Towards an SDL-DEVS simulator

Pau Fonseca i Casas
Universitat Politècnica de Catalunya
Department of Statistics and Operations Research
C/ Jordi Girona, 31. 08034 Barcelona, Catalonia, Spain
pau@fib.upc.edu

Abstract

Specification and Description Language is a graphical language, standardized under the ITU Z.100 recommendation, widely used to represent telecommunication systems, process control and real-time applications in general. In this paper we present the first prototype that allows executing a XML DEVS representation of a simulation model on a SDL simulator. This execution is based on a transformation of the DEVS representation of a simulation model to an equivalent SDL representation. The simulator used, currently used on production is named SDLPS, and allows to perform a distributed simulation of all the components that represents the model based on a representation of the model using Specification and Description Language.

Keywords: **DEVS; SDL; XML; simulation; specification; formalism**

1. INTRODUCTION

The need of a conceptual model who allows a complete understanding of how the simulation model behaves prior to any implementation is well understood by the simulation community. Also is growing the set of tools that allows to perform a simulation using a formal representation of a model (as an example we can see CPTools, CD++ [1], Cindarella [2], IBM's Tau-Telelogic [3] , etc, using different formal languages, and SDLPS [4] who is used in this work).

This implies that the verification phase (if not avoided) can be performed faster than if we use a simulation infrastructure (a simulation software) that do not understand the formal language or, if we are performing a complete implementation using a common programming language (like C++ or C#).

It is remarkable that the growing complexity of simulation models implies that the different individuals working to build the model more and more come from different areas with different background and formation. An example would be a social model or an economic model with engineers, economists, sociologists, psychologists and other researchers working on it. This factor, detailed in [5], makes it necessary to establish mechanisms to help in the definition of elements' behavior and mechanisms to transform models between different formalisms. I that sense several authors [6], [5] clearly express the need to establish mechanisms for working with models specified by different formalisms. On [6] it is explained three of the main mechanisms for doing this, (i) Meta-formalism: A formalism that incorporates the different formalisms of the various sub models that makes up the system, (ii) Common formalism: A mechanism that converts all formalisms to a common formalism and (iii) Co-simulation: Independent simulators that work together.

The specification formalism must be easy and clear, so that people who are not used to working with formalisms can quickly understand the model. The formalism must also be powerful, so that the complexity of the model can be represented. However it is difficult to determine if formalism is simpler than another is, because everyone has a personal preference. This paper doesn't want to discuss this, therefore uses the connection between SDL and DEVS to implement an infrastructure that works with the common formalism paradigm.

SDL is one of the graphical languages that one can use to represent a simulation model (other languages can be Petri nets [7], state diagrams, SysML [8], [9] or other language dependent activity diagrams like GPSS diagrams [10], [11] or Arena diagrams). We are using Specification and Description Language because [12]:

1. Allows an unambiguous description of the simulation model.
2. Allows a graphical and a textual description of a system.
3. It is an ISO, and for that the rules that define the grammar are well known and precise. This implies that several tools allow an automatic simulation or code generation from a description of the model.
4. It is not language dependent.
5. Is easy to combine this formalism with an UML formalization of the entire Decision Support System (DSS), since exists concise rules to make this combination.

DEVS is one of the formalisms that one can use to represent a simulation model. We are using that formalism because [13]:

1. Allows an unambiguous description of the simulation model.
2. It is not language dependent.
3. Is widely used and well known in the simulation community.
4. All the systems can be represented using DEVS.

In this paper we present an infrastructure that thanks the relation that exists between SDL and DEVS languages [14], [15] allows to go further and use DEVS models in a distributed simulator named SDLPS [4]. This allows performing simulations of DEVS models combined with SDL models. Also enables to build an automatic representation of DEVS models.

2. SPECIFICATION AND DESCRIPTION LANGUAGE

Specification and Description Language (SDL) is an object-oriented, formal language defined by the International Telecommunication Union – Telecommunication Standardization Sector (ITU–T) on the Recommendation Z.100. The standardization work of ITU dates back to 1865, with the birth of the International Telegraph Union. It became a United Nations specialized agency in 1947, and the International Telegraph and Telephone Consultative Committee (CCITT), (from the French name "Comité Consultatif International Téléphonique et Télégraphique") was created in 1956. It was renamed ITU-T in 1993.

