outgoing2_1:apsm::Transition {
     stateMachine=stateMachine2,
     guard=gurad21:apsm::Guard {
      name='false'
     },
     source=choice2,
     target=state2_1:apsm::State {
      stateMachine=stateMachine2:apsm::StateMachine {
       states=stateMachine2.states->append(state2_1)
      },
      incomings=addTransition(state2 1.incomings, outgoing2 1),
      name='Error State '+eventName,
      presentation=presentation2:apsm::Presentation {
       name='Error Page '+eventName,
       uiComponents=errorMessage2:apsm::StaticText {
        name='Error Performing Operation'+eventName
       }
      },
      outgoings=outgoings2_1_1:apsm::Transition {
       stateMachine=stateMachine2,
       source=state2_1,
       signalEvent=signalEvent21:apsm::SignalEvent {
        transition=outgoings2_1_1,
        name='retry'
       },
       target=state2_1_1:apsm::State {
        stateMachine=stateMachine2,
        name=stateName,
        incomings=addTransition(state2_1_1.incomings, outgoings2_1_1)
       }
      },
      outgoings=outgoings2_1_2:apsm::Transition {
       stateMachine=stateMachine2,
       source=state2_1,
       signalEvent=signalEvent22:apsm::SignalEvent {
        transition=outgoings2_1_2,
        name='cancel'
       },
       target=finalState2:apsm::FinalState {
        name='finalState',
```

```
            stateMachine=stateMachine2:apsm::StateMachine {
             states=stateMachine2.states->append(finalState2)
            },
            incomings=addTransition(finalState2.incomings, outgoings2 1 2)
           }
          }
         }
        },
        outgoings=outgoing2_2:apsm::Transition {
         stateMachine=stateMachine2,
         source=choice2,
         guard=gurad22:apsm::Guard {
          name='true'
         }
        }
       }
     }
   },
   stateMachine=stateMachine2
  };


  when {
   statemachine_statemachine(stateMachine1, stateMachine2);
  }
  where {
   dataAssociation callEvent(uiAssociation1, callEvent2);
   if (not uiAssociation1.next->isEmpty()) then
    next_presentationAssociations_signalEvents(uiAssociation1.next, outgoing2_2)
   else
    presentationAssociations_finalState(stateMachine1, stateMachine2, '', outgoing2_2)
   endif;
  }
}

relation dataAssociation_callEvent {
 checkonly domain wPIM dataAssociation1:wpim::DataAssociation {
 };

 enforce domain aPSM callEvent2:apsm::CallEvent {
 };

 where {
  hiddenFieldUpdate_callEvent(dataAssociation1, callEvent2) or
  singleDataUpdate callEvent(dataAssociation1, callEvent2) or
  first_createData_callEvent(dataAssociation1, callEvent2) or
  createData_callEvent(dataAssociation1, callEvent2);
 }
}

relation first_createData_callEvent {
 dataName:String;
 fieldName:String;
 ucName:String;

 checkonly domain wPIM dataAssociation1:wpim::DataAssociation {
  isFirst=true,
  owner=operationTrigger1:wpim::OperationTrigger {
   uiComponents=uiComponent1:wpim::UIComponent {
    name=fieldName
   },
   presentation=presentation1:wpim::Presentation {
    presentationState=state1:wpim::State {
```

```
      stateMachine=stateMachine1:wpim::StateMachine {
       useCase=useCase1:wpim::UseCase {
        name=ucName
       }
      }
     }
    }
   },
   dataOperation=dataOperation1:wpim::DataOperation {
    queryType=wpim::QueryType::create
   },
   dataComposite=dataComposite1:wpim::DataComposite {
    name=dataName
   }
  };

  enforce domain aPSM callEvent2:apsm::CallEvent {
   name='call create'+dataName+'()',
   operation=operation2:apsm::Operation {
    name='create'+dataName,
    parameters=parameter21:apsm::Parameter {
     name=fieldName,
     type=type2_1:apsm::Type {
      primitiveType=getParameterType(uiComponent1)
     },
     behavior=operation2
    },
    returnType=apsm::DataType::boolean,
    class=controller2:apsm::Controller {
     name=ucName+'Controller',
     useCase=useCase2:apsm::UseCase {
      name=ucName
     },
     operations=addOperation(controller2, operation2)
    },
    callee=serviceOperation2:apsm::Operation {
     crudNature=apsm::QueryType::create,
     name='create'+dataName,
     parameters=parameter22:apsm::Parameter {
      name=dataName+'Composite',
      behavior=serviceOperation2,
      type=type2_2:apsm::Type {
       auxiliaryType=dataComposite2:apsm::DataComposite {
        name=dataName,
        dataEntities=dataEntity2:apsm::DataEntity {
         name=dataName
        },
        service=service2:apsm::Service {
         name=dataName+'Service',
         dataComposite=dataComposite2,
         operations=addOperation(service2, serviceOperation2)
        }
       }
      }
     },
     class=service2
    }
   }

  };

  where {
   uiComponents_dataAttributes(operationTrigger1, dataEntity2);
```

```
  }
}

relation createData callEvent {
 dataName:String;
 dataName1:String;
 fieldName:String;
 ucName:String;


 checkonly domain wPIM dataAssociation1:wpim::DataAssociation {
  owner=operationTrigger1:wpim::OperationTrigger {
   uiComponents=uiComponent1:wpim::UIComponent {
    name=fieldName
   },
   presentation=presentation1:wpim::Presentation {
    presentationState=state1:wpim::State {
     stateMachine=stateMachine1:wpim::StateMachine {
      useCase=useCase1:wpim::UseCase {
       name=ucName
      }
     }
    }
   },
   dataAssociations=dataAssociation1 1:wpim::DataAssociation {
    isFirst=true,
    dataComposite=dataComposite1 1:wpim::DataComposite {
     name=dataName1
    }
   }
  },
  dataOperation=dataOperation1:wpim::DataOperation {
   queryType=wpim::QueryType::create
  },
  dataComposite=dataComposite1:wpim::DataComposite {
   name=dataName
  }
 };

 enforce domain aPSM callEvent2:apsm::CallEvent {
  name='call create'+dataName+'()',
  operation=operation2:apsm::Operation {
   name='create'+dataName,
   parameters=parameter21:apsm::Parameter {
    name=dataName1,
    type=type2_1:apsm::Type {
     auxiliaryType=dataComposite21:apsm::DataComposite {
      name=dataName,
      dataEntities=dataEntity21:apsm::DataEntity {
       name=dataName
      },
      service=service21:apsm::Service {
       name=dataName1+'Service',
       operations=operation21:apsm::Operation {
        class=service21,
        name='update'+dataName1+'()'
       },
       dataComposite=dataComposite21
      }
     }
    },
    behavior=operation2
   },
```

```
    returnType=apsm::DataType::boolean,
    class=controller2:apsm::Controller {
     name=ucName+'Controller',
     useCase=useCase2:apsm::UseCase {
      name=ucName
     },
     operations=addOperation(controller2, operation2)
    },
    callee=serviceOperation2:apsm::Operation {
     crudNature=apsm::QueryType::create,
     name='create'+dataName,
     parameters=parameter22:apsm::Parameter {
      name=dataName+'Composite',
      behavior=serviceOperation2,
      type=type2 2:apsm::Type {
       auxiliaryType=dataComposite2:apsm::DataComposite {
        name=dataName,
        dataEntities=dataEntity2:apsm::DataEntity {
         name=dataName
        },
        service=service2:apsm::Service {
         name=dataName+'Service',
         operations=addOperation(service2, serviceOperation2),
         dataComposite=dataComposite2
        }
       }
      }
     },
     class=service2
    }
   }

  };

  where {
   uiComponents_dataAttributes(operationTrigger1, dataEntity2);
  }
}

relation hiddenFieldUpdate_callEvent {
 dataName:String;
 fieldName:String;

 checkonly domain wPIM dataAssociation1:wpim::DataAssociation {
  owner=operationTrigger1:wpim::OperationTrigger {
   uiComponents=uiComponent1:wpim::UIComponent {
    visible=false,
    name=fieldName
   }
  },
  dataOperation=dataOperation1:wpim::DataOperation {
   queryType=wpim::QueryType::update
  },
  dataComposite=dataComposite1:wpim::DataComposite {
   name=dataName
  }
 };

 enforce domain aPSM callEvent2:apsm::CallEvent {
  name='call update'+dataName+fieldName+'()',
  operation=operation2:apsm::Operation {
   name='update'+dataName+fieldName,
   parameters=parameter2:apsm::Parameter {
```

```
      name=fieldName,
      type=type2:apsm::Type {
       primitiveType=getParameterType(uiComponent1)
      }
     },
     returnType=apsm::DataType::boolean,
     callee=serviceOperation2:apsm::Operation {
      crudNature=apsm::QueryType::updateField,
      class=serviceClass2:apsm::Class {
       name=dataName+'Service'
      }
     }
    },
    owner=transition2:apsm::Transition {
     target=choice2:apsm::Choice {
      outgoings=outgoing2:apsm::Transition {
       guard=guard2:apsm::Guard {
        name='false'
       },
       target=target2:apsm::State {
        presentation=presentation2:apsm::Presentation {
         uiComponents=errorMessage2:apsm::StaticText {
          name='Error Updating '+fieldName+ ' of '+dataName
         }
        }
       }
      }
     }
    }
   };

   when {
    uiComponent1.dataAssociations->includes(dataAssociation1);
   }
}

relation singleDataUpdate callEvent {
 dataName:String;
 fieldName:String;
 ucName:String;

 checkonly domain wPIM dataAssociation1:wpim::DataAssociation {
  owner=operationTrigger1:wpim::OperationTrigger {
   uiComponents=uiComponent1:wpim::UIComponent {
    name=fieldName
   },
   presentation=presentation1:wpim::Presentation {
    presentationState=state1:wpim::State {
     stateMachine=stateMachine1:wpim::StateMachine {
      useCase=useCase1:wpim::UseCase {
       name=ucName
      }
     }
    }
   }
  },
  dataOperation=dataOperation1:wpim::DataOperation {
   queryType=wpim::QueryType::update
  },
  dataComposite=dataComposite1:wpim::DataComposite {
   name=dataName
  }
 };
```

```
 enforce domain aPSM callEvent2:apsm::CallEvent {
  name='call update'+dataName+'()',
  operation=operation2:apsm::Operation {
   name='update'+dataName,
   parameters=parameter21:apsm::Parameter {
    name=fieldName,
    type=type2:apsm::Type {
     primitiveType=getParameterType(uiComponent1)
    }
   },
   returnType=apsm::DataType::boolean,
   class=controller2:apsm::Controller {
    useCase=useCase2:apsm::UseCase {
     name=ucName
    }
   },
   callee=serviceOperation2:apsm::Operation {
    name='update'+dataName,
    parameters=parameter22:apsm::Parameter {
     name=dataName+'Composite',
     type=type2:apsm::Type {
      auxiliaryType=dataComposite2:apsm::DataComposite {
       name=dataName,
       dataEntities=dataEntity2:apsm::DataEntity {
        name=dataName
       },
       service=service2:apsm::Service {
        name=dataName+'Service'
       }
      }
     }
    },
    class=service2
   }
  },
  owner=transition2:apsm::Transition {
   target=choice2:apsm::Choice {
    outgoings=outgoing2:apsm::Transition {
     guard=guard2:apsm::Guard {
      name='false'
     },
     target=target2:apsm::State {
      presentation=presentation2:apsm::Presentation {
       uiComponents=errorMessage2:apsm::StaticText {
        name='Error Updating '+dataName
       }
      }
     }
    }
   }
  }

