MODELING AND SIMULATION OF BEHAVIORAL SCENARIOS BY USING COUPLED DEVS MODELS

Sqali Mamoun[(a)], Lucile Torres[(b)]

[(a) (b)]LSIS (Laboratoire des Sciences de l'Information et des Systèmes)
Université Paul Cezanne Aix-Marseille III
avenue Escadrille Normandie-Niemen13397 MARSEILLE CEDEX 20

[(a)] mamoun.sqali@lsis.org, [(b)]lucile.torres@lsis.org

ABSTRACT
Sequence diagrams, such as those included in UML, are widely used to express behavioral requirements of a system. They are employed to refine use cases and to induce an abstract model of the system behavior. In the approach we propose, the global behavioral model is obtained from a set of sequence diagrams and it is expressed in the DEVS (Discrete EVent System Specification) formalism. Finally, the DEVS model can be validated and verified by simulation.

Keywords: Sequence Diagrams, DEVS modeling, Simulation.

1. INTRODUCTION

In general, the scenarios use Sequence Diagrams (SD) to represent graphically the interactions between objects. They can be composed by using flow control operators (alternative, sequence, parallelism and repetition) in order to form more complex scenarios. A SD show, as parallel vertical lines, different processes or objects that live simultaneously, and, as horizontal arrows, the messages exchanged between them, in the order in which they occur.

This allows the specification of simple runtime scenarios in a graphical manner. It seems illusive, to define the system by conceiving all its scenarios. Moreover, designing the system behavior directly with statecharts is not an intuitive process, since the concept of state is not obvious in the first stages of the development process. Many approaches address the scenario synthesis problem (Liang, Dingel and Diskin 2006) and makes possible to induce a total behavior model expressed in a state machine format starting from a set of scenarios. All of them agree with the need to have a sufficient number of scenarios. They especially differ in the source notation and semantics, in the target notation, in their ability to support scenario composition mechanisms, in their use of merging techniques of identical states, and in the implemented synthesis algorithm. We present in the following sections, the SD notation, the Discret EVent system Specification (DEVS) formalism, and the synthesis problem.

When the synthesis has been realized, the induced coupled DEVS model represents the system behavior that includes all the behaviors expressed by the scenarios and it is used to validate the specification by simulation.

2. RECALL

2.1. Sequence Diagrams

A SD is one of the five diagrams used in UML for modeling the system dynamic aspects of systems. A SD shows an interaction, consisting of a set of objects and their relationships. The diagram includes messages that may be dispatched among them with an emphasis on their time orderings. Graphically, a SD is a table that shows objects arranged along the X axis and messages, ordered in increasing time, along the Y axis.

The vertical lines in the SD represent the lifelines of the objects taking part in the scenario. Time is usually assumed to flow downwards along each lifeline. The directed arrows going across the lifelines represent the causal links from a sent event (the source of the arrow) to the corresponding received event (the target of the arrow), with the label on the arrow denoting the event.

A great number of notations are commonly used for the description of SD: Message Sequence Charts (MSC) defined within an international standard (ITU 2000), that can be distinguished between basic MSC (bMSC) and compound MSC (hMSC for High-Level MSC), Live Sequence Charts (LSC), which are an extension of bMSC proposed by (Damm and Harel, 2001), the UML SD (OMG 2005), which are a simplified version of bMSC (Brian and Hans 2004), …

In this article, we propose a method for translating a set of SD into state machines represented in the formal DEVS specification. We use a simple example to illustrate our approach. This example (see Figure 1) shows how to obtain a coffee from a coffee machine. The SD is composed by three objects: Customer, CoffeeMachine and Stock. Customer is considering as an environment object.

A SD has a structure (E, ≤, α, Φ, A, I, T) where:

- E is a set of events

- ≤ is a partial ordering between events imposed by lifelines and messages
- α is a set of action names
- Φ denotes the location to an event (i.e the object affected by the event)
- A is a set of actions (message sending and receiving);
- I is a set of objects
- T is a set of interaction time constraints;