The language is designed to specify complex, event-driven, real-time, interactive applications involving many concurrent activities using discrete signals to enable communication [16], [17], [12].

The definition of the model is based on different components:

- Structure: system, blocks, processes and processes hierarchy.
- Communication: signals, with the parameters and channels that the signals use to travel.
- Behavior: defined through the different processes and procedures.
- Data: based on Abstract Data Types (ADT).
- Inheritances: to describe the relationships between, and specialization of, the model elements.

The language has 4 levels (**Figure 1**), (i) System, (ii) Blocks, (iii) Processes and (iv) Procedures. To know more about the Specification and Description Language please refers to www.sdl-forum.org, [17], [16] or Z.100 recommendation [12].

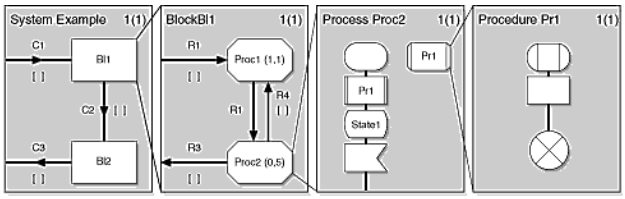
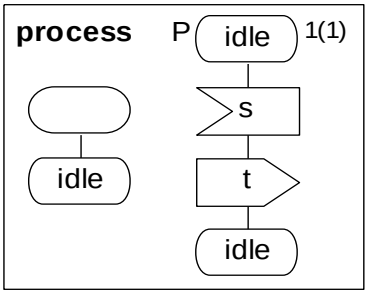


Figure 1: The levels of an SDL model, source: http://www.iec.org/online/tutorials/sdl/topic04.html

3. GRAPHICAL AND NO GRAPHICAL LANGUAGE

SDL have two representations, SDL PR and SDL GR. SDL-PR is conceived to be easily processed by computers, also allows a compact representation of a model, while SDL-GR has some textual elements which are identical to SDL-PR (this is to allow specification of data and signals) it is mainly graphical. In **Figure 2** figure we show an example of a textual and graphical representation of an SDL process.

We are not using the textual version of SDL only for one reason. Some different representations of DEVS based on XML exist. Since we what to allow an automatic transformation from SDL to DEVS, the use of XML simplifies our programming code because now is easy to read and write structured text files that follow the XML syntax, and also, thanks the XSD we can validate the correctness of its syntax. We are using the XML representation for SDL proposed in [4]. Since the more important aspects of an XML file can be represented, and validated, through an XSD file, in the next section some areas of the XSD file are shown.




```
process P;
  start;
  nextstate idle;
  state idle;
    input s;
      output t;
      nextstate idle;
  endstate idle;
endprocess P;
```

Figure 2: textual and graphical SDL representation

3.1. XML representation of an SDL simulation model

This representation was first presented on [15], no modifications has been done from this schema. We next describe the more important elements. For further details, please see [15], or download the complete schema from http://www-eio.upc.es/~pau/index.php?q=node/30.

In **Figure 3** we show the first level of the XSD schema we use to validate the structure of our XML. As is represented, this first level of this schema represents the first level of the Specification and Description Language (system outmost block)