 };

 where {
  uiComponents_dataAttributes(operationTrigger1, dataEntity2);
 }
}

relation fieldAssociation_callEvent {
 checkonly domain wPIM fieldAssociation1:wpim::FieldAssociation {
 };
```

```
 enforce domain aPSM callEvent2:apsm::CallEvent {
 };
}

relation equalFieldAssociation_callEvent {
 fieldName:String;

 checkonly domain wPIM fieldAssociation1:wpim::FieldAssociation {
  actionType=wpim::ActionType::isEqual,
  source=source1:wpim::UIComponent {
   name=fieldName
  }
 };

 enforce domain aPSM callEvent2:apsm::CallEvent {
  name='call areEqual()',
  operation=operation2:apsm::Operation {
   name='areEqual',
   parameters=parameter21:apsm::Parameter {
    name=fieldName+'1',
    type=type2:apsm::Type {
     primitiveType=apsm::DataType::string
    }
   },
   parameters=parameter22:apsm::Parameter {
    name=fieldName+'2',
    type=type2:apsm::Type {
     primitiveType=apsm::DataType::string
    }
   },
   returnType=apsm::DataType::boolean
  },
  owner=transition2:apsm::Transition {
   target=choice2:apsm::Choice {
    outgoings=outgoing2:apsm::Transition {
     guard=guard2:apsm::Guard {
      name='false'
     },
     target=target2:apsm::State {
      presentation=presentation2:apsm::Presentation {
       uiComponents=errorMessage2:apsm::StaticText {
        name=fieldName+'s are not equal'
       }
      }
     }
    }
   }
  }

 };
}

relation emailFieldAssociation_callEvent {
 email:String;
 field:String;

 checkonly domain wPIM fieldAssociation1:wpim::FieldAssociation {
  actionType=wpim::ActionType::email,
  source=source1:wpim::UIComponent {
   name=field
  },
  target=target1:wpim::UIComponent {
```

```
    name=email
   }
  };

  enforce domain aPSM callEvent2:apsm::CallEvent {
   name='call email'+field+'()',
   operation=operation2:apsm::Operation {
    name='email'+field,
    parameters=parameter2:apsm::Parameter {
     name=field,
     type=type2:apsm::Type {
      primitiveType=getParameterType(target1)
     }
    }
   },
   owner=transition2:apsm::Transition {
    target=choice2:apsm::Choice {
     outgoings=outgoing2:apsm::Transition {
      guard=guard2:apsm::Guard {
       name='false'
      },
      target=target2:apsm::State {
       presentation=presentation2:apsm::Presentation {
        uiComponents=errorMessage2:apsm::StaticText {
         name='Email Failed: You provided '+email
        }
       }
      }
     }
    }
   }
  };
}

relation uiComponents_dataAttributes {
  checkonly domain wPIM uiComposite1:wpim::UIComposite {
   uiComponents=uiComponent1:wpim::UIComponent {
   }
  };

  enforce domain aPSM dataEntity2:apsm::DataEntity {
   attributes=attribute2:apsm::Attribute {
   }
  };

  where {
   select_integer(uiComponent1, attribute2) or
   text_string(uiComponent1, attribute2) or
   password_string(uiComponent1, attribute2) or
   check boolean(uiComponent1, attribute2) or
   date_date(uiComponent1, attribute2);
  }
}

relation select_integer {
  componentName:String;

  checkonly domain wPIM uiComponent1:wpim::Select {
   name=componentName
  };

  enforce domain aPSM attribute2:apsm::Attribute {
   name=getAttributeName(componentName).firstToLower(),
```

```
    dataType=apsm::DataType::integer
   };
  }

  relation text_string {
   componentName:String;

   checkonly domain wPIM uiComponent1:wpim::InputField {
    name=componentName
   };

   enforce domain aPSM attribute2:apsm::Attribute {
    name=getAttributeName(componentName).firstToLower(),
    dataType=apsm::DataType::string
   };
  }

  relation password_string {
   componentName:String;

   checkonly domain wPIM uiComponent1:wpim::PasswordField {
    name=componentName
   };

   enforce domain aPSM attribute2:apsm::Attribute {
    name=getAttributeName(componentName).firstToLower(),
    dataType=apsm::DataType::string
   };
  }

  relation check_boolean {
   componentName:String;

   checkonly domain wPIM uiComponent1:wpim::CheckBox {
    name=componentName
   };

   enforce domain aPSM attribute2:apsm::Attribute {
    name=getAttributeName(componentName).firstToLower(),
    dataType=apsm::DataType::boolean
   };
  }

  relation date_date {
   componentName:String;

   checkonly domain wPIM uiComponent1:wpim::Date {
    name=componentName
   };

   enforce domain aPSM attribute2:apsm::Attribute {
    name=getAttributeName(componentName).firstToLower(),
    dataType=apsm::DataType::date
   };
  }

  query setSource(transition:apsm::Transition, state:apsm::State):Boolean {
   transition.source=state
  }

  query removeOutgoing(state:apsm::State, transi-
tion:apsm::Transition):OrderedSet(apsm::Transition) {
    state.outgoings->reject(t|t=transition)
```

```
  }

  query removeIncoming(state:apsm::State, transi-
tion:apsm::Transition):OrderedSet(apsm::Transition) {
   state.incomings->reject(t|t=transition)
  }

  query getDataOperationName(dataOperation:wpim::QueryType):String {
   if (dataOperation=wpim::QueryType::remove) then 'remove'
    else if (dataOperation=wpim::QueryType::create) then 'create'
     else if (dataOperation=wpim::QueryType::update) then 'update'
      else if (dataOperation=wpim::QueryType::select) then 'select'
       else 'selectAll'
      endif
     endif
    endif
   endif
  }

  query getAttributeName(componentName:String):String {
   componentName.split(' ')->iterate(s:String;acc:String=''|acc.concat(s.firstToUpper()))
  }

  query getForeignIDName(dataName:String):String {
   dataName.split(' ')->iterate(s:String;acc:String=''|acc.concat(s.firstToUpper()))
  }

  query addTransition(transitions:OrderedSet(apsm::Transition), transi-
tion:apsm::Transition):OrderedSet(apsm::Transition) {
   transitions->append(transition)
  }

  query addCallEvent(callEvents:OrderedSet(apsm::CallEvent), callE-
vent:apsm::CallEvent):OrderedSet(apsm::CallEvent) {
   callEvents->append(callEvent)
  }

  query getParameterType(uiComponent:wpim::UIComponent):apsm::DataType {
   if (uiComponent->oclIsTypeOf(wpim::InputField)) then apsm::DataType::string
    else if (uiComponent->oclIsTypeOf(wpim::PasswordField)) then apsm::DataType::string
     else apsm::DataType::string
    endif
   endif
  }

  query getCRUDNature(queryType:wpim::QueryType):apsm::QueryType {
   if (queryType=wpim::QueryType::remove) then apsm::QueryType::remove
    else if (queryType=wpim::QueryType::create) then apsm::QueryType::create
     else if (queryType=wpim::QueryType::update) then apsm::QueryType::update
      else if (queryType=wpim::QueryType::select) then apsm::QueryType::selectSingle
       else apsm::QueryType::selectAll
      endif
     endif
    endif
   endif
  }

  query removeState(stateMachine:apsm::StateMachine):OrderedSet(apsm::State) {
   stateMachine.states->reject(s|s->oclIsTypeOf(apsm::State) and s.name->isEmpty())
  }

  query getFinalState(stateMachine:apsm::StateMachine):apsm::FinalState {
```

```
   stateMachine.states->select(s|s->oclIsTypeOf(apsm::FinalState))-
>first().oclAsType(apsm::FinalState)
  }

  query getDataNameOf(dataAssociation:wpim::UIComponentAssociation):String {
   if (dataAssociation->oclIsTypeOf(wpim::DataAssociation)) then
    dataAssociation.oclAsType(wpim::DataAssociation).dataComposite.name
   else
    dataAssociation.oclAsType(wpim::FieldAssociation).source.name
   endif
  }

  query addUseCase(actor:apsm::Actor, useCase:apsm::UseCase):OrderedSet(apsm::UseCase) {
   actor.useCases->append(useCase)
  }

  query addOperation(class:apsm::Class, opera-
tion:apsm::Operation):OrderedSet(apsm::Operation) {
   class.operations->append(operation)
  }