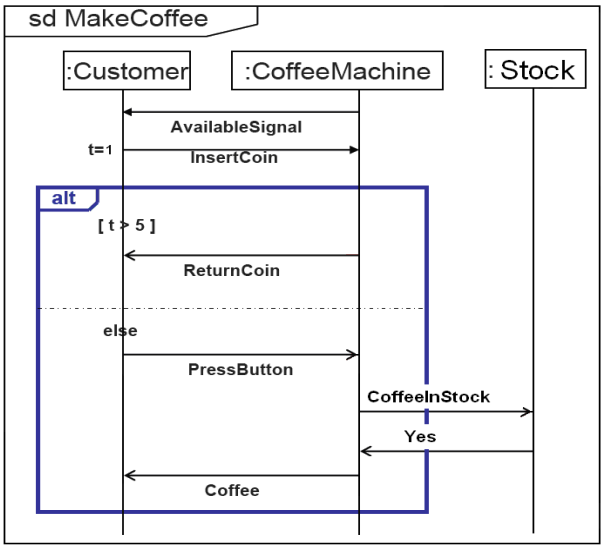

Figure 1: A SD for making coffee

The behavior represented by a SD is a set of event sequences determined by the causal priority. The events are totally ordered for one considered object. The causal relationship determines a partial order, noted ≤, on the events between all the objects. This partial order can be derived from the SD in respect with two principal rules:

- an event e drawn higher than another event e' on the same lifeline of an object precedes necessarily e';
- the event associated with a message sending precedes necessarily the event associated with the reception of this message (in the case of an asynchronous communication). For a synchronous communication, messages are used to be considered instantaneous in the order to do not differentiate the message sending from the message reception.

2.2. The DEVS Formalism

The DEVS formalism introduced by Zeigler (Zeigler 2000) provides a means for modeling discrete event system in a hierarchical and modular manner. The use of this formalism facilitates the modeling activity and guarantees a best accuracy of the model by decomposing the system into component models and by specifying the coupling between them. There are two kinds of DEVS models, atomic models and coupled ones.

2.2.1. Atomic DEVS model

An atomic model describes the behavior of a component, which is indivisible, in a timed state transition level. Formally, an atomic model is defined by a 7-tuple <X, Y, S, δint, δext, λ, ta> where:

- X is the set of the external input events;
- Y is the set of the external output events;
- S is the set of sequential states;
- δint : S -> S is the internal transition function that defines the state changes caused by internal events;
- δext : Q × X -> S is the external transition function, where Q = {(s,e)|s∈ S, 0≤e≤ta(s)} is the set of total state; this function specifies the state changes due to external events, with the ability to define a future state according to the elapsed time in the current state;
- λ is the output function that generates output events;
- ta : R+ ∪ {0, ∞} gives the lifetime of the states, where R+ ∪ {0, ∞} is the set of positive reals with 0 and ∞; ta(s) represents the interval during which the model remains in the state s if external event occurs.

While the internal transition function expresses the autonomous evolution of the model, the external transition function defines its evolution when occurring external events.

2.2.2. Coupled DEVS model

A coupled model is a compound component consisting of atomic models and/or coupled models. The coupled model can itself be employed as a component in a larger coupled model, there by giving rise to the construction of complex models with hierarchical structures. A coupled model is formally defined by a 7-tuple < X, Y, M, EIC, EOC, IC, SELECT > where:

- X is the set of input events;
- Y is the set of output events;
- M is the set of all the DEVS component models;
- EIC ⊆ X × UiXi is the external input coupling relation;
- EOC ⊆ UiYi × Y is the external output coupling relation;
- IC ⊆ UiXi × UiYi is the internal coupling relation;
- SELECT: 2M - ϕ, M is a function which chooses one model when more than 2 models are scheduled simultaneously.

EIC, EOC and IC specify the connections between the input and output ports of the various DEVS models.

3. STATE OF THE ART

Many approaches address the scenario synthesis problem, like (Harel, Kugler and Pnueli 2005) who

proposed a synthesis approach using the scenario-based language of Live Sequence Charts (LSC) as requirements, and synthesizing a state-based object system composed of a collection of finite state machines. (Letier, Kramer and Uchitel 2005) who presents a technique to generate Labelled Transition System (LTS) from Hight Level Message Sequence Chart (hMSC), in this approach, complex system behaviour can be modelled by parallel composition of the component LTS models. Parallel composition models components that execute asynchronously but synchronize on shared events; also they presented a technique to detect implicit scenarios.