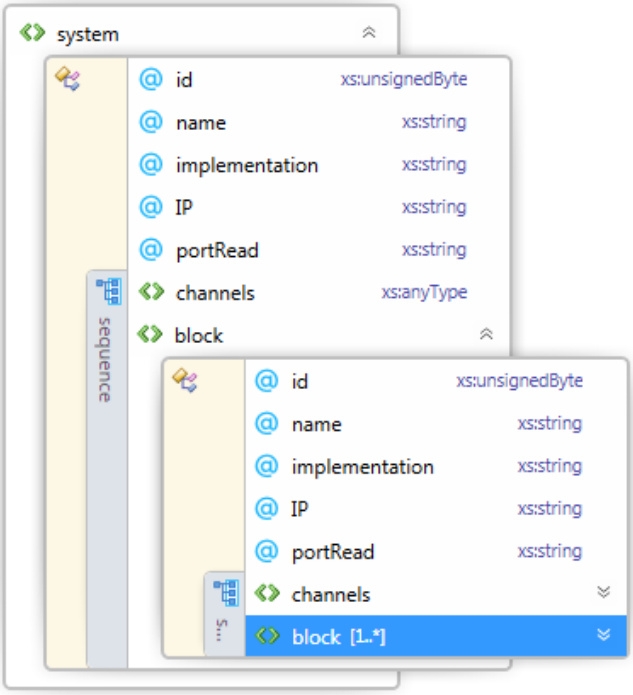


Figure 3. XSD schema, *system* view

In **Figure 4** we shown the process *type* that allows represent an SDL process.

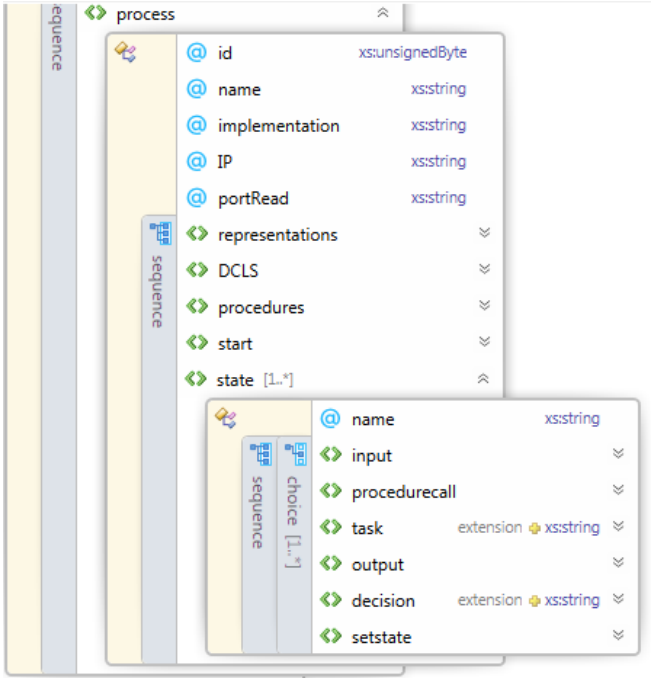


Figure 4. XSD schema, *process* view

4. DEVS FORMALISM

Proposed by Bernard Zeigler in the 70's, the main scope of Discrete Event System Specification (DEVS) is the representation of simulation models. The definition of a model using DEVS formalism is a tuple composed by the elements defined as follows:

$M =< Z, S, Y, \delta_{int}, \delta_{ext}, \lambda, ta > \textbf{where},$
$X: set\ of\ input\ values$
$S: set\ of\ state\ values$
$Y: set\ of\ output\ values$
$\delta_{int}: internal\ transition\ function;\ \delta_{int}: S \to S$
$\delta_{ext}: external\ transition\ function;\ \delta_{ext} \to Q \times X \to S$
$Q = \{(s, e) | s \epsilon S, 0 \leq e \leq ta(s)\}: set\ of\ states$
$e: time\ from\ the\ last\ transition$
$\lambda: S \to Y: output\ function$
$ta: time\ advance\ function$
$ta: S \to R_0^+$

DEVS allows distinguish between an internal and external transition. An internal transition is a kind of transition that doesn't need any external event to be launched. As an example, if in a "t" time, the system reaches state "s", The system remains in this state the during the time defined on a "time advance" function "ta(s)" (if no external event is received). When the time reaches the value defined in the "ta(s)" function an output event is produced (this output is defined on the "λ(s)" function) and the state changes to "s' ". This process is defined in the internal transition s'= δ_{int}(s).