  query getNumberOfFinalStates(stateMachine:apsm::StateMachine):String {
   if stateMachine.states->select(s|s->oclIsTypeOf(apsm::FinalState))->size()=0 then
    '0'
   else
    if stateMachine.states->select(s|s->oclIsTypeOf(apsm::FinalState))->size()=1 then
     '1'
    else
     '2'
    endif
   endif
  }
}
```

Figures 1-5 display the visualized PIM-to-APSM transformations using the visualization technique of [].

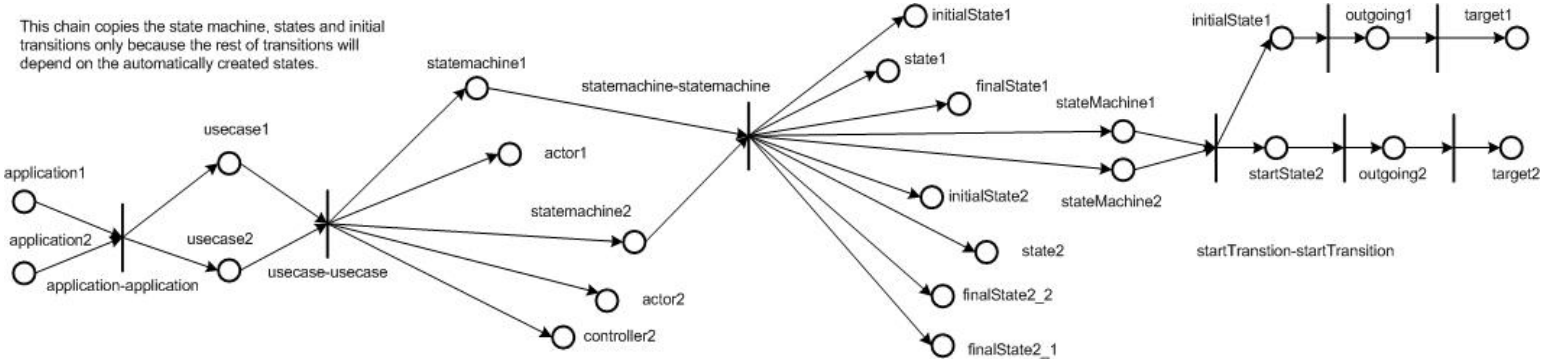

Figure 1 – A chain of relations mapping state machines, states and transitions

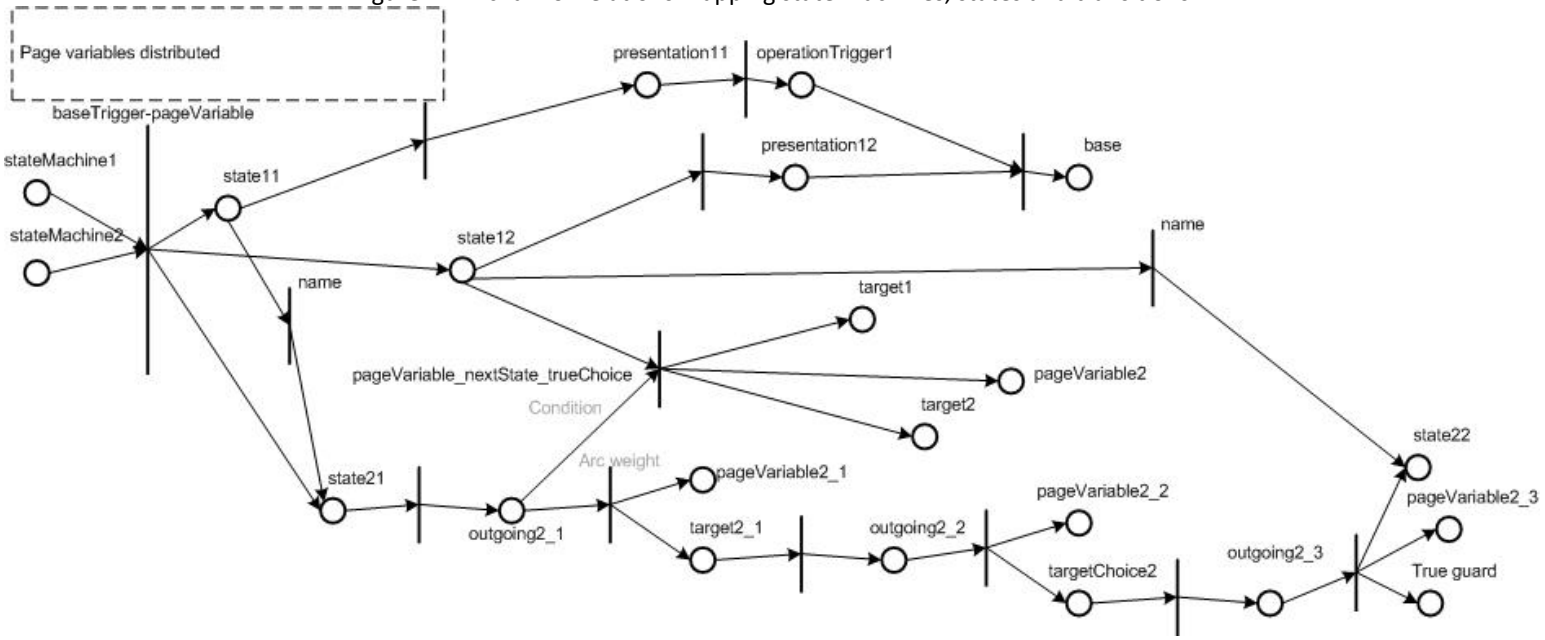

Figure 2 – A chain of relations to distribute page variables amongst different states of the application

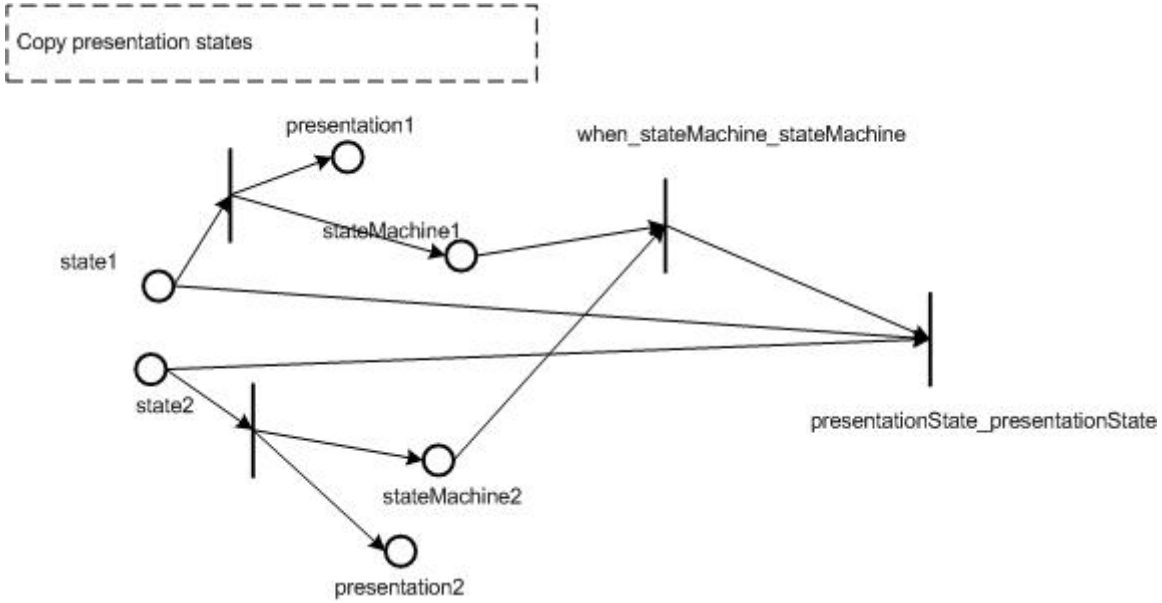

Figure 3 – The relation presentationState_presentationState

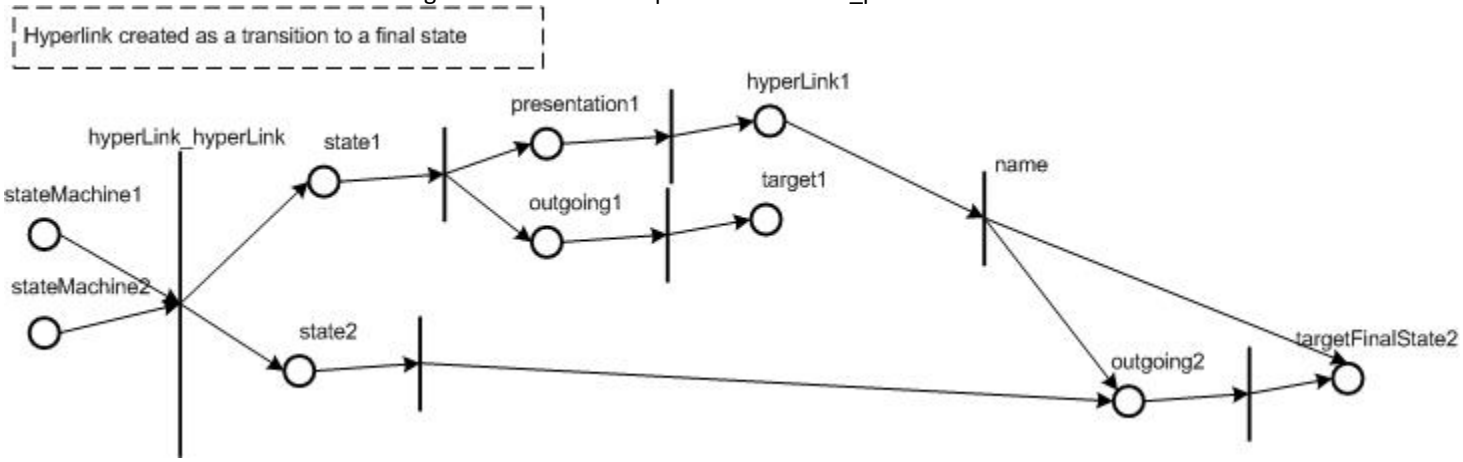

Figure 4 – The relation hyperlink_hyperLink

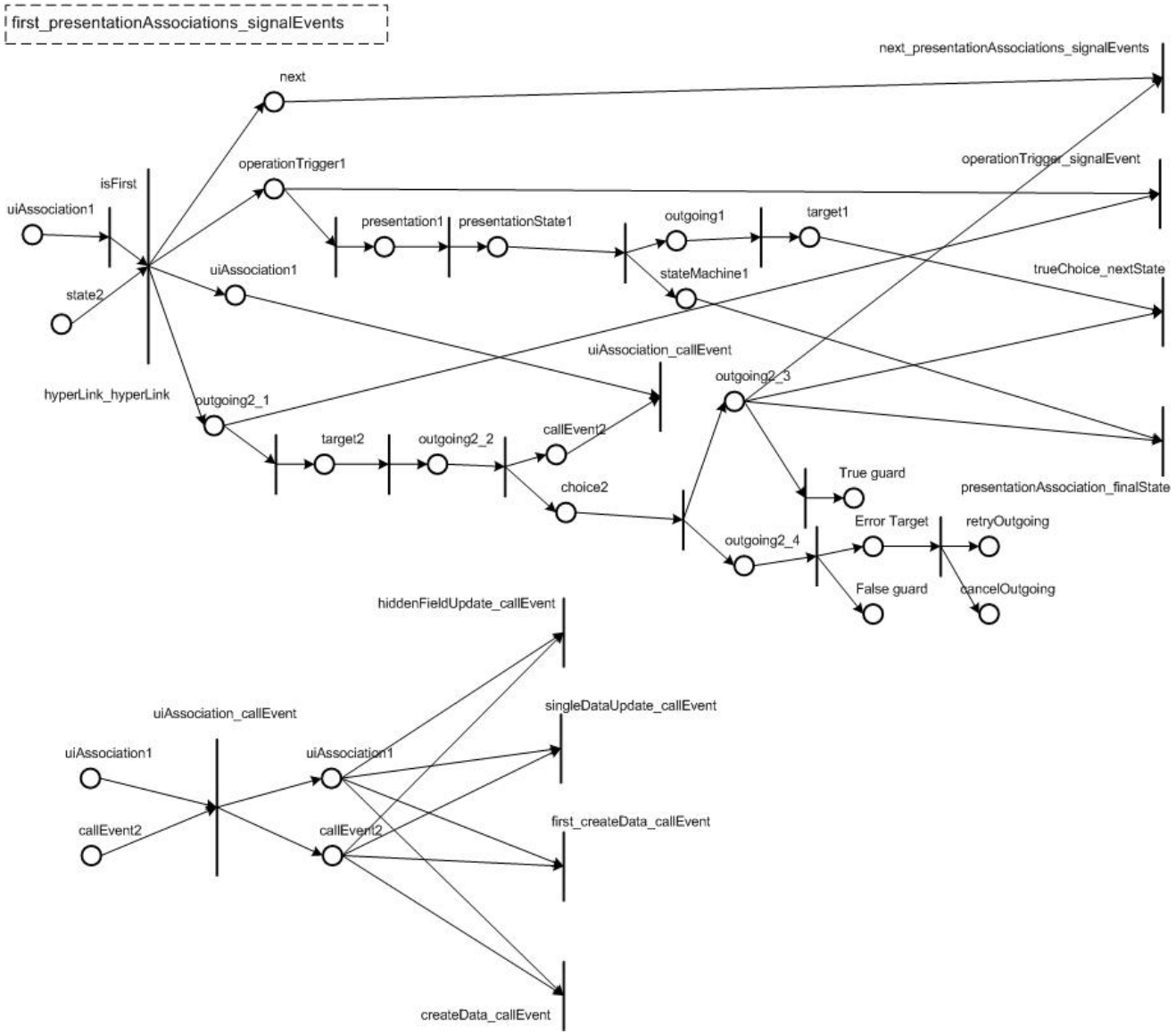

Figure 5 – The relation hyperlink_hyperLink

3 Listing of the APSM-to-AndroMDA Relations