Ziadi, Hélouët and Jézéquel (2004) have proposed an idea to synthesize statecharts starting from scenarios expressed by UML2.0 Sequence Diagrams, and give an algorithm for synthesizing a composition of statecharts between them. Also (Damas and Lambeau 2006) has presented an approach to generate Lablled Transition System from a collection of basics MSC's, and use a technique to merge the identical states. That will help us for our approach, and will return our atomic DEVS models minimal and determinists.

4. FROM SEQUENCE DIAGRAMS TO DEVS

4.1. Transformation steps of SD without operators

Let us show transformation from SD sets to DEVS in the five following steps.

Step I) Constructing an atomic DEVS model for one object belonging to one scenario:

1. For one object in one SD, identifying an atomic DEVS model (We consider for this example, the CoffeeMachine object).
2. Defining input/output events of the atomic DEVS model (X, Y)
3. Defining a set of states. A state must include information of input/output events in a SD. As shown in the figure 2, each state should be associated with both an external and/or internal transitions (δint, δext) in an atomic DEVS model.

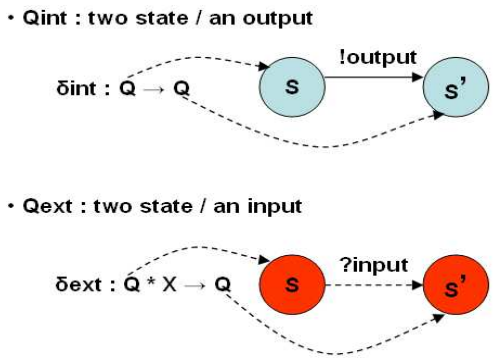

Figure 2: Definition rules of the set of states

4. Defining state transitions between states obtained in 3 from the SD:

A. Showing message transmission among the defined states (figure 3).

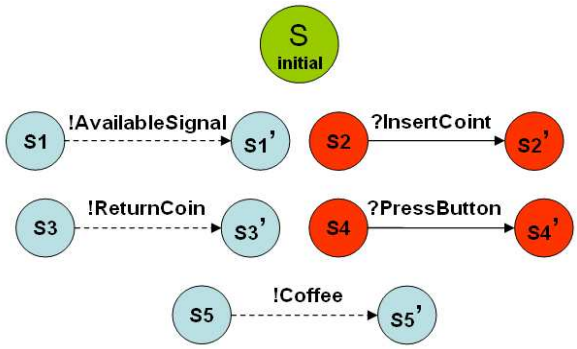

Figure 3: Translating message transmission

B. Merging the start point of the SD and its end point into the initial state of the DEVS model (figure 4).

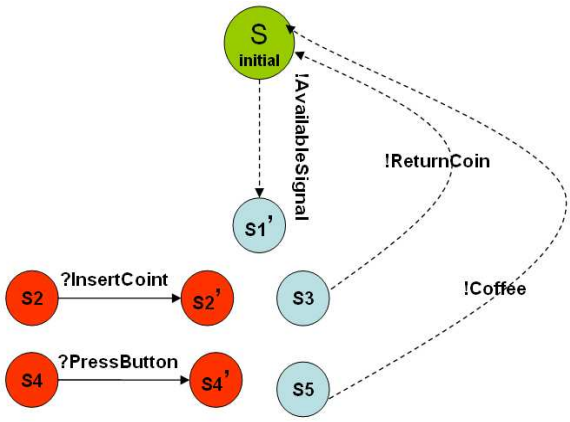

Figure 4: Defining the initial state

C. Translating all the message sequences of the SD to state transitions of the atomic model, and minimizing the number of states if equivalent states exist.

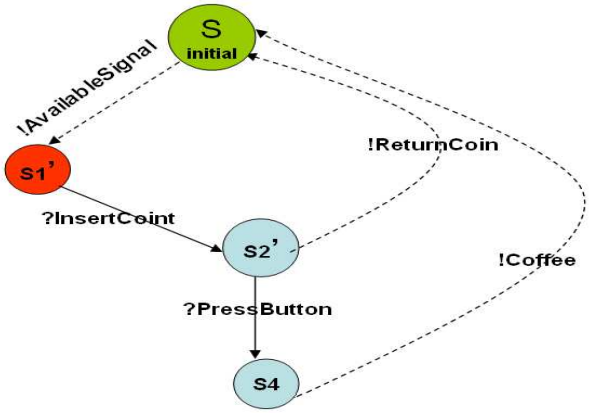

Figure 5: Minimizing the number of states

D. After the previous steps A~C, the specification of time advance functions in atomic DEVS models is remained. If a special time constraint is defined in the SD, the

condition must be translated in the confirmed state. If only an input causes the state transition from a state s, it can be describe by a passive state with ta(s)= ∞. If a state s' describes only output events, then ta(s')=1.