External transitions define the modifications in the model due to the reception of external events. For example, before the model reach the state "s' ", in a time "t", due to his internal transition, an external event, with value x, is processed. In this case the system reach state (s,e) where e<ta(s), the transition follows the external transition function, defined by s'= δ_{ext}(s,e,x), and no exit event is produced.

At this point it is important to underline that "ta(s)" could be any real number, plus 0 and ∞, and:

- If ta(s) is 0, "s" is a *transitory* state.
- If ta(s)=∞, "s" is a *passive* state.

In the next lines we review two examples from [1]. We use these two models to transform them automatically to a SDL specification and to perform using SDLPS [4] a simulation.

4.1. Processor example

This example represents a single processor that receives different jobs. Each job has associated a processing time (represented by a real number). Once the time is over event "ready" is produced. When a new event reach the processor, if this is working with a job, this event is ignored.

The DEVS formalization of this model is:

$$M =< X, S, Y, \delta_{int}, \delta_{ext}, \lambda, ta > \textbf{where},$$
$$X = \{job_1, job_2, .., job_n\}$$
$$S = \{job_1, job_2, .., job_n\} \cup [\emptyset] \times R^+$$
$$Y = \{y(job_1), y(job_2), .., y(job_n)\}$$
$$\delta_{int}(job, \sigma) = (\emptyset, \infty)$$
$$\delta_{ext}(job, \sigma, e, x) = \begin{cases} (x, tp(x) if\ job = \emptyset \\ (job, \sigma - e) otherwise \end{cases}$$
$$\lambda(job, \sigma) = y(job)$$
$$ta(job, \sigma) = \sigma$$

4.2. FIFO Queue example

The queue represented in this example has the following characteristics:

- The queue has infinite capacity.
- Different jobs reach the queue to be stored, while the "ready" signals symbolize the necessity of transmit the first job of the queue.
- The transmission of this job is done through an output event.
- The queue spends 0 time units in the exit delay.

The DEVS model is:

$$M =< X, S, Y, \delta_{int}, \delta_{ext}, \lambda, ta > \textbf{where},$$
$$X = \{job_1, job_2, .., job_n\} \cup \{'ready'\}$$
$$S = \{job_1, job_2, .., job_n\} \cup [\emptyset] \times R^+$$
$$Y = \{y(job_1), y(job_2), .., y(job_n)\}$$
$$\delta_{int}(q \cdot job, \sigma) = (q, \infty)$$
$$\delta_{ext}(job, \sigma, e, x) = \begin{cases} (x \cdot q, \infty) if\ x \in J \\ (q, 0) otherwise \end{cases}$$
$$\lambda(q \cdot job, \sigma) = job$$
$$ta(job, \sigma) = \sigma$$

5. DEVS COUPLED MODELS

DEVS also allows formalize simulation models without describing the behavior for each element belonging the model, due is possible to describe the structural relations that exist among identical elements. These models are named "coupled models".

In DEVS there are two main types of coupled models:

- Modular coupling.
- Non modular coupling.

In modular coupling integration among different model components happens only across entries and exits defined in the components, while in non-modular coupling, interaction is produced across states. The literature established that is possible to pass from one kind of coupling model to the other [5], therefore in present paper we will focus on show the existing relation among SDL formalism and the DEVS modular formalism.

For simplicity the DEVS coupled model used in this paper is DEVS coupled model with ports. In this model a series of input and output ports are described. With this logic is possible to depict the following example **Figure 5**, representing the combination of the two models that have been seen previously (the queue and the processor).

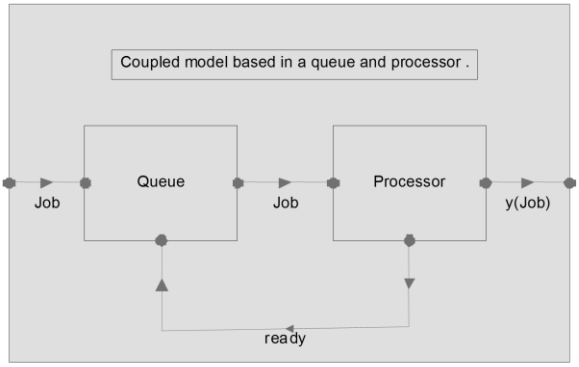


Figure 5. DEVS coupled model.