```
transformation apsm_andromda(aPSM:apsm, androMDA:andromda) {
  key apsm::Application{name};
  key andromda::NamedElement{name, owner};
  key andromda::Model{name};
  key andromda::Transition{source, target};
  key andromda::UseCase{name};
  key andromda::Operation{name, owner};
  key andromda::Parameter{name, owner};
  key andromda::SignalEvent{name, owner};

  top relation application_model {
   applicationName:String;

   checkonly domain aPSM application1:apsm::Application {
    name=applicationName,
    useCases=useCase1:apsm::UseCase{}
   };

   checkonly domain aPSM dataComposite1:apsm::DataComposite {
   };

   enforce domain androMDA model2:andromda::Model {
    name='Data',
    ownedMember=package2:andromda::Package {
     owner=model2,
     name=applicationName,
     ownedMember=package2_1:andromda::Package {
      owner=package2,
      name='org.modewis.'+applicationName,
      ownedMember=package2_1_1:andromda::Package {
       owner=package2 1,
       name='services'
      },
      ownedMember=package2_1_2:andromda::Package {
       owner=package2_1,
       name='vo'
      },
      ownedMember=package2_1_3:andromda::Package {
       owner=package2 1,
       name='domain'
      },
      ownedMember=package2_1_4:andromda::Package {
       owner=package2 1,
       name='web'
      }
     }
    }
   };
   when {
    application1.name->notEmpty();
   }
   where {
    dataComposite_valueObject(application1, dataComposite1, model2);
    usecase_webPackage(useCase1, model2);
   }
  }

  relation usecase_webPackage {
   usecaseName:String;
```

```
  controllerName:String;
  machineName:String;
  applicationName:String;

  checkonly domain aPSM useCase1:apsm::UseCase {
   name=usecaseName,
   application=application1:apsm::Application {
    name=applicationName
   },
   controller=controller1:apsm::Controller {
    name=controllerName
   },
   stateMachine=stateMachine1:apsm::StateMachine {
    name=machineName
   }
  };

  enforce domain androMDA model2:andromda::Model {
   name='Data',
   ownedMember=package2:andromda::Package {
    owner=model2,
    name=applicationName,
    ownedMember=package2_1:andromda::Package {
     owner=package2,
     name='org.modewis.'+applicationName,
     ownedMember=package2_1_1:andromda::Package {
      name='web',
      owner=package2_1,
      name=usecaseName,
      ownedMember=useCase2:andromda::UseCase {
       name=usecaseName,
       stereotypes=addStereotype(useCase2, andromda::Stereotype::FrontEndUseCase),
       behaviour=stateMachine2:andromda::StateMachine {
        owner=package2_1_1,
        name=machineName,
        controller=controller2:andromda::Class {
         owner=stateMachine2,
         name=controllerName
        }
       }
      }
     }
    }
   }
  };

  where {
   stateMachine_stateMachine(stateMachine1, package2_1_1, stateMachine2);
  }
}

relation stateMachine_stateMachine {
  machineName:String;
  stateName:String;

  checkonly domain aPSM stateMachine1:apsm::StateMachine {
   name=machineName,
   states=state1:apsm::State {
    name=stateName
   }
  };

  primitive domain ownerPackage:andromda::Package;
```

```
 enforce domain androMDA stateMachine2:andromda::StateMachine {
  name=machineName,
  owner=ownerPackage,
  states=state2:andromda::State {
   owner=stateMachine2,
   name=stateName
  }
 };

 where {
  finalState_finalState(stateMachine1, ownerPackage, stateMachine2);
  starttransition_starttransition(stateMachine1, ownerPackage, stateMachine2) or
   transition transition(state1, ownerPackage, state2) or
    transitionSignal transitionSignal(state1, ownerPackage, state2) or
     transitionCall transitionCall(state1, ownerPackage, state2);
 }
}

relation finalState finalState {
 machineName:String;

 checkonly domain aPSM stateMachine1:apsm::StateMachine {
  name=machineName
 };

 primitive domain ownerPackage:andromda::Package;

 enforce domain androMDA stateMachine2:andromda::StateMachine {
  name=machineName,
  owner=ownerPackage,
  states=finalState:andromda::State {
   isFinal=true,
   name='finalState',
   owner=stateMachine2
  }
 };
}

relation starttransition_starttransition {
 targetName:String;
 machineName:String;

 checkonly domain aPSM stateMachine1:apsm::StateMachine {
  name=machineName,
  states=startState1:apsm::StartState {
   outgoings=outgoing1:apsm::Transition {
    target=target1:apsm::State {
     name=targetName
    }
   }
  }
 };

 primitive domain ownerPackage:andromda::Package;

 enforce domain androMDA stateMachine2:andromda::StateMachine {
  name=machineName,
  owner=ownerPackage,
  states=source2:andromda::State {
   name='startState',
   isInitial=true,
   owner=stateMachine2,
```

```
    outgoings=outgoing2:andromda::Transition {
     owner=stateMachine2,
     source=source2,
     target=target2:andromda::State {
      owner=stateMachine2,
      name=targetName,
      incomings=incoming2:andromda::Transition {
       owner=stateMachine2,
       source=source2,
       target=target2
      }
     }
    }
   }
  };
}

relation transition_transition {
 targetName:String;
 machineName:String;
 sourceName:String;

 checkonly domain aPSM source1:apsm::State {
  name=sourceName,
  stateMachine=stateMachine1:apsm::StateMachine {
   name=machineName
  },
  outgoings=outgoing1:apsm::Transition {
   target=target1:apsm::State {
    name=targetName
   }
  }
 };

 primitive domain ownerPackage:andromda::Package;

 enforce domain androMDA source2:andromda::State {
  name=sourceName,
  owner=stateMachine2:andromda::StateMachine {
   name=machineName,
   owner=ownerPackage
  },
  outgoings=outgoing2:andromda::Transition {
   owner=stateMachine2,
   source=source2,
   target=target2:andromda::State {
    owner=stateMachine2,
    name=targetName,
    incomings=incoming2:andromda::Transition {
     owner=stateMachine2,
     source=source2,
     target=target2
    }
   }
  }
 };
}

relation transitionSignal_transitionSignal {
 targetName:String;
 machineName:String;
 sourceName:String;
 signalName:String;
```

```
  checkonly domain aPSM source1:apsm::State {
   name=sourceName,
   stateMachine=stateMachine1:apsm::StateMachine {
    name=machineName
   },
   outgoings=outgoing1:apsm::Transition {
    target=target1:apsm::State {
     name=targetName
    },
    signalEvent=signalEvent1:apsm::SignalEvent {
     name=signalName
    }
   }
  };

  primitive domain ownerPackage:andromda::Package;

  enforce domain androMDA source2:andromda::State {
   name=sourceName,
   owner=stateMachine2:andromda::StateMachine {
    name=machineName,
    owner=ownerPackage
   },
   outgoings=outgoing2:andromda::Transition {
    owner=stateMachine2,
    source=source2,
    target=target2:andromda::State {
     owner=stateMachine2,
     name=targetName,
     incomings=incoming2:andromda::Transition {
      owner=stateMachine2,
      source=source2,
      target=target2
     }
    },
    signalEvent=signalEvent2:andromda::SignalEvent {
     owner=outgoing2,
     name=signalName
    }
   }
  };

  where {
   signalEventParam signalEvent(signalEvent1, outgoing2, signalEvent2) or signalE-
vent_signalEvent(signalEvent1, signalEvent2);
  }
 }

 relation transitionCall transitionCall {
  targetName:String;
  machineName:String;
  sourceName:String;
  eventName:String;
  operationName:String;
  ucName:String;

  checkonly domain aPSM source1:apsm::State {
   name=sourceName,
   stateMachine=stateMachine1:apsm::StateMachine {
    name=machineName,
    useCase=useCase2:apsm::UseCase {
     name=ucName
```

```
    }
   },
   outgoings=outgoing1:apsm::Transition {
    target=target1:apsm::State {
     name=targetName
    },
    callEvents=callEvent1:apsm::CallEvent {
     name=eventName,
     operation=operation1:apsm::Operation {
      name=operationName
     }
    }
   }
  };

  primitive domain ownerPackage:andromda::Package;

  enforce domain androMDA source2:andromda::State {
   name=sourceName,
   owner=stateMachine2:andromda::StateMachine {
    name=machineName,
    owner=ownerPackage,
    controller=controller2:andromda::Class {
     name=ucName+'Controller',
     owner=stateMachine2,
     operation=operation2:andromda::Operation {
      owner=controller2,
      name=operationName
     }
    }
   },
   outgoings=outgoing2:andromda::Transition {
    owner=stateMachine2,
    source=source2,
    target=target2:andromda::State {
     owner=stateMachine2,
     name=targetName,
     incomings=incoming2:andromda::Transition {
      owner=stateMachine2,
      source=source2,
      target=target2
     }
    },
    callEvent=callEvent2:andromda::CallEvent {
     owner=outgoing2,
     name=eventName,
     operation=operation2
    }
   }
  };

  where {
   operationParam_operationParam(operation1,controller2,operation2) or
    operation_operation(operation1, operation2);
  }
}

relation operationParam_operationParam {
 operationName:String;
 paramName:String;

 checkonly domain aPSM operation1:apsm::Operation {
  name=operationName,
```

```
   parameters=parameter1:apsm::Parameter {
    name=paramName
   }
  };

  primitive domain class2:andromda::Class;

  enforce domain androMDA operation2:andromda::Operation {
   name=operationName,
   owner=class2,
   parameter=parameter2:andromda::Parameter {
    name=paramName,
    owner=operation2
   }
  };
}

relation operation_operation {
 checkonly domain aPSM operation1:apsm::Operation {
 };

 enforce domain androMDA operation2:andromda::Operation {
 };
}

relation signalEventParam_signalEvent {
 paramName:String;
 signalName:String;

 checkonly domain aPSM signalEvent1:apsm::SignalEvent {
  name=signalName,
  parameters=parameter1:apsm::Parameter {
   name=paramName,
   type=type1:apsm::Type {
   }
  }
 };

 primitive domain transition2:andromda::Transition;

 enforce domain androMDA signalEvent2:andromda::SignalEvent {
  name=signalName,
  owner=transition2,
  parameter=parameter2:andromda::Parameter {
   owner=signalEvent2,
   name=paramName
  }
 };
}

relation signalEvent_signalEvent {
 checkonly domain aPSM signalEvent1:apsm::SignalEvent {
 };

 enforce domain androMDA signalEvent2:andromda::SignalEvent {
 };
}

relation dataComposite_valueObject {
 dataName:String;
 compositeName:String;
 applicationName:String;
 serviceName:String;
```