Step II) Repeating the first step (I) for all objects of the considered SD. This step generates the set of atomic models associated with the SD.

Step III) Repeating the step II for all the SD. This step generates the set of atomic models associated with the set of SD.

Step IV) Building one atomic model from the atomic models on a given object. This step generates as many atomic models as there are objects in the set of SD.

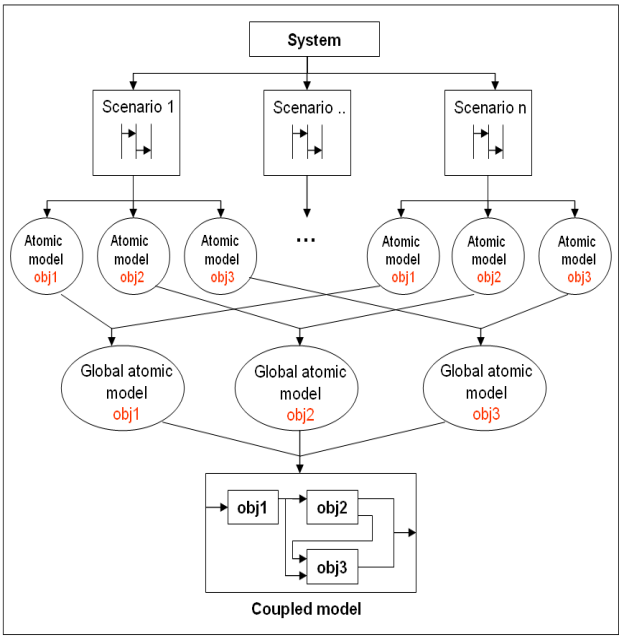

Figure 6: Example of the steps IV and V

Step V) Building a coupled final model which is composed of the atomic models obtained in the previous step, so that the output of an atomic model is the input of another, then all objects in the system can communicate between them. The final coupled model describes the overall behavior of the system.

An algorithm which translates a basic SD into an atomic DEVS model by object was presented in (Sqali and Torres 2008a). Also, the principle of the construction of an atomic DEVS model by object starting from several composed SD, and the construction of coupled DEVS model is described in (Sqali and Torres 2008b).

4.2. Transformation of SD with seq, loop and alt operators

- Seq specifies a sequence between two SD (strong sequential composition).

Da1 ***seq*** *Da2* = <X, Y, S, δint, δext, λ, ta>:

- S =- S1 ∪ S2- {s02} if (Da2 ≠ ∅)
  - S2 if (Da1= ∅)
- X = X1 ∪ X2
- Y = Y1 ∪ Y2
- δint = δint1∪ δint1
- δext = δext1∪ δext2

- Loop specifies an iteration of a SD

Loop *(Da1)* = <X, Y, S, δint, δext, λ, ta>:

- S = - (S1-sn1) ∪ {s01}
  - s0=s01
- X=X1
- Y=Y1
- δint = δint1∪ {sn -> s0} ∨ δext = δext1∪ {Q ×X -> s0}
- Loop (Da∅) = Da∅

- Alt defines a choice between a set of

Da1 ***alt*** *Da2* = <X, Y, S, δint, δext, λ, ta>:

- S =- S1 if (Da1 ≠ ∅ ∧ Da2 = ∅)
  - S2 if (Da1 = ∅ ∧ Da2 ≠ ∅)
  - {s0} if ((Da1 = ∅ ∧ Da2 = ∅)
  - S1 ∪ S2 ∪ {s} if (Da1 and DA2 are loops) ∧ (Da1 ≠ ∅ and Da2 ≠ ∅)
  - S1 ∪ S2 otherwise
- s0 =- A new state if (Da1 and DA2 are loops) ∧ (Da1 ≠ ∅ and Da2 ≠ ∅)
  - s01 if Da2 are loops
  - s02 if Da1 are loops
- X = X1 ∪ X2
- Y = Y1 ∪ Y2
- δint = δint1∪ δint1
- δext = δext1∪ δext2

5. CASE STUDY AND SIMULATION

Let us consider SD of the figure 1 that contains three objects; Customer, CoffeeMachine and Stock. By using the previous steps of transformation into DEVS models, we first obtain the DEVS atomic models represented in the figures 6 and 7.