The coupling model specification for this model is:

N = (X, Y, D, {Md | d Î D}, EIC, EOC, IC, Select), on
X=Jx{inport1}
Y={y(Job) | Job □ J } x { outport1}
D={P,Q}
EIC{(N, inport1), (Q, inport1)}
EOC{(P,outport1), (N, outport1)}
IC{(P, outport2), (Q, outport2)}

6. XML REPRESENTATION OF DEVSMODELS

Our XML representation for DEVS models takes some ideas from the XML representation presented on [18]. In our approach we try to go little further allowing to represent the common structures used in a DEVS model, like programming logic, loops and if-else constructs. Regarding the internal code of the specification we use ANSI C, since it is an ISO standard.

We follow some conventions to represent a DEVS model using XML syntax:

1. All the code needed to fully define the simulation model is defined on the "values" xml section.
2. The initial conditions of the model is defined in the XML as well, using a "value" attribute related to all the variables that defines the state of an atomic DEVS model.

3. Also, to represent the value ∞ used in the passive states we use ‘inf’ literal value.

Some parts of the XML schema we use to represent bot, coupled and an atomic model is represented in **Figure 6**. This schema can be found on http://www-eio.upc.es/~pau/index.php?q=node/30.

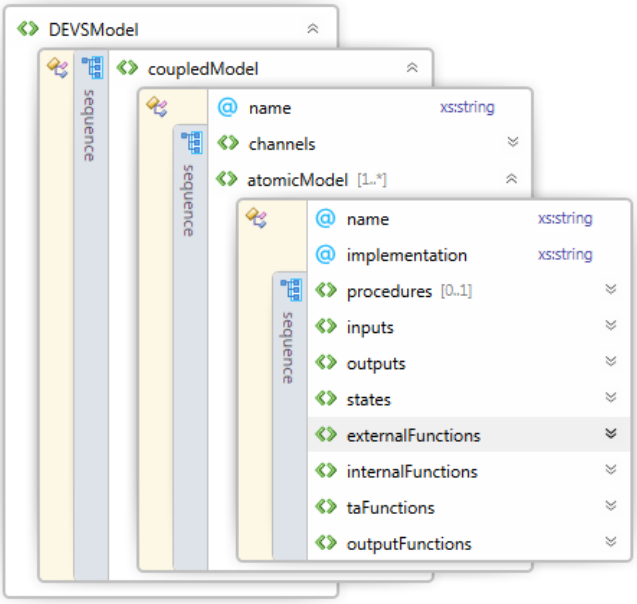


Figure 6. DEVS XML schema

The complete definition of the *DEVSmodel* using XML is show next. On **Figure 7** is represented the whole DEVS model using XML, on **Figure 8** the definition of the *states*, on **Figure 9** the definition of the *input* and the *output* elements, on **Figure 10** the *external functions* and in **Figure 11** the *time advance* and *output functions*.

From this DEVS XML representation we can obtain an equivalent model described using Specification and Description Language, using again XML.

```
<?xml version="1.0" encoding="UTF-8"?>
<DEVSModel xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi
  <coupledModel name="GG1">
    <channels>
      <channel name="in" start="queue" end="processor1" dual="no">
        <event name="job_id"></event>
      </channel>
      <channel name="out" start="processor1" end="queue" dual="no">
        <event name="job_id"></event>
      </channel>
    </channels>
    <atomicModel name="queue" implementation="CDEVSqueue">
      <procedures>...</procedures>
      <inputs>...</inputs>
      <outputs>...</outputs>
      <stateVariables>...</stateVariables>
      <internalFunctions>...</internalFunctions>
      <externalFunctions>...</externalFunctions>
      <taFunctions>...</taFunctions>
      <outputFunctions>...</outputFunctions>
    </atomicModel>
    <atomicModel name="processor1">
      <procedures>...</procedures>
      <inputs>...</inputs>
      <outputs>...</outputs>
      <states>...</states>
      <externalFunctions>...</externalFunctions>
      <internalFunctions>...</internalFunctions>
      <taFunctions>...</taFunctions>
      <outputFunctions>...</outputFunctions>
    </atomicModel>
  </coupledModel>
</DEVSModel>
```

Figure 7. GG1 DEVS model.