```
operationName:String;

checkonly domain aPSM application1:apsm::Application {
 name=applicationName
};

checkonly domain aPSM dataComposite1:apsm::DataComposite {
 name=compositeName,
 service=service1:apsm::Service{
  name=serviceName,
  operations=serviceOperation1:apsm::Operation {
   name=operationName
  }
 },
 dataEntities=entity1:apsm::DataEntity {
  name=dataName
 }
};

enforce domain androMDA model2:andromda::Model {
 name='Data',
 ownedMember=package2:andromda::Package {
  owner=model2,
  name=applicationName,
  ownedMember=package2 1:andromda::Package {
   owner=package2,
   name='org.modewis.'+applicationName,
   ownedMember=voPackage2:andromda::Package {
    name='vo',
    owner=package2_1,
    ownedMember=valueObject2_1:andromda::Class {
     name=compositeName.firstToUpper()+'VO',
     stereotypes=addStereotype(valueObject2 1, andromda::Stereotype::ValueObject),
     attributes=attribute2:andromda::Attribute {
      owner=valueObject2_1,
      name='id',
      type=andromda::DataTypes::Long
     },
     owner=voPackage2
    }
   },
   ownedMember=package2_1_1:andromda::Package {
    name='services',
    owner=package2_1,
    ownedMember=serviceClass2:andromda::Class {
     owner=package2_1_1,
     name=serviceName,
     stereotypes=addStereotype(serviceClass2, andromda::Stereotype::Service),
     operation=operation2:andromda::Operation {
      name=operationName,
      owner=serviceClass2,
      parameter=parameter2:andromda::Parameter {
       owner=operation2,
       name=compositeName.toLower()+'VO',
       type=type2:andromda::Type {
        auxiliaryType=valueObject2 1
       }
      }
     }
    }
   }
  }
 }
```

```
  };

  where {
  }
}

top relation dependencyControllerService {
 machineName:String;
 usecaseName:String;
 applicationName:String;
 serviceName:String;
 controllerName:String;

 checkonly domain aPSM application1:apsm::Application {
  name=applicationName,
  useCases=useCase1:apsm::UseCase{
   stateMachine=stateMachine1:apsm::StateMachine {
    name=machineName
   },
   name=usecaseName,
   controller=controller1:apsm::Controller {
    name=controllerName,
    operations=operation1:apsm::Operation {
     callee=callee1:apsm::Operation {
      class=class1:apsm::Service {
       name=serviceName
      }
     }
    }
   }
  }
 };

 enforce domain androMDA model2:andromda::Model {
  name='Data',
  ownedMember=package2:andromda::Package {
   owner=model2,
   name=applicationName,
   ownedMember=package2 1:andromda::Package {
    owner=package2,
    name='org.modewis.'+applicationName,
    ownedMember=package2_1_1:andromda::Package {
     name='services',
     owner=package2_1,
     ownedMember=serviceClass2:andromda::Class {
      owner=package2_1_1,
      name=serviceName
     }
    },
    ownedMember=package2 1 4:andromda::Package {
     owner=package2_1,
     name='web',
     ownedMember=useCase2:andromda::UseCase {
      name=usecaseName,
      stereotypes=addStereotype(useCase2, andromda::Stereotype::FrontEndUseCase),
      behaviour=stateMachine2:andromda::StateMachine {
       name=machineName,
       owner=package2_1_4,
       controller=controller2:andromda::Class {
        owner=stateMachine2,
        name=controllerName,
        clientDependencies=dependency2:andromda::Dependency {
         client=controller2,
```

```
            supplier=serviceClass2
           }
          }
         }
        }
       }
      }
     }
   };
}

top relation presentation_frontEndView {
 machineName:String;
 stateName:String;
 usecaseName:String;
 applicationName:String;

 checkonly domain aPSM application1:apsm::Application {
  name=applicationName,
  useCases=useCase1:apsm::UseCase{
   name=usecaseName,
   stateMachine=stateMachine1:apsm::StateMachine {
    name=machineName,
    states=state1:apsm::State {
     name=stateName,
     presentation=presentation1:apsm::Presentation {
     }
    }
   }
  }
 };

 enforce domain androMDA model2:andromda::Model {
  name='Data',
  ownedMember=package2:andromda::Package {
   owner=model2,
   name=applicationName,
   ownedMember=package2_1:andromda::Package {
    owner=package2,
    name='org.modewis.'+applicationName,
    ownedMember=package2_1_4:andromda::Package {
     owner=package2_1,
     name='web',
     ownedMember=useCase2:andromda::UseCase {
      name=usecaseName,
      stereotypes=addStereotype(useCase2, andromda::Stereotype::FrontEndUseCase),
      behaviour=stateMachine2:andromda::StateMachine {
       name=machineName,
       owner=package2_1_4,
       states=state2:andromda::State {
        owner=stateMachine2,
        name=stateName,
        stereotypes=addStereotype(state2, andromda::Stereotype::FrontEndView)
       }
      }
     }
    }
   }
  }
 };
}
```

```
  query addStereotype(element:andromda::NamedElement, stereo-
type:andromda::Stereotype):OrderedSet(andromda::Stereotype) {
    element.stereotypes->append(stereotype)
  }

  query addOutgoing(state:andromda::State, transi-
tion:andromda::Transition):OrderedSet(andromda::Transition) {
    state.outgoings->append(transition)
  }

  query addIncoming(state:andromda::State, transi-
tion:andromda::Transition):OrderedSet(andromda::Transition) {
    state.incomings->append(transition)
  }
}
```

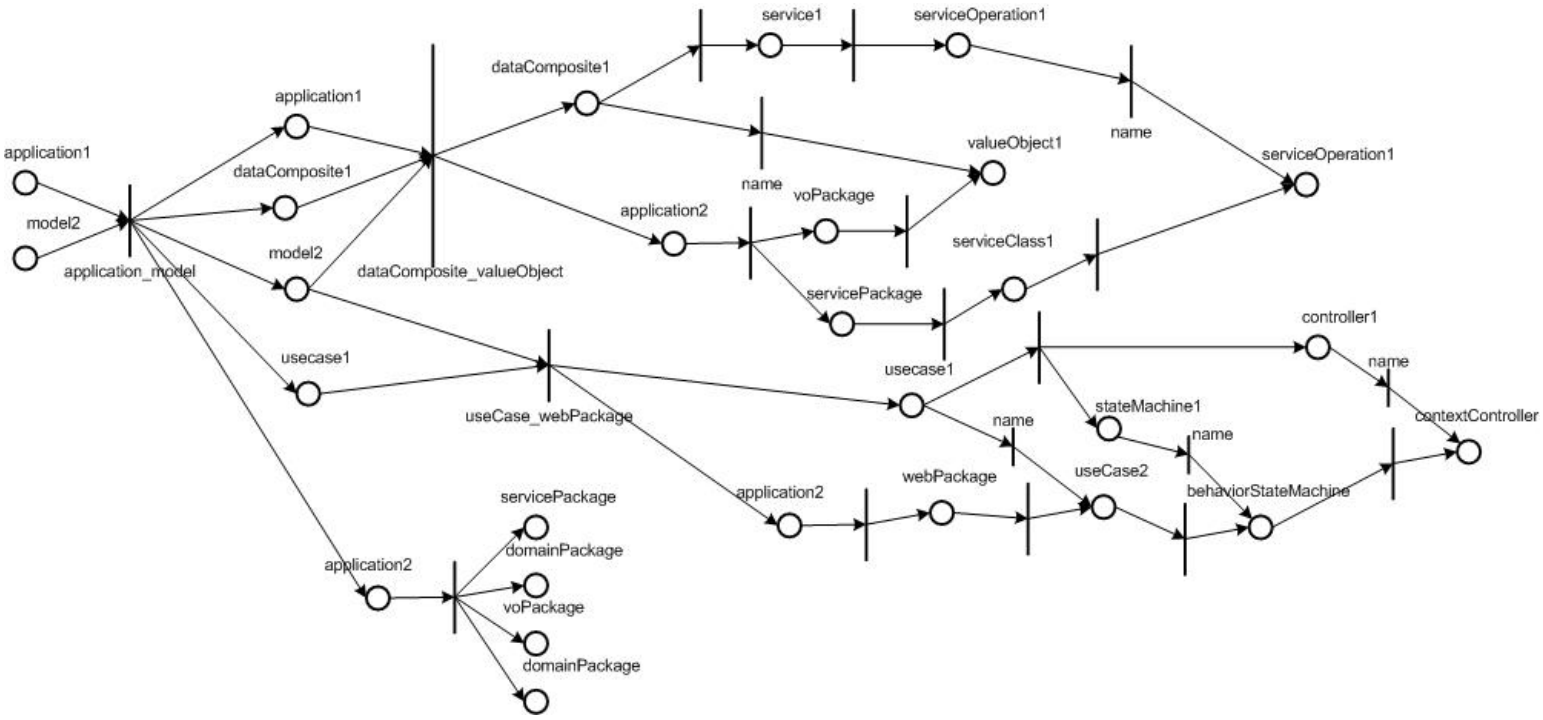

Figure 6 – A Chain of relations creating the application, use cases, state machines and the data

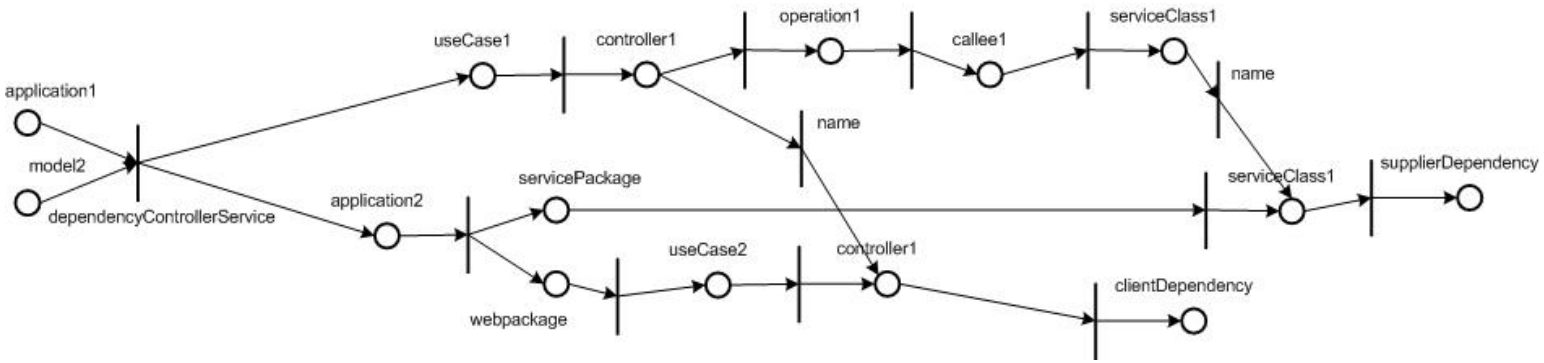

Figure 7 – Creating the dependencies between elements

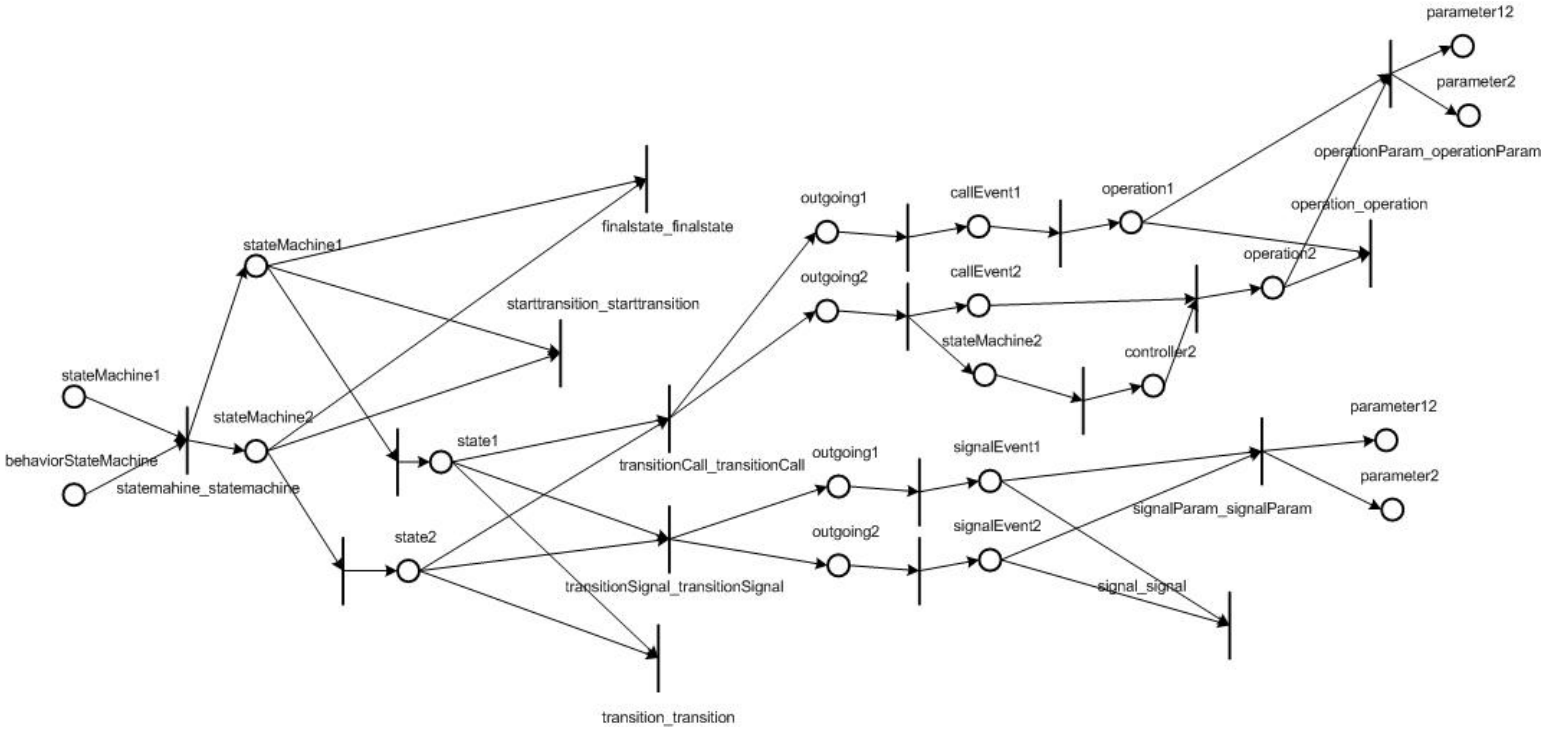

Figure 8 – A Chain of relations creating the details of state machines

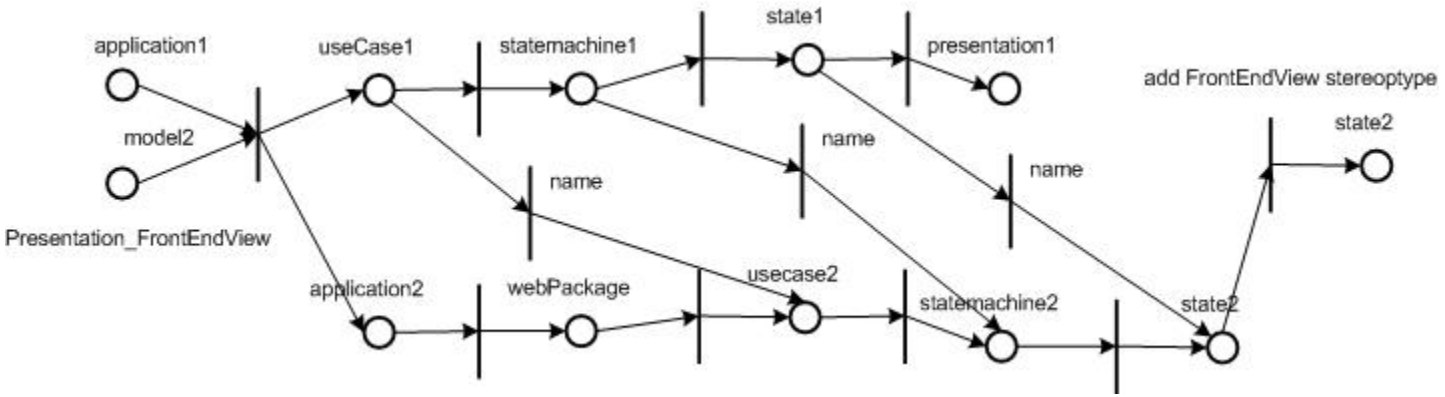

Figure 9 – Applying stereotypes

4 Listing of the APSM-to-WebRatio Relations

