FROM SEQUENCE DIAGRAMS UML 2.x TO FD-DEVS BY MODEL TRANSFORMATION

Roberto Pasqua, Damien Foures, Vincent Albert, Alexandre Nketsa

FROM SEQUENCE DIAGRAMS UML 2.x TO FD-DEVS BY MODEL TRANSFORMATION

Roberto Pasqua[1,2]
Damien Foures[1,2]
Vincent Albert[1,2]
Alexandre Nketsa[1,2]
[1]CNRS ; LAAS ; 7 avenue du colonel Roche, F-31400 Toulouse, France
[2]Univ de Toulouse ; LAAS ; F-31400 Toulouse, France
E-mail: {rpasqua,dfoures,valbert,alex}@laas.fr

KEYWORDS
MDA, meta-modeling, Sequence Diagrams, Finite and Deterministic DEVS, discrete event simulation

ABSTRACT

This work transforms sequence diagrams to Finite and Deterministic DEVS (FD-DEVS) in Model-Driven Engineering field. The main goal is the formalisation of behaviours, described with UML sequence diagram, to make verification activity by space state exploration and to make validation activity of a set of traces by simulation. In this context, we have chosen to elaborate a model transformation. This paper shows how, after the construction of meta-model for sequence diagrams and for Finite and Deterministic DEVS, it is possible to automate the transformation from one instance of source meta-model to one instance of destination meta-model. The source model is a sequence diagram and the target model is a FD-DEVS component. The destination model is converted into DEVSJava code to simulate its execution.

INTRODUCTION

Model-Driven Engineering offers tools, languages and standard notions to establish and transform models. Today, employment of models lies at the bottom of systems engineering. One of the main purposes in this field is the ability to reuse these models. For that we will have to expand the knowledge about model handling.

Context

System Engineering defines a series of approaches to the development of complex system through the definition of a process from management of requirements to the system release. In every step of the development it is possible to make the simulation predict system behaviour. Verification and Validation (V&V) is a key step towards the development of complex systems and we need formal languages and simulations to do this. The Unified Modeling Language (UML) is a very expressive language which designs the system behaviour but it is not enough formal to provide that this comportment be truly follow after the modelling. The more the language is expressive the less formal it is. The Object Management Group (OMG) introduces important novelties in the 2.0 version of UML. Several of these changes concern the Interaction Diagrams and, in particular, the expressiveness of sequence diagrams. (Micskei and Waeselynck 2011) give an overview about the proposed formal semantics in the several papers. In this context we will concentrate our regard on the concept of trace which is the main construct of the sequence diagrams semantics. In UML specification the trace is defined like *"a sequence of event occurrences, each of which is described by an OccurrenceSpecification in a model"*. In particular in order to make a V&V process, we need to make the distinction between valid and invalid traces. We want to consider the valid trace and to ignore the invalid trace. The DEVS formalism has a well defined semantic to the simulation. FD-DEVS is a subset of DEVS and it has a finite state space, so we use this formalism because we can make the verification process through an exploration of its state space. Then, the transformation to sequence diagrams into FD-DEVS (SD2FDDEVS) allows us to verify, with model checking techniques or validate by simulation, the composition of the trace issue from the scenario described by sequence diagrams. Moreover, we consider specification of invalid trace that we transform into linear temporal logic (LTL) constraints. Today it does not exist a tool to check the LTL constraints in DEVS although it is a current research topic (Hwang and Zeigler 2009).

Approach

We try to develop a platform to formal verification and validation of a simulation models. A model is a hypothetical description of a complex entity that is used to describe a particular behaviour of a real entity. UML allows to modelling a behaviour by three diagrams: Activity, Interaction and StateMachine. A transforma-

tion which comes toward the formalisation of UML behaviour is (Foures et al. 2012), where the authors transform SysML activity diagrams into Petri Nets. With the same approach, we propose the formalisation of sequence diagrams, which are a kind of interaction diagrams. In order to realize it we also develop a transformation based on Atlas Transformation Language (Atlas group and INRIA 2006) to put in practice the specified rules established between two meta-models. This paper is structured as follow: in the next section we discuss about associated work, after this, we present the models and the meta-models in section named preliminaries; in the section named transformation we explain the transformation rules and our method; in the second last section we present the transformation with an example and, finally, in the last section we conclude the paper with some proposal to future works.