The coupled model is

```
<atomicModel name="processor1">
  <procedures>...</procedures>
  <inputs>...</inputs>
  <outputs>...</outputs>
  <states>
    <variables>
      <!--
      The initial confitions of the model is defined on the value.
      -->
      <variable name="job" type="Integer" value="0"></variable>
      <variable name="processing_time" type="Real" value="'inf'"></variable>
      <!--
      In this case it is not needed a literal variable to define the states.
      This is an example of how discrete states con be defined on a model.
      -->
      <variable name="state" type="literal" value="idle|working"></variable>
    </variables>
  </states>
  <externalFunctions>...</externalFunctions>
  <internalFunctions>...</internalFunctions>
  <taFunctions>...</taFunctions>
  <outputFunctions>...</outputFunctions>
</atomicModel>
```

Figure 8. States definition.

```
<atomicModel name="processor1">
  <procedures>...</procedures>
  <inputs>
    <port name="in">
      <signal name="job_id" type="Integer"></signal>
    </port>
  </inputs>
  <outputs>
    <port name="out">
      <signal name="job_id" type="Integer"></signal>
    </port>
  </outputs>
  <states>...</states>
  <externalFunctions>...</externalFunctions>
  <internalFunctions>...</internalFunctions>
  <taFunctions>...</taFunctions>
  <outputFunctions>...</outputFunctions>
</atomicModel>
```

Figure 9. Input and output elements.

```
<atomicModel name="processor1">
  <procedures>...</procedures>
  <inputs>...</inputs>
  <outputs>...</outputs>
  <states>...</states>
  <externalFunctions>
    <function port="in" job="Integer" processing_time="Real" x="">
      <condition>
        <code>job==0</code>
        <body>
          <setstate job="job_id" processing_time="tp(job_id)"></setstat
        </body>
      </condition>
    </function>
  </externalFunctions>
  <internalFunctions>
    <function job="Integer" processing_time="Real">
      <condition>
        <code>true</code>
        <body>
          <setstate job="0" processing_time="'inf'"></setstate>
        </body>
      </condition>
    </function>
  </internalFunctions>
  <taFunctions>...</taFunctions>
  <outputFunctions>...</outputFunctions>
</atomicModel>
</coupledModel>
```

Figure 10. External an internal functions.

```
<atomicModel name="processor1">
  <procedures>...</procedures>
  <inputs>...</inputs>
  <outputs>...</outputs>
  <states>...</states>
  <externalFunctions>...</externalFunctions>
  <internalFunctions>...</internalFunctions>
  <taFunctions>
    <function job="Integer" processing_time="Real">
      <condition>
        <code></code>
        <body>
          <return value="processing_time"></return>
        </body>
      </condition>
    </function>
  </taFunctions>
  <outputFunctions>
    <function job="Integer" processing_time="Real">
      <condition>
        <code></code>
        <body>
          <send port="out" signal="job_id">job;</send>
        </body>
      </condition>
    </function>
  </outputFunctions>
</atomicModel>
```

Figure 11. Time advance and output functions.

7. TRANSFORMING FROM DEVS TO SDL

The transformation algorithm is based on [14]. This allows us to obtain a new XML file that represents a DEVS model. Since the schema used here to represent the SDL model is based on those presented on [15] we only show here the more important aspects of the resulting XML file.

```
<?xml version="1.0"?>
<system id="0" name="GG1" implementation="" IP="" portRead="">
  <channels>...</channels>
  <process id="" name="queue" implementation="" IP="" portRead="">
    <DCLS>...</DCLS>
    <procedures>...</procedures>
    <start>...</start>
    <state name="joblist='Intege">...</state>
  </process>
  <process id="" name="processor1" implementation="" IP="" portRead="">
    <DCLS>...</DCLS>
    <procedures>...</procedures>
    <start>...</start>
    <state name="job='Integer' p">...</state>
  </process>
</system>
```

Figure 12. XML representation of the model.