```
transformation apsm_webratio(aPSM:apsm, webRatio:webratio) {
  key webratio::WebProject{name};
  key webratio::WebModel{name};
  key webratio::SiteView{name};
  key webratio::Unit{name, siteView};
  key webratio::Entity{name};
  key webratio::Field{name, unit};
  key webratio::Link{name};
  key webratio::Attribute{name, entity};
  key webratio::Parameter{name, link};

  top relation actor_siteView {
   projectName:String;
   viewName:String;
   machineName:String;

   checkonly domain aPSM application1:apsm::Application {
    name=projectName,
    useCases=useCase1:apsm::UseCase {
     actors=actor1:apsm::Actor {
      name=viewName
     },
     stateMachine=stateMachine1:apsm::StateMachine {
      name=machineName,
      states=startState1:apsm::StartState {
       outgoings=outgoing1:apsm::Transition {
        target=state1:apsm::State {
        }
       }
      }
     }
    }
   };

   enforce domain webRatio    webProject2:webratio::WebProject {
    name=projectName,
    webModel=webModel2:webratio::WebModel {
     name=projectName,
     siteViews=siteView2:webratio::SiteView {
      name=viewName,
      homePage=homePage2:webratio::Page {
       name=viewName+'Home',
       siteView=siteView2,
       links=link2:webratio::Link {
        name=machineName,
        to=unit2:webratio::Unit {
         siteView=siteView2
        }
       }
      },
      pages=addPage(homePage2, siteView2)
     }
    }
   };

   where {
    presentationState_page(state1, unit2);
   }
  }
```

```
relation presentationState page {
 presentationName:String;
 unitName:String;

 checkonly domain aPSM state1:apsm::State {
  name=presentationName,
  presentation=presentation1:apsm::Presentation {
   name=unitName
  },
  outgoings=transition1:apsm::Transition {
  }
 };

 enforce domain webRatio page2:webratio::Page {
  name=presentationName
 };

 where {
  signalEvent entryUnit(transition1, page2);
 }
}

relation signalEvent_entryUnit {
 checkonly domain aPSM transition1:apsm::Transition {
  signalEvent=signalEvent1:apsm::SignalEvent {
  }
 };

 checkonly domain webRatio page2:webratio::Page {
 };

 where {
  signalEvent entryUnitcreateUnit(transition1, page2);
 }
}

relation signalEvent_entryUnitcreateUnit {
 fieldName:String;
 dataName:String;
 attributeName:String;
 signalName:String;
 viewName:String;
 eventName:String;
 errorName:String;
 homeName:String;
 pageName:String;

 checkonly domain aPSM transition1:apsm::Transition {
  source=source1:apsm::State {
   name=pageName
  },
  stateMachine=stateMachine1:apsm::StateMachine {
   useCase=useCase1:apsm::UseCase {
    actors=actor1:apsm::Actor {
     name=viewName
    }
   }
  },
  signalEvent=signalEvent1:apsm::SignalEvent {
   name=signalName,
   parameters=parameter1:apsm::Parameter {
    name=fieldName
```

```
    }
   },
   target=target1:apsm::State {
    outgoings=outgoing1:apsm::Transition {
     target=choice1:apsm::Choice {
      outgoings=falseOutgoing:apsm::Transition {
       guard=guard1:apsm::Guard {
        name='false'
       },
       target=errorState1:apsm::State {
        name=errorName,
        outgoings=retryOutgoing:apsm::Transition {
         signalEvent=retryEvent:apsm::SignalEvent {
          name='retry'
         },
         target=target1 1:apsm::State {
          name=homeName
         }
        },
        outgoings=cancelOutgoing:apsm::Transition {
         signalEvent=cancelEvent:apsm::SignalEvent {
          name='cancel'
         }
        }
       }
      }
     },
     callEvents=callEvent1:apsm::CallEvent {
      name=eventName,
      operation=operation1:apsm::Operation {
       callee=callee1:apsm::Operation {
        crudNature=apsm::QueryType::create,
        class=service1:apsm::Service {
         dataComposite=dataComposite1:apsm::DataComposite {
          name=dataName,
          dataEntities=dataEntity1:apsm::DataEntity {
           name=dataName,
           attributes=attribute1:apsm::Attribute {
            name=attributeName
           }
          }
         }
        }
       }
      }
     }
    }
   }
  };

  enforce domain webRatio page2:webratio::Page {
   name=pageName,
   siteView=siteView2:webratio::SiteView {
    name=viewName
   },
   contentUnits=entryUnit2:webratio::EntryUnit {
    siteView=siteView2,
    name='Enter '+dataName+' Info',
    fields=field2:webratio::Field {
     unit=entryUnit2,
     name=fieldName
    },
    links=link2:webratio::Link {
```

```
      name=signalName,
      to=createUnit2:webratio::CreateUnit {
       siteView=siteView2,
       name=eventName,
       entity=entity2:webratio::Entity {
        name=dataName,
        attributes=attribute2:webratio::Attribute {
         entity=entity2,
         name=attributeName
        },
        attributes=id2:webratio::Attribute {
         entity=entity2,
         name='oid'
        }
       },
       fields=field2 1:webratio::Field {
        unit=createUnit2,
        name=attributeName
       },
       fields=field2 2:webratio::Field {
        unit=createUnit2,
        name='oid'
       },
       links=koLink2:webratio::KOLink {
        name=eventName+'Failure',
        to=errorPage:webratio::Page {
         siteView=siteView2,
         name=errorName,
         links=link2_1:webratio::Link {
          name='retry',
          to=page2_1:webratio::Page {
           name=homeName,
           siteView=siteView2:webratio::SiteView {
            name=viewName,
            pages=addPage(page2_1, siteView2)
           }
          }
         }
        }
       }
      },
      linkParameters=parameter2:webratio::Parameter {
       link=link2,
       source=field2,
       target=findTargetFor(fieldName.toLower(), createUnit2.fields),
       name=fieldName+'_'+findTargetFor(fieldName.toLower(), createUnit2.fields).name
      }
     }
    }
   };

   when {

   }

   where {
    callEventcallEvent_UnitcreateUnit(viewName, homeName, callEvent1, createUnit2);
   }
}

relation callEventcallEvent_UnitcreateUnit {
 dataName:String;
 attributeName:String;
```