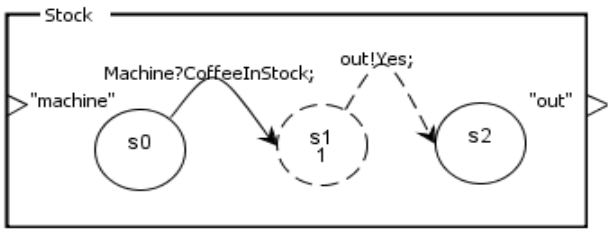

Figure 6: An atomic DEVS model for the object Stock

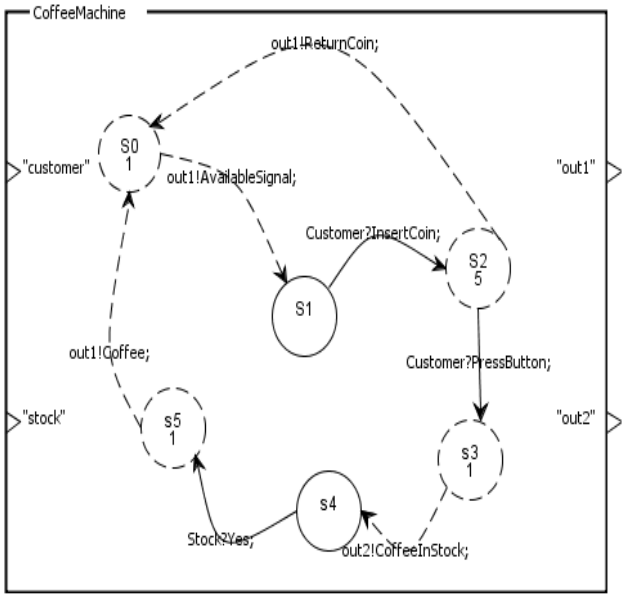

Figure 7: An atomic DEVS model for the object CoffeeMachine

After the global atomic models for all objects of the system have been built, we construct the coupled model given in the figure 8 who describes the overall behavior of the system.

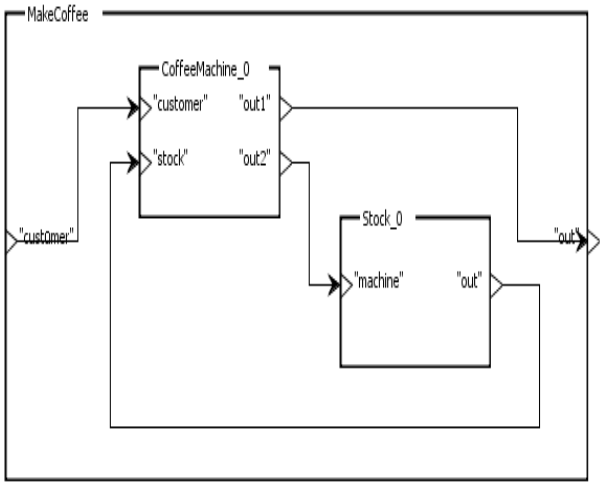

Figure 8: A final coupled model

To validate the specification of the system behavior from the final coupled DEVS model, we use the LSIS-DME tool developed within our laboratory. From a model and a set of data, the simulator provides the simulation results. The dataset is defined by the user who enters all external events supposed to occur during the simulation. For that, he entered for each event the port of the model on which the event will occur and the value of the event. For the final model, with the inputs specified in the figures 9 and 11, the results obtained during the simulation are given in the figures 10 and 12 respectively. The simulation of the global DEVS model produces a sequence of events for the considered sequence of input events belonging to the set of event sequences defined by the SD. We note that the behaviors obtained are exactly the same as those specified in the SD.