On **Figure 12** we can see the whole representation of the DEVS model, now transformed to a SDL XML representation. We can see, as we can expect, that the model contains two processes, the *queue* and the *procesor1*.

```
<process id="" name="queue" implementation="" IP="" portRead="">
  <DCLS>...</DCLS>
  <procedures>...</procedures>
  <start>...</start>
  <state name = "joblist='IntegerList' processing_time='Real' ">
    <input id="1" name="INT1"></input>
    <decision id="2" name="" iftrue="3" iffalse="5">true</decision>
    <task id="3" name="">
      processing_time="inf";
      joblist=remove_first_list_element(joblist,job_id);
    </task>
    <output id="4" name="INT1" self="yes" to="" via="">
      <param name="delay" value="processing_time"></param>
      <param name="priority" value="0"></param>
    </output>
    <setstate id="5" name="joblist='IntegerList' processing_time='Real' "></setstate>
    <input id="6" name="EXT1"></input>
    <decision id="7" name="" iftrue="3" iffalse="9">is_job(x)</decision>
    <task id="8" name="">
      joblist=add_new_job_to_list(x,joblist);
      processing_time="inf";
    </task>
    <setstate id="9" name="joblist='IntegerList' processing_time='Real' "></setstate>
    <input id="10" name="EXT1"></input>
    <decision id="11" name="" iftrue="12" iffalse="13">true</decision>
    <task id="12" name="">
      processing_time="0";
      joblist=add_new_job_to_list(x,joblist);
      processing_time="inf";
    </task>
    <setstate id="13" name="joblist='IntegerList' processing_time='Real' "></setstate>
  </state>
</process>
```

Figure 13. Process queue definition.

On **Figure 13** the XML representation using SDL for the DEVS queue element is shown.

8. SIMULATING THE DEVS MODEL ON SDLPS

Regarding the infrastructure used, it is remarkable that SDLPS has been build using C++ and C languages. To establish the communication between all the different distributed elements is used the TCP/IP layer.

On the next figure we can see the DEVS GG1 model on SDLPS.

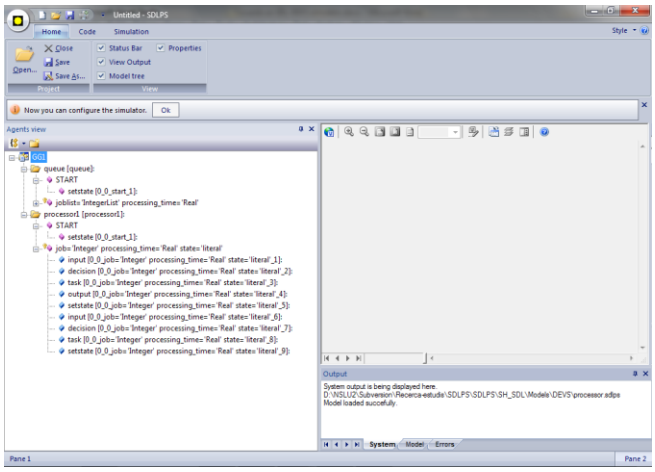


Figure 14. SDLPS system loading the DEVS model.

On the left side we can see the tree that contains all the elements that define the model.

9. CONCLUDING REMARKS

In this paper we present an infrastructure that allows the simulation of DEVS and SDL models. This combination of both languages can be done thanks the XML representation used for DEVS and SDL models. Using this XML representations and the algorithm presented on [15] it is possible to transform a DEVS model to an SDL model allowing its internal use by the SDLPS simulator.

It is remarkable that the SDL model can be a graphical representation for the DEVS models. This representation can be obtained automatically using a Microsoft Visio® Plugin that reads the XML representation of the SDL model. This implies that on SDLPS we can see the structure of the whole model (coupled model) despite of if its behavior is defined using SDL or DEVS. Also, the atomic DEVS models are represented using SDL process diagrams.

This infrastructure is currently used in a production environment in real simulation projects for different well known industries.

The future work is focused in the integration of the Microsoft Visio® plugin with the SDLPS system in order to obtain the representation of the DEVS model right on the simulator.

References

[1] Wainer, G. (2002). CD++: a toolkit to develop DEVS models. *Software, Practice and Experience , 32* (3), pp. 1261-1306.

[2] CINDERELLA SOFTWARE. (2007). *Cinderella SDL*. Retrieved 03 31, 2009, from http://www.cinderella.dk

[3] IBM. (2009). *TELELOGIC*. Retrieved 03 31, 2009, from http://www.telelogic.com/

[4] Fonseca i Casas, P. (2008). SDL distributed simulator. *Winter Simulation Conference 2008*. Miami: INFORMS.

[5] Werner, R. (2000). Structure, flow, change: Towards a social systems simulation methodology. *Annual meeting of Sunbelt XXI: International Sunbelt Social Network Conference*. Budapest, Hungary.

[6] Vangheluwe, H. L. (2000). DEVS as a common denominator for multi-formalism hybrid systems modelling. *IEEE International Symposium on Computer-Aided Control System Design* (pp. 129--134). IEEE Computer Society Press.

[7] Recalde, L., Teruel, E., & Silva, E. (1999). Autonomous continuous P/T systems. Application and Theory of Petri Nets. *Lecture Notes in Computer Science* , 107-126.

[8] OMG SysML. (2010, June). *OMG SysML*. Retrieved December 2010, from OMG: http://www.omg.org/spec/SysML/1.2/

[9] Schönherr, O., & Rose, O. (2009). FIRST STEPS TOWARDS A GENERAL SYSML MODEL FOR DISCRETE PROCESSES IN PRODUCTION SYSTEMS.

In M. D. Rossetti, R. R. Hill, B. Johansson, A. Dunkin, & R. G. Ingalls (Ed.), *Proceedings of the 2009 Winter Simulation Conference*, (pp. 1711 - 1718).

[10] Fonseca i Casas, P., & Casanovas, J. (2009). JGPSS, an Open Source GPSS Framework to Teach Simulation. *Procediings of the Winter Simulation Conference 2009.* Austin.

[11] Crain, R. C., & Henriksen, J. O. (1999). Simulation using GPSS/H. In P. A. Farrington, H. B. Nembhard, D. T. Sturrock, & G. W. Evans (Ed.), *Proceedings of the 1999 Winter Simulation Conference*, (pp. 182-187).

[12] Telecommunication standardization sector of ITU. (1999). *Specification and Description Language (SDL).* Retrieved April 2008, from Series Z: Languages and general software aspects for telecommunication systems.: http://www.itu.int/ITU-T/studygroups/com17/languages/index.html

[13] Zeigler, b., Praehofer, h., & Kim, d. (2000). *Theory of Modeling and Simulation.* Academic Press.

[14] Fonseca i Casas, P., & Casanovas Garcia, J. (2005). Using SDL diagrams in a DEVS specification. In G. Tonella (Ed.), *The Fifth IASTED International conference on Modeling Simulation and Optimization.* IASTED.

[15] Fonseca i Casas, P. (2009). Towards an automatic transformation from a DEVS to a SDL specification. *Procediings of the 2009 Summer Simulation Multiconference.* Istanbul, Turkey.

[16] Reed, R. (2000). SDL-2000 form New Millenium Systems. *Telektronikk 4.2000* , 20-35.

[17] *SDL Tutorial*. (n.d.). Retrieved January 2009, from IEC International Enginyeriing Consortium: http://www.iec.org/online/tutorials/sdl/

[18] Risco-Martín, J., Mittal, S., López-Peña, M., & De la Cruz, J. (2007). A W3C XML Schema for DEVS Scenarios. *Spring Simulation Multiconference 2007, DEVS Symposium*, pp. 279-286. Norfork, Virginia.