# Uncovering DEVS Simulation Behaviour Throughout The Open Provenance Model

**Alejandro Moreno[1], José L. Risco-Martín[2], Joaquín Aranda[1]**

**[1]Departamento de Informática y Automática**
**Escuela Técnica Superior de Informatica**
**Universidad Nacional de Educación a Distancia (UNED)**
**28040 Madrid, Spain**
**amoreno@bec.uned.es, jaranda@dia.uned.es**

**[2]Departamento de Arquitectura de Computadores y Automática**
**Facultad de Informática**
**Universidad Complutense de Madrid (UCM)**
**28040 Madrid, Spain**
**jlrisco@dacya.ucm.es**

**Keywords:** Discrete event simulation, DEVS, provenance-aware systems, OPM, interoperability

## Abstract

DEVS models are getting to be more and more sophisticated due to large scale objectives such as modeling complex physical continuous systems. The assurance of the validity and quality of simulation data handling and analysis may lead to a great challenge. Thus, a need of providing insight of DEVS simulation performance can arise. In particular, information about simulation processes workflow and internal data values and management. Provenance-aware systems are designed to satisfy those needs by documenting processes and casual dependencies. Provenance is a critical concept in scientific workflows, since it allows scientists to understand the origin of their results, to repeat their experiments, and to validate the processes that were used to derive data products. Recently, the provenance community agreed that a “Provenance Challenge” should be set to compare and understand existing approaches and updated the *Open Provenance Model (OPM)*. We introduce provenance documentation technology to uncover DEVS models behaviours during a simulation. In addition, we apply the OPM formalization due to interoperability reasons and provide support to the *Third Provenance Challenge*.

## 1. INTRODUCTION

DEVS M&S is a modular and hierarchical formalism for modeling and simulation of discrete event systems, continuous state systems and hybrid continuous state systems [1]. Because of the modular and hierarchical modeling views as well as its simulation analysis capability, DEVS formalism is used in many application of engineering such as hardware and software design or control systems. These modeling views are getting to be more and more sophisticated due to large scale objectives such as modeling of complex dynamic continuous systems. As a result, simulation analysis turns out even more confusing. The analysis and validation of a DEVS model simulation behaviour becomes a great challenge. DEVS modeling optimization such as finding bottlenecks or useless and harmful modeling elements may be a difficult task. Therefore, the need of providing insight of DEVS simulation performance arises, in particular, information about simulation processes, workflow, internal data values, and data management. This necessity emphasizes if the simulation framework applies a real time and non coordinated architecture, since the degree of unawareness of simulation-based behaviour is extremely high due to the self-dependent nature of each simulator. Provenance-aware systems intend to satisfy those needs by documenting processes and casual dependencies. Provenance is a critical concept in scientific workflows, since it allows scientists to understand the origin of their results, to repeat their experiments, and to validate the processes that were used to derive data products. Besides, recently the provenance community [2] decided that it needs to understand the different representations used for provenance, common aspects, and the reasons for their differences. As a result, the community agreed that a “Provenance Challenge” should be set to compare and understand existing approaches and updated the *Open Provenance Model (OPM)* [3]. The OPM is a community-driven data model for Provenance that is designed to support interoperability of provenance technologies. Summarizing, DEVS simulation analysis is becoming more complicated, therefore we introduce provenance documentation technology to uncover DEVS models behaviours during a simulation. And, in addition, we apply the OPM formalization due to interoperability reasons and provide support to the *Third Provenance Challenge* [4].

This document is organized as follows. Section 2 collects some relevant aspects of the two formalisms DEVS and Provenance combined in the overall approach. Section 3 presents the basics of provenance documentation process embedded in the simulation execution. In section 4 we discuss the results obtained by an experimental case study. After, in section 5 we compare our proposal with other provenance-aware systems oriented to scientific simulations. Finally, in section 6 some conclusions are drawn and some future research lines are presented.

## 2. BACKGROUND

### 2.1. DEVS

The Discrete Event System Specification is a general formalism for discrete event system modeling based on set theory [1]. It allows representing any system by three sets and five functions: input set ($X$), output set ($Y$), state set ($S$), external transition function ($\delta_{ext}$), internal transition function ($\delta_{int}$), confluent function ($\delta_{con}$), output function ($\lambda$), and time advanced function ($ta$). DEVS formalism provides the framework for information modeling which gives several advantages to analyze and design complex systems: completeness, verifiability, extensibility, and maintainability. DEVS can also approximate continuous systems using numerical integration methods. Thus, simulation tools based on DEVS are potentially more general than other tools including continuous simulation tools [5].

DEVS defines system behaviour as well as system structure. System behaviour in DEVS formalism is described using input and output events as well as states. To this end, DEVS has two kind of models