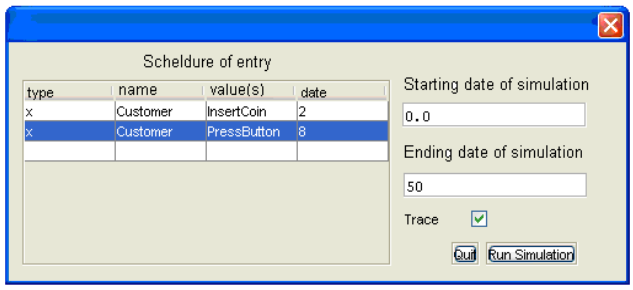

Figure 9: Filling input schedules

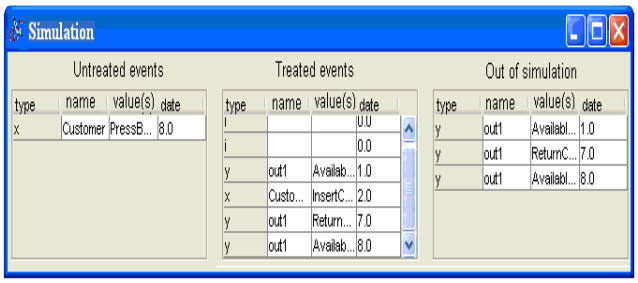

Figure 10: Simulation results for Figure 9

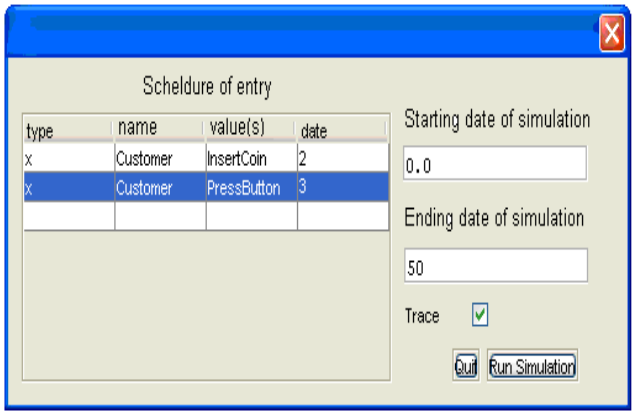

Figure 11: Another filling input schedules

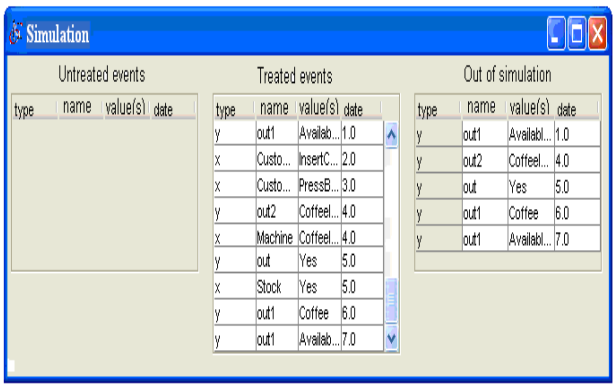

Figure 12: Simulation results for Figure 11

The LSIS_DME tool also provides an interface XML to describe models DEVS textually. This makes possible to transform models DEVS obtained under

format XML. We can thereafter modify them and simulate them. For this, we propose to use XML and XSL to realize the synthesis in models DEVS expressed in XML

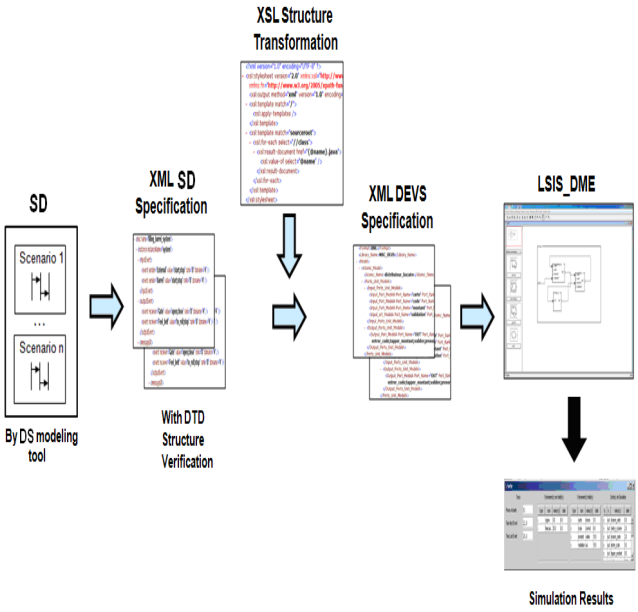

Figure 13: Transformation using XML

The transformation procedure consists to:

1) Generate a XML representation of an atomic DEVS model for each object in all the scenarios of the system, by defining the XML format of the scenarios class with a DTD grammar, to apply XSL language on XML representation of the scenarios models, and generate a XML document for each object of the system.
2) Construct a final coupled DEVS model, composed of the various atomic models obtained, describing the total behavior of the system.