```
eventName:String;

primitive domain viewName:String;
primitive domain homeName:String;

checkonly domain aPSM callEvent1:apsm::CallEvent {
 owner=owner1:apsm::Transition {
  stateMachine=statemachine1:apsm::StateMachine {
   useCase=useCase1:apsm::UseCase {
    actors=actor1:apsm::Actor {
     name=viewName
    }
   }
  },
  target=choice1:apsm::Choice {
   outgoings=outgoing1:apsm::Transition {
    guard=guard1:apsm::Guard {
     name='true'
    },
    target=state1:apsm::State {
     outgoings=outgoing1_1:apsm::Transition {
      callEvents=callEvent1 1:apsm::CallEvent {
       name=eventName,
       operation=operation1:apsm::Operation {
        callee=callee1:apsm::Operation {
         crudNature=apsm::QueryType::create,
         class=service1:apsm::Service {
          dataComposite=dataComposite1:apsm::DataComposite {
           name=dataName,
           dataEntities=dataEntity1:apsm::DataEntity {
            name=dataName,
            attributes=attribute1:apsm::Attribute {
             name=attributeName
            }
           }
          }
         }
        }
       }
      }
     }
    }
   }
  }
 }
};

enforce domain webRatio unit2:webratio::OperationUnit {
 fields=field2:webratio::Field {
  unit=unit2,
  name=attributeName
 },
 links=link2:webratio::OKLink {
  name=unit2.name+'Success',
  to=createUnit2:webratio::CreateUnit {
   name=eventName,
   siteView=unit2.siteView,
   entity=entity2:webratio::Entity {
    name=dataName,
    attributes=id2:webratio::Attribute {
     entity=entity2,
     name='oid'
    }
```

```
      },
      fields=field2_2:webratio::Field {
       unit=createUnit2,
       name='oid'
      },
      links=koLink2:webratio::KOLink {
       name=eventName+'Failure',
       to=errorPage:webratio::Page {
        siteView=unit2.siteView,
        name='error',
        links=link2_1:webratio::Link {
         name='retry',
         to=page2_1:webratio::Page {
          name=homeName,
          siteView=siteView2:webratio::SiteView {
           name=viewName
          }
         }
        }
       }
      }
     },
     linkParameters=parameter2:webratio::Parameter {
      link=link2,
      source=findIDSourceField(unit2),
      target=field2_2,
      name='id id'
     }
    }
   };

   where {
    if createUnit2->notEmpty() then
     callEventfinalState UnitcreateUnit(viewName, callEvent1 1, createUnit2)
    else
     true
    endif;
   }
}

relation callEventfinalState_UnitcreateUnit {
 eventName:String;

 primitive domain viewName:String;

 checkonly domain aPSM callEvent1:apsm::CallEvent {
  name=eventName,
  owner=owner1:apsm::Transition {
   stateMachine=statemachine1:apsm::StateMachine {
    useCase=useCase1:apsm::UseCase {
     actors=actor1:apsm::Actor {
      name=viewName
     }
    }
   },
   target=choice1:apsm::Choice {
    outgoings=outgoing1:apsm::Transition {
     guard=guard1:apsm::Guard {
      name='true'
     },
     target=finalState1:apsm::FinalState {
     }
    }
```

```
    }
   }
  };

  enforce domain webRatio unit2:webratio::OperationUnit {
   name=eventName,
   siteView=siteView2:webratio::SiteView {
    name=viewName
   },
   links=link2:webratio::OKLink {
    name=eventName+'Success',
    to=homePage2:webratio::Page {
     name=viewName+'Home',
     siteView=siteView2
    }
   }
  };
}

relation event link {
 fieldName:String;
 dataName:String;
 attributeName:String;
 signalName:String;
 viewName:String;

 checkonly domain aPSM transition1:apsm::Transition {
  stateMachine=stateMachine1:apsm::StateMachine {
   useCase=useCase1:apsm::UseCase {
    actors=actor1:apsm::Actor {
     name=viewName
    }
   }
  },
  signalEvent=signalEvent1:apsm::SignalEvent {
   name=signalName,
   parameters=parameter1:apsm::Parameter {
    name=fieldName
   }
  },
  target=target1:apsm::State {
   outgoings=outgoing1:apsm::Transition {
    callEvents=callEvent1:apsm::CallEvent {
     operation=operation1:apsm::Operation {
      callee=callee1:apsm::Operation {
       crudNature=apsm::QueryType::create,
       class=service1:apsm::Service {
        dataComposite=dataComposite1:apsm::DataComposite {
         name=dataName,
         dataEntities=dataEntity1:apsm::DataEntity {
          name=dataName,
          attributes=attribute1:apsm::Attribute {
           name=attributeName
          }
         }
        }
       }
      }
     }
    }
   }
  };
```

```
 enforce domain webRatio    page2:webratio::Page {
  siteView=siteView2:webratio::SiteView {
   name=viewName
  },
  contentUnits=entryUnit2:webratio::EntryUnit {
   siteView=siteView2,
   name='Enter '+dataName+' Info',
   fields=field2:webratio::Field {
    unit=entryUnit2,
    name=fieldName
   },
   links=link2:webratio::Link {
    name=signalName,
    to=createUnit2:webratio::CreateUnit {
     siteView=siteView2,
     name='Create '+dataName,
     entity=entity2:webratio::Entity {
      name=dataName,
      attributes=attribute2:webratio::Attribute {
       entity=entity2,
       name=attributeName
      }
     },
     fields=field2 1:webratio::Field {
      unit=createUnit2,
      name=attributeName
     }
    },
    linkParameters=parameter2:webratio::Parameter {
     link=link2,
     source=field2,
     target=findTargetFor(fieldName.toLower(), createUnit2.fields),
     name=source.name+' '+target.name
    }
   }
  }
 };
}

relation parameter_contentField {
 paramName:String;
 paramType:apsm::DataType;

 checkonly domain aPSM behavior1:apsm::Behavior {
  parameters=parameter1:apsm::Parameter {
   name=paramName,
   type=type1:apsm::Type {
    primitiveType=paramType
   }
  }
 };

 enforce domain webRatio contentUnit2:webratio::ContentUnit {
  fields=field2:webratio::Field {
   name=paramName,
   type=getWebRatioType(paramType),
   unit=contentUnit2
  }
 };
}

query getWebRatioType(apsmType:apsm::DataType):webratio::DataType {
```

```
   if (apsmType=apsm::DataType::string) then webratio::DataType::string
    else webratio::DataType::string
   endif
  }

  query findTargetFor(fieldName:String,
fields:OrderedSet(webratio::Field)):webratio::Field {
   fields->select(f|f.name=fieldName)->first()
  }

  query findIDSourceField(unit:webratio::Unit):webratio::Field {
   unit.fields->select(f|f.name='oid')->first()
  }

  query addPage(page:webratio::Page, site-
View:webratio::SiteView):OrderedSet(webratio::Page) {
   siteView.pages->append(page)
  }
}
```

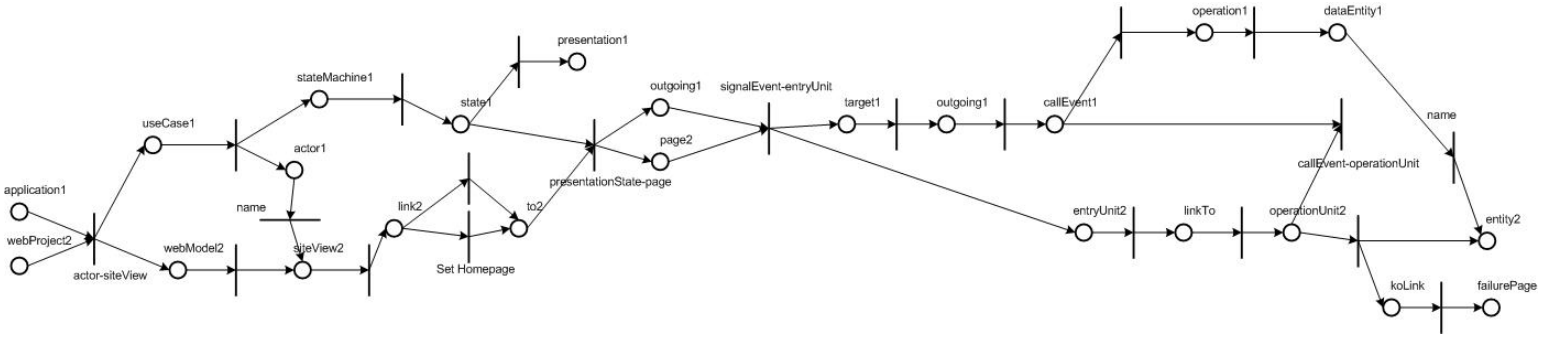

Figure 10 – Mapping the APSM to WebRatio

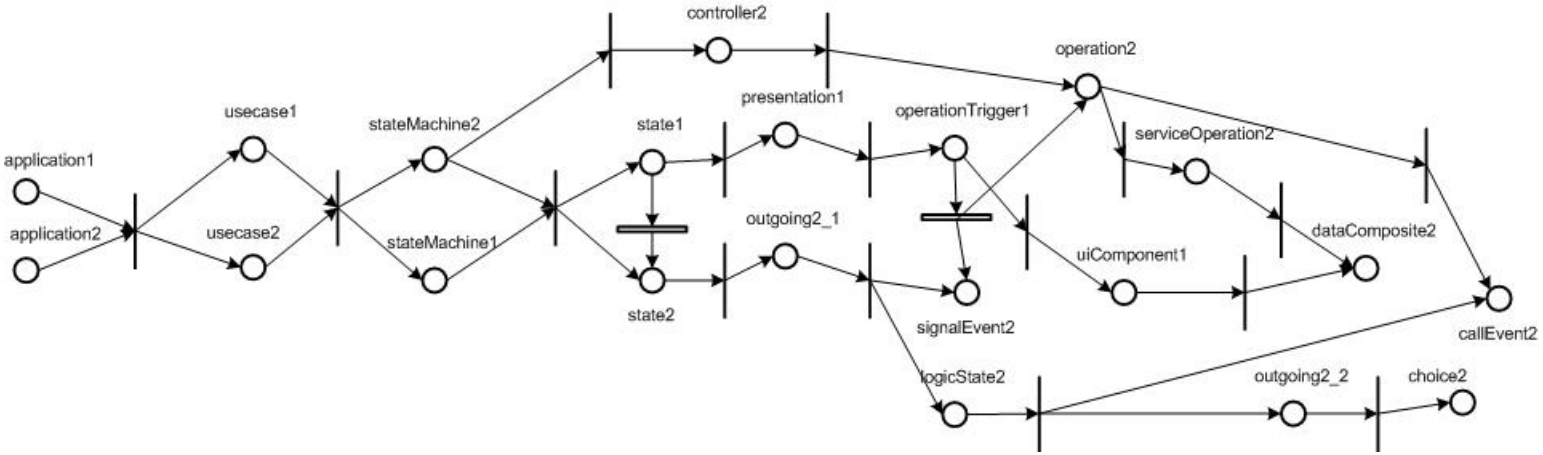

Figure 11 – PIM-to-APSM Transformations summarized in one shot

5 Listing of the APSM-to-GWT Mappings