RELATED WORKS

In view of their importance in describing the scenarios, several transformations concerning sequence diagrams have been developed. In (Ouardani et al. 2006), the authors transform sequence diagrams into Petri Nets without considering the time explicitly. The sequence diagrams meta-models are reduced to take into account only messages and lifelines (objects in their notation). Another work that transforms sequence diagrams into Petri Nets is (Ameedeen and Bordbar 2008), where the authors focus on the flow of events described with sequence diagrams and they do not consider the constraints which can appear in the scenario. Here the decomposition of sequence diagrams into fragments and mapping them into "Petri Nets blocks" is interesting. We used the same approach. Sequence diagrams have not only been transformed into Petri Nets, but also into State Machines by graph transformation (Gronmo and Moller-Pedersen 2011) and into Communicating Sequential Processes (Dan 2010) to enable a formal verification and analysis. We can not forget (Sqali and Trojet 2009) that translates scenario (described by Message Sequence Charts) in state machine (represented by DEVS). In this work, the authors show the advantages of the use of coupled DEVS, which enables the behaviour simulation of every objects of the system.

PRELIMINARIES

Models

Sequence Diagrams
Sequence diagrams are the most used diagrams to display interactions in UML. The sequence diagrams show the interactions between objects and how they are temporally ordered. An interaction describes how the instances interact dynamically among themselves and it describes the messages exchanged. An interaction can represent a scenario or a use case. Figure 1 shows an Interaction composed by two Lifelines, one Message and four OccurrenceSpecification. There are two kinds of OccurrenceSpecification : message and execution. Above all that, they are the CombinedFragment. It can be composed of one or most operators and UML offer a few operation kind (see meta-model class InteractionOperandKind in figure 6). We focus our attention on alternative (ALT), negative (NEG) and iterative (LOOP) fragments: ALT allows to modelling a conditional construct, NEG is used to show a forbidden behaviour and LOOP represents the recursive part of the diagrams. In this work we consider only the negative fragments.

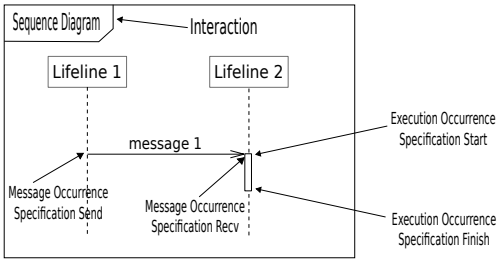

Figure 1: Graphic formalism of sequence diagrams

Finite and Deterministic DEVS
The FD-DEVS formalism is defined in (Hwang and Zeigler 2009). We use two definitions as follows:

Definition 1. *An atomic FD-DEVS is a 7-tuple:*

$$A = \langle X, Y, S, s_0, \tau, \delta_x, \delta_y \rangle \tag{1}$$

where: X is a finite set of input events; Y is a finite set of output events; S is a non-empty and finite states set; $s_0 \in S$ is the initial state; $\tau : S \to \mathbb{Q}_0^{+,\infty}$ is the maximum sojourning time of a state where $\mathbb{Q}_0^{+,\infty}$ denotes a set of non-negative rational numbers plus infinity; $\delta_x : S \times X \to S$ is the external transition function; $\delta_y : S \to S \times Y^{\emptyset}$ is the internal transition function where $Y^{\emptyset} = Y \cup \{\emptyset\}$ and $\emptyset$ denotes the silent event.

Definition 2. *A coupled FD-DEVS is a 5-tuple:*

$$C = \langle X, Y, \{M_d\}, D, EIC, IOC, IC \rangle \tag{2}$$

where: X is a finite set of input events; Y is a finite set of output events; $\{M_d \,|\, d \in D\}$ is an index set of FD-DEVS models. $\{M_d\}$ can be either an atomic or an coupled model; D is a finite set of names of subcomponents; EIC is a set of external input connectors; IOC is a set of internal output connectors; IC is a set of internal connectors.

Semantic Figure 2 shows the graphic semantic chosen to represent a FD-DEVS model. *phase* 1 is a transitive state since $t_a = 0$. Then δ_y is immediately triggered

after entering in *phase* 1. An output event is triggered on Y and next state is the passive state *phase* 2. When an input event occurs on X, δ_x is triggered updating the state to *phase* 3.

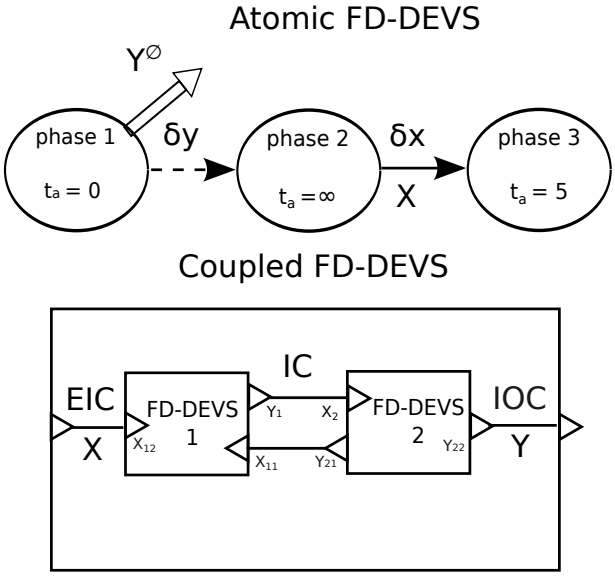

Figure 2: Graphic semantic of FD-DEVS

Meta-Models

The metamodeling process is used in accordance with the OMG standard Meta-Object Facility (MOF). For this, we use Eclipse Modeling Framework (EMF) that uses Ecore as metamodeling language.

Sequence Diagrams
OMG defined the sequence diagrams in (Omg 2011). In accordance with this specification, the sequence diagrams meta-model is shown in Figure 6. We fix our attention on the main classes considered in the transformation : Interaction, Lifeline, OccurrenceSpecification, Message and CombinedFragment. In particular, we focus our interest in GeneralOrdering which allows us to give a temporal order of the specification occurrences present in the interaction. All instances of GeneralOrdering contain the valid trace. To explicitly consider the time in sequence diagram, we use DurationConstraint.

Finite and Deterministic DEVS
The meta-model shown in Figure 7 is a rigorous description of FD-DEVS definition. An FD-DEVS instance can be an Atom or a Coupled and it is composed of Events. In the case of an Atom, it is also composed of States and Transitions. A State is composed of one Time and a set of Variable. A Transition is defined between two States. In the case of a Coupled, it is composed of FD-DEVSs and Connectors. Every instance of these classes can have a name so we have introduced the Ecore class ENamedElement with inheritance relationships.

TRANSFORMATIONS

They are founded on ATL. The first transformation is a Model-to-Model transformation realized with an ATL module while, the second one, is a Model-to-Text transformation realized with ATL query.

SD2FDDEVS

The transformation rules, defined at the Meta-Model level, allow us to transform an input model (compliant with sequence diagrams meta-model) represented in the XMI (Extensible Mark-up Language Metadata Interchange) format into another model compliant with FDDEVS meta-model represented in the same format. The input model can be created with a graphic editor that support the UML 2.x specification.
The main rules of correspondence established between sequence diagrams and FDDEVS are illustrated as follows:

- one interaction that contains at least two lifelines generates a coupled;
- every lifeline that is present in the interaction, generates an atom included in the associated coupled;
- every message generates an input and an output event (we consider complete MessageKind and synchCall, asynchCall MessageSort) in the atom. These events will be used to create internal connectors.
- every specification occurrence that is not contained in a combined fragment generates a state in the atom.

To generate the transitions we use the class GeneralOrdering of the sequence diagram meta-model. All instances of this class give a temporal order to the events associated with the specification occurrences. Two consecutive specification occurrences, that belong in the same lifeline, generate an output internal transition. If two consecutive specification occurrences do not belong in the same lifeline it generates an "artificial" state and one or two transitions, that can be internal or external, depending on the type of specification occurrences. At the same time the states are created, we create the Time and the set of Variables associated. This variables are the atomic propositions that we use to check LTL properties on the model. For the combined fragments we consider the case of negative fragments with one message inside. For each of these fragments we generate two text files that contain a LTL constraint to verify the model construction.

EXAMPLE

In this example we illustrate our transformation by studying a use case of an automatic coffee machine that allows the possibility of understanding the key concepts of this work. The user can choose one product to display the price. The controller cannot accept a second choice. This is specified by the negative fragment in Figure 3. After this, when the interface sends a message to communicate that the money in input is enough, the controller starts the engine to make a coffee. At the end of the process (process of production), the controller sends a last message to the interface.

Source model

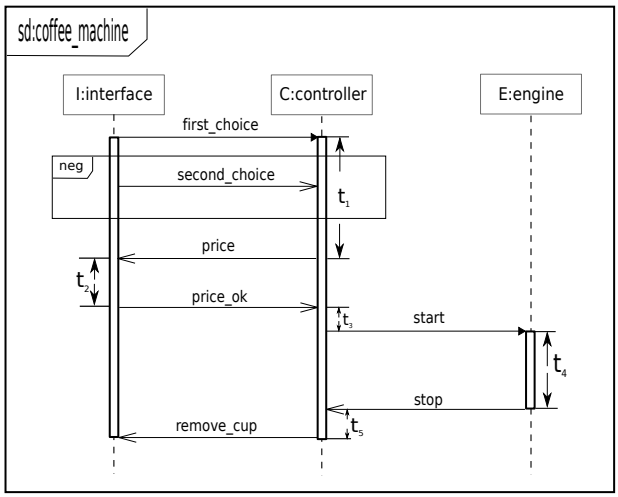

Figure 3: Sequence diagrams of use case

We use Table 1 to show the temporal order of specification occurrences. An instance of GeneralOrdering is defined for example like this:

$$GO\,17 \Longrightarrow \begin{cases} before : MessOccSpecRecv4 \\ after : ExecOccSpecFinish \end{cases}$$

t_i represents duration constraints into two following messages.

Target model

After transformation we have a coupled FD-DEVS composed by three atoms (Interface, Controller and Engine) and seven internal connectors (one for each message). Figure 4 shows the graphical representation of the atom Controller. We point out that "artificial" states and transitions depend on instances of GeneralOrdering. With this model, we can make a LTL verification and/or a validation by simulation. Table 2 shows binary variables associated to first six states of Controller. To generate these variables we use the events attached on

Table 1: List that represents GeneralOrdering

OccurrenceSpecification	Lifeline
ExecOccSpecStart	Interface
MessOccSpecSend	Interface
MessOccSpecRecv	Controller
ExecOccSpec0Start	Controller
MessOccSpecSend0	Controller
MessOccSpecRecv0	Interface
MessOccSpecSend1	Interface
MessOccSpecRecv1	Controller
MessOccSpecSend2	Controlleur
MessOccSpecRecv2	Engine
ExecOccSpec1Start	Engine
MessOccSpecSend3	Engine
ExecOccSpec1Finish	Engine
MessOccSpecRecv3	Controller
MessOccSpecSend4	Controller
ExecOccSpec0Finish	Controller
MessOccSpecRecv4	Interface
ExecOccSpecFinish	Interface

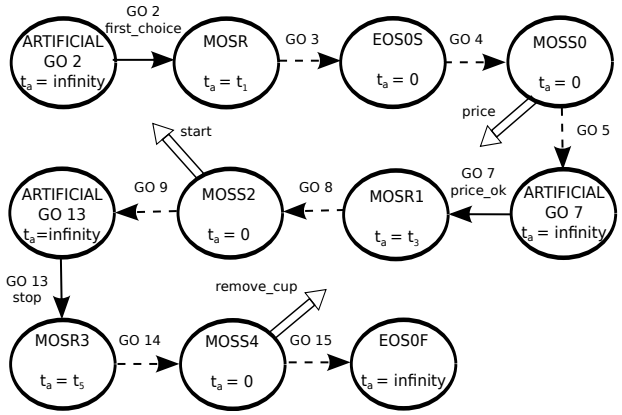

Figure 4: FD-DEVS model of Controller

every specification occurrences: R is for received event and S is for send event. The LTL constraint generated by transformation for the atom Controller is the follow:

$$first\_choice\_R = 1 \rightarrow (second\_choice\_R = 0 \cup price\_S = 1)$$

To simulate the FD-DEVS model we have devellopped the FDDEVS2DEVSJava transformation with ATL query and we use DEVS-Suite (DEVS-SuiteSim 2010) to execute the code. We give in Appendix the code generated by this transformation for the Controller. Figure 5 shows the execution trace of the simulation up to 10 time unit.

CONCLUSION

We have shown how is possible the formalisation of sequence diagrams by means of model transformation. We

Table 2: Variables associated to state

$ARTIFICIALGO2$	MR
$first\_choice\_R = 0$	$first\_choice\_R = 1$
$second\_choice\_R = 0$	$second\_choice\_R = 0$
$price\_S = 0$	$price\_S = 0$
$price\_ok\_R = 0$	$price\_ok\_R = 0$
$start\_S = 0$	$start\_S = 0$
$stop\_R = 0$	$stop\_R = 0$
$remove\_cup\_S = 0$	$remove\_cup\_S = 0$
$AES0S$	$MS0$
$first\_choice\_R = 1$	$first\_choice\_R = 1$
$second\_choice\_R = 0$	$second\_choice\_R = 0$
$price\_S = 0$	$price\_S = 0$
$price\_ok\_R = 0$	$price\_ok\_R = 0$
$start\_S = 0$	$start\_S = 0$
$stop\_R = 0$	$stop\_R = 0$
$remove\_cup\_S = 0$	$remove\_cup\_S = 0$
$ARTIFICIALGO7$	$MR1$
$first\_choice\_R = 1$	$first\_choice\_R = 1$
$second\_choice\_R = 0$	$second\_choice\_R = 0$
$price\_S = 1$	$price\_S = 1$
$price\_ok\_R = 0$	$price\_ok\_R = 1$
$start\_S = 0$	$start\_S = 0$
$stop\_R = 0$	$stop\_R = 0$
$remove\_cup\_S = 0$	$remove\_cup\_S = 0$

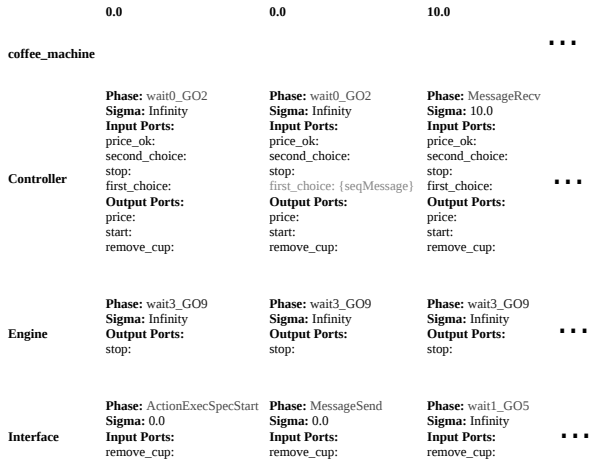

Figure 5: Output log file of simulation

have used the FD-DEVS as destination formalism, enriched with a set of state variables, so we can make a verification activity through space state exploration of the system source model represented by the sequence diagram. Moreover, it is possible to make a validation activity by simulation, to analyse the execution of a valid set of traces. Our method allows to consider the time explicitly in the sequence diagram construction to translate this in clock device offered by FD-DEVS. The work in this purpose is not complete and we point out the following topics for future work: improvement of explicitly time consideration; transformation of a source model that contains others combined fragments like ALT, LOOP, etc.; possibility to generate different LTL constraints; merge of many use cases (many interactions) to generate a set of FD-DEVS components.

REFERENCES

Ameedeen M. and Bordbar B., 2008. *A Model Driven Approach to Represent Sequence Diagrams as Free Choice Petri Nets*. In *Enterprise Distributed Object Computing Conference, 2008. EDOC '08. 12th International IEEE*. ISSN 1541-7719, 213 –221. doi:10.1109/EDOC.2008.42.

Atlas group L. and INRIA N., 2006. *ATL:Atlas Transformation Language ATL User Manual. OMG specification*, version 0.7.

Dan L., 2010. *QVT Based Model Transformation from Sequence Diagram to CSP*. In *Engineering of Complex Computer Systems (ICECCS), 2010 15th IEEE International Conference on*. 349 –354. doi:10.1109/ICECCS.2010.47.

DEVS-SuiteSim, 2010. *Project Information*. URL `http://devs-suitesim.sourceforge.net/`.

Foures D.; Albert V.; Pascal J.; and NKETSA A., 2012. *Automation of SysML activity diagram simulation with model-driven engineering approach*. In *Spring Simulation Multiconference (SpringSim'12)*. Orlando (USA), 8p.

Gronmo R. and Moller-Pedersen B., 2011. *From UML 2 Sequence Diagrams to State Machines by Graph Transformation. Journal of Object Technology*, 10, 8:1–22. ISSN 1660-1769. doi:10.5381/jot.2011.10.1.a8. URL `http://www.jot.fm/contents/issue_2011_01/article8.html`.

Hwang M.H. and Zeigler B.P., 2009. *Reachability Graph of Finite and Deterministic DEVS Networks. IEEE T Automation Science and Engineering*, 6, no. 3, 468–478.

Micskei Z. and Waeselynck H., 2011. *The many meanings of UML 2 Sequence Diagrams: a survey. Software and Systems Modeling*, 10, 489–514. ISSN 1619-1366. URL `http://dx.doi.org/10.1007/s10270-010-0157-9`. 10.1007/s10270-010-0157-9.

Omg, 2011. *OMG Unified Modeling Language (OMG UML), Superstructure Specification (Version 2.4.1)*. Tech. Rep. OMG Document Number: formal/2011-08-06, Object Management Group.

Ouardani A.; Esteban P.; Paludetto M.; and Pascal J., 2006. *A Meta-modeling Approach for Sequence Diagrams to Petri Nets Transformation within the requirements validation process.* In *Proceedings of the European Simulation and Modeling Conference (ESM2006).* 345–349.

Sqali M. and Trojet W., 2009. *Multi-models approach for describing and verifying constraints based interactive systems.* In *International Journal of Electrical and Electronics Engineering (IJEEE).* Tokyo(Japan), vol. 3, 342–352.

APPENDIX

Code Java

Controller.java

```
package DEVS_seq;

import GenCol.*;
import model.modeling.*;
import model.simulation.*;
import view.modeling.ViewableAtomic;
import view.simView.*;

public class Controller extends
ViewableAtomic{

protected entity seqMessage;

public Controller(){}

public Controller(String name){
super(name);
addInport("first_choice");
addInport("second_choice");
addInport("price_ok");
addInport("stop");
addOutport("price");
addOutport("start");
addOutport("remove_cup");}

public void initialize(){
seqMessage = new entity
("seqMessage");
holdIn("wait0_GO2",INFINITY);}

public void deltint(){
if (phaseIs("MessageSend0"))
        holdIn("wait2_GO7",INFINITY);
else if (phaseIs("MessageSend2"))
    holdIn("wait4_GO13",INFINITY);
else if (phaseIs("MessageRecv1"))
        holdIn("MessageSend2",0);
else if (phaseIs
                ("ActionExecSpec0Start"))
        holdIn("MessageSend0",0);
else if (phaseIs("MessageRecv3"))
        holdIn("MessageSend4",0);
else if (phaseIs("MessageSend4"))
    holdIn("ActionExecSpec0Finish",
    INFINITY);
else if (phaseIs("MessageRecv"))
        holdIn("ActionExecSpec0Start",0);}

public message out(){
message m = new message();

if (phaseIs("MessageSend0"))
m.add(makeContent
         ("price",seqMessage));
else if (phaseIs("MessageSend2"))
m.add(makeContent
         ("start",seqMessage));
else if (phaseIs("MessageSend4"))
m.add(makeContent
     ("remove_cup",seqMessage));

                return m;}

public void deltext
(double e, message m){
if (phaseIs("wait2_GO7")){
   for (int i=0; i< m.size();i++)
  if (messageOnPort(m,"price_ok",i))
        holdIn("MessageRecv1",1);}
else if (phaseIs("wait4_GO13")){
   for (int j=0; j< m.size();j++)
  if (messageOnPort(m,"stop",j))
        holdIn("MessageRecv3",1);}
else if (phaseIs("wait0_GO2")){
   for (int k=0; k< m.size();k++)
  if (messageOnPort
                (m,"first_choice",k))
        holdIn("MessageRecv",10);}

        }

}
```

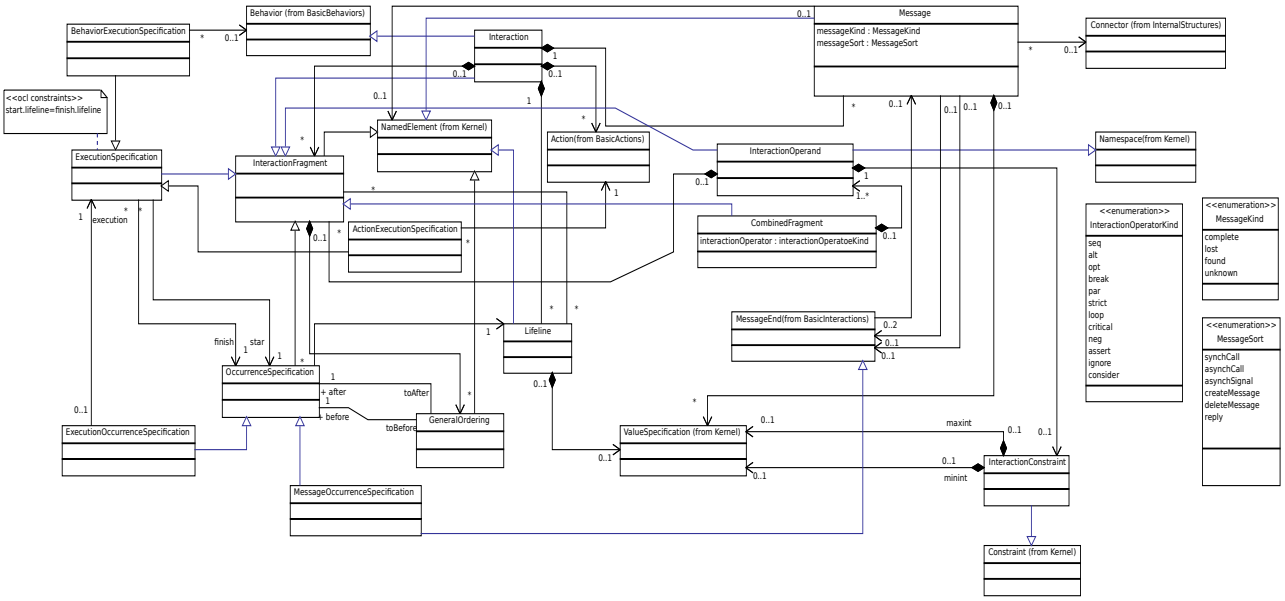

Figure 6: Meta-models Sequence Diagrams

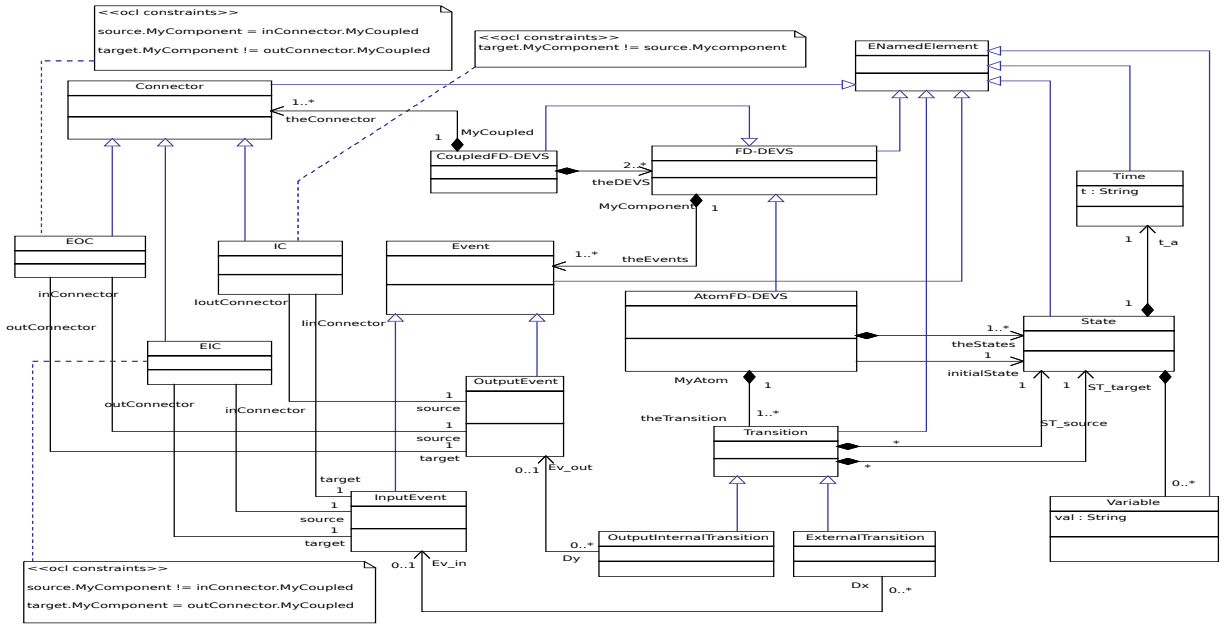

Figure 7: Meta-models Finite and Deterministic DEVS