By using the library saxon8.jar, we import the javax.xml.transform.*; javax.xml.transform.stream._ Stream_Result; and javax.xml.transform.stream._ StreamSource;. The following code creates an XSL transformer which has as parameters the path of scenario XML document which we want to transform, and generates a DEVS XML file.

```
// Create a transform factory instance.
TransformerFactory tfactory = TransformerFactory_
.newInstance();
// Create a transformer for the stylesheet.
Transformer trans = null;
    try {
          trans = tfactory.newTransformer(new
        StreamSource("C:/Path/convert.xsl"));
    }
    catch (TransformerConfigurationException ex) {
// Transform the source XML to System.out.
    try {
          trans.transform(
          new StreamSource("C:/Path/scenario.xml), new
          StreamResult( new
          File("exampleSimple.out")));
    }
    catch (TransformerException ex)
```

6. CONCLUSION

Many approaches address the scenario synthesis problem. All of them agree with the need to have a sufficient number of scenarios. They especially differ in the source notation and semantics, in the target notation, and the identification of common states for merging scenarios (Liang, Dingel and Diskin 2006).

We have presented the Sequence Diagrams synthesis into a DEVS model in order to verify and validate by simulation the behavioral specification given by the scenarios. We have chosen a scenario semantics restricted to event sequences with the notion of (repetition, alternativity and sequence). Thereafter, we proposed to use XML and XSL to realize the synthesis in DEVS models expressed in XML (José, Risco-Martin and Mittal 2007).

REFERENCES

Brian, L., Hans, E., 2004. UML 2 toolkit. *Whiley Publishing OMG press*

.

Zeigler, B., Herbert, P., Tag Gon, K., 2000. Theory of Modeling and Simulation. *ACADEMIC PRESS.*

Damas, C., Lambeau, B., Lamsweerde A., 2006. Scenarios, Goals, and State Machines: a Win-Win Partnershi for Model Synthesis. *14th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pp. 197-207. Portland, Oregon, USA.

Damm, W., Harel, D., 2001. LSCs: Breathing life into message sequence charts. *Formal Methods in System Design*, 19(1):45--80.

David, H., Kugler, H., Pnueli, A., 2005. Synthesis Revisited: Generating Statechart Models from Scenario-Based Requirements. *Formal Methods in Software and Systems Modeling*.

Letier, E., Kramer, J., Magee, J., Uchitel, S., 2005. Monitoring and Control in Scenario-Based Requirements Analysis**.** *International Conference on Software Engineering, Proceedings of the 27th international conference on Software engineering.*

José, L., Risco-Martin, Mittal, S., 2007, *A W3C XML Schema for DEVS Scenarios*, submitted to DEVS Integrative M&S Symposium DEVS' 07, Spring Simulation Multi-Conference, March.

ITU, 2000. Message Sequence Charts. Recommendation Z.120. *International Telecommunications Union*. Telecommunication Standardisation Sector.

Liang, H., Dingel, J., Diskin, Z., A, 2006. Comparative Survey of Scenario-based to State-based Model Synthesis Approaches. *SCESM'06 : International Workshop on Scenarios and State Machines: Models, Algorithms, and Tool*, pp.5-12. May, Shangaï, China.

OMG, 2005. *UML 2.0 Specification*. Object Management Group. Avaibale from: http://www.omg.org [August 2005].

Sqali, M., Torres, L., Frydman, C., 2008. Synthèse de scénarios en DEVS (a), *7ème conférence internationale de Modélisation et SIMulation (MOSIM08)*. Paris, 31 mars-2 avril, .

Sqali, M., Torres, L., Frydman, C., 2008. Synthetizing scenarios to DEVS models (b). *Poster Session of SpringSim08*. Ottawa,Canada.

Ziadi, T., Hélouët, L., Jézéquel, J., 2004. Revisiting Statechart Synthesis with an Algebraic Approach. *Proc. of 26th International Conference on Software Engineering (ICSE), IEEE Computer Society*.p. 242-251.Edinburgh, May.