```
transformation apsm_gwt(aPSM:apsm, gwt:gwt) {
  key gwt::Widget{name, container};
  key gwt::Operation{name, class};
  key gwt::Package{name};
  key gwt::Application{name};
  key gwt::Class{name, owningPackage};

  top relation application application {
   applicationName:String;

   checkonly domain aPSM application1:apsm::Application {
    name=applicationName
   };

   enforce domain gwt application2:gwt::Application {
    name=applicationName,

    mainEntryPoint=entryPoint2:gwt::MainEntryPoint {
     name=applicationName+'EntryPoint'
    },
    client=client2:gwt::Package {
     name='client',
     classes=classes->append(entryPoint2)
    },
    server=server2:gwt::Package {
     name='server'
    },
    shared=shared2:gwt::Package {
     name='shared'
    }
   };

   where {
   }
  }

  top relation operationTrigger_submitButton {
   panelName:String;

   checkonly domain aPSM operationTrigger1:apsm::OperationTrigger {
    name=panelName
   };

   enforce domain gwt panel2:gwt::Panel {
    name=panelName,
    widgets=button2:gwt::Button {
     name=panelName,
     container=panel2,
     eventHandlers=eventHandler2:gwt::EventHandler {
      name=panelName+'Hanlder',
      owningPackage=package2:gwt::Package {
       name='client',
       classes=classes->append(eventHandler2)
      },
      operations=operation1:gwt::Operation {
       name='onClick',
       class=eventHandler2
      }
```

```
    }
   }
  };

  when {
   uiComposite_panel(operationTrigger1, panel2);
  }
}

top relation service service {
 serviceName:String;
 operationName:String;
 parameterName:String;
 compositeName:String;

 checkonly domain aPSM service1:apsm::Service {
  name=serviceName,
  operations=operation1:apsm::Operation {
   name=operationName,
   parameters=parameter11:apsm::Parameter {
    name=parameterName,
    type=type11:apsm::Type {
    }
   },
   parameters=parameter12:apsm::Parameter {
    name=compositeName,
    type=type12:apsm::Type {
     auxiliaryType=dataComposite1:apsm::DataComposite {
     }
    }
   }
  }
 };

 enforce domain gwt implService2:gwt::ServiceImpl {
  name=serviceName+'Impl',
  owningPackage=package2:gwt::Package {
   name='server',
   classes=classes->append(implService2)
  },
  operations=operation2:gwt::Operation {
   name=operationName,
   class=implService2,
   parameters=parameter2:gwt::Parameter {
    name=parameterName,
    type=type2:gwt::Type {
    }
   },
   returnType=returnType2:gwt::Type {
    classType=voClass2:gwt::Class {
     name=compositeName,
     owningPackage=sharedPackage2:gwt::Package {
      name='shared',
      classes=classes->append(voClass2)
     }
    }
   }
  },
  implements=service2:gwt::Service {
   name=serviceName,
   operations=operation2_1:gwt::Operation {
    name=operationName,
    class=service2,
```

```
      parameters=parameter2:gwt::Parameter {
       name=parameterName,
       type=type2
      },
      returnType=returnType2
     },
     owningPackage=package2_1:gwt::Package {
      name='client',
      classes=classes->append(service2)
     },
     asyncService=asyncService2:gwt::ServiceAsync {
      name='Async'+serviceName,
      owningPackage=package2_1:gwt::Package {
       name='client',
       classes=classes->append(asyncService2)
      },
      operations=operation2_1_1:gwt::Operation {
       name=operationName,
       class=asyncService2,
       parameters=parameter21:gwt::Parameter {
        name=parameterName,
        type=type2
       },
       parameters=parameter2:gwt::Parameter {
        name=compositeName+'Callback',
        type=type2_1_1:gwt::Type {
         classType=callBack2:gwt::AsyncCallback {
          name='AsyncCallback',
          owningPackage=sharedPackage2:gwt::Package {
           name='shared'
          },
          parameter=voClass2
         }
        }
       },
       returnType=returnType2_1_1:gwt::Type {
        dataType=gwt::DataTypes::void
       }
      }
     }
    }
   };

   where {
    if not type11.auxiliaryType->notEmpty() then
     primaryType_primaryType(type11, type2)
    else
     auxiliaryType_classType(type12, type2)
    endif;
   }
}

relation primaryType_primaryType {
 checkonly domain aPSM type1:apsm::Type {
  primitiveType=primitiveType1:apsm::DataType {
  }
 };

 enforce domain gwt type2:gwt::Type {
  dataType=getGWTPrimitiveType(primitiveType1)
 };
}
```

```
relation auxiliaryType_classType {
 checkonly domain aPSM type1:apsm::Type {
 };

 enforce domain gwt type2:gwt::Type {
 };

 where {
  if type1.auxiliaryType->oclIsTypeOf(apsm::DataComposite) then
   dataCompositeType voClassType(type1, type2)
  else
   true
  endif;
 }
}

relation dataCompositeType_voClassType {
 typeName:String;

 checkonly domain aPSM type1:apsm::Type {
  auxiliaryType=dataComposite1:apsm::DataComposite {
   name=typeName
  }
 };

 enforce domain gwt type2:gwt::Type {
  classType=voClass2:gwt::Class {
   name=typeName,
   owningPackage=sharedPackage2:gwt::Package {
    name='shared'
   }
  }
 };
}

top relation presentationState_clientPackage {
 applicationName:String;

 checkonly domain aPSM application1:apsm::Application {
  name=applicationName,
  useCases=useCase1:apsm::UseCase {
   stateMachine=stateMachine1:apsm::StateMachine {
    states=state1:apsm::State {
     presentation=presentation1:apsm::Presentation {
     }
    }
   }
  }
 };

 enforce domain gwt application2:gwt::Application {
  name=applicationName,

  client=client2:gwt::Package {
   name='client',
   classes=rootPanel2:gwt::Panel {
    name='rootPanel',
    container=rootPanel2
   }
  }
 };

 where {
```

```
   uiComposite_panel(presentation1, rootPanel2);
  }
}

relation uiComposite_panel {
 panelName:String;

 checkonly domain aPSM uiComposite1:apsm::UIComposite {
  name=panelName,
  uiComponents=uiComponent1:apsm::UIComponent {
  }
 };

 enforce domain gwt panel2:gwt::Panel {
  widgets=panel2 1:gwt::Panel {
   name=panelName,
   container=panel2
  }
 };

 where {
  operationTrigger panel(uiComponent1, panel2 1) or
  inputField_textBox(uiComponent1, panel2_1) or
  select_listBox(uiComponent1, panel2_1) or
  staticText label(uiComponent1, panel2 1) or
  checkBox_checkBox(uiComponent1, panel2_1);
 }
}

relation operationTrigger_panel {
 checkonly domain aPSM operationTrigger1:apsm::OperationTrigger {
 };

 checkonly domain gwt panel2:gwt::Panel {
 };

 where {
  uiComposite_panel(operationTrigger1, panel2);
 }
}

relation inputField_textBox {
 checkonly domain aPSM inputField1:apsm::InputField {
 };

 enforce domain gwt panel2:gwt::Panel {
  widgets=textBox2:gwt::TextBox {
   container=panel2
  }
 };
}

relation staticText_label {
 checkonly domain aPSM staticText1:apsm::StaticText {
 };

 enforce domain gwt panel2:gwt::Panel {
  widgets=label2:gwt::Label {
   container=label2
  }
 };
}
```

```
  relation select_listBox {
   checkonly domain aPSM select1:apsm::Select {
   };

   enforce domain gwt panel2:gwt::Panel {
    widgets=listBox2:gwt::ListBox {
     container=panel2
    }
   };
  }

  relation checkBox_checkBox {
   checkonly domain aPSM select1:apsm::CheckBox {
   };

   enforce domain gwt panel2:gwt::Panel {
    widgets=checkBox2:gwt::CheckBox {
     container=panel2
    }
   };
  }

  query getGWTPrimitiveType(apsmPrimitiveType:apsm::DataType):gwt::DataTypes {
   if apsmPrimitiveType=apsm::DataType::string then gwt::DataTypes::string
    else if apsmPrimitiveType=apsm::DataType::integer then gwt::DataTypes::integer
     else if apsmPrimitiveType=apsm::DataType::boolean then gwt::DataTypes::boolean
      else if apsmPrimitiveType=apsm::DataType::void then  gwt::DataTypes::void
       else gwt::DataTypes::unknown
      endif
     endif
    endif
   endif
  }
}
```

6 Conclusion

In this document, we presented three sets of QVT relations as means of implementing transformations in a model-driven method for web development. The meta-model upon which the transformations are based is an abstract web model called APSM. The first and the most important set transforms PIMs to the APSM. We presented this set with the details of the transformations as well as their visual representations. Two other transformations transform the APSM to specific PSMs defined based on AndroMDA and WebRatio. The APSM-to_SPSM transformations were presented as complete listings plus their visual appearance. Further work will be required to develop a tool to import the outputs of transformations directly into their target platforms.

References

[1] OMG, MDA Guide Version 1.0.1, 12-06-2003

[2] Almeida, P. J. Dijkman, R. van Sinderen, M. Pires, L. F.: On the Notion of Abstract Platform in MDA Development. In: Proceedings of the 8th IEEE International Enterprise Distributed Object Computing Conference (EDOC 2004),pp: 253-263.

[3] OMG, MOF QVT Final Adopted Specification, November 2005

[4] P. Stevens, A Landscape of Bidirectional Model Transformations, Generative and Transformational Techniques in Software Engineering II: International Summer School, GTTSE 2007, Pages: 408 - 424

[5] James L. Peterson. Theory and the Modeling of Systems. Prentice-Hall, N.J., 1981.

[6] Fatolahi, A. Somé, S. S. Lethbridge, T. C. Designing a Map of Mappings : Visulaization of QVT Relations using Basic Petri-Nets. Proceedings of the Second International Workshop on Future Trends of Model-Driven Development (FTMDD 2010). Pp 35-45.

[7] Fatolahi, A. Somé, S. S. Lethbridge, An Integrated Visual Language for Modeling Web UIs, Available from … August 2010

[8] Fatolahi, A. Somé, S. S. Lethbridge, A Meta-Model for Model-Driven Development of Web Applications, Available from www.site.uottawa.ca/eng/school/publications/techrep/2010/TR-2010-04.pdf, July 2010

[9] AndroMDA, www.andromda.org, 15-02-2007

[10] WebRatio, www.webratio.com, 6-5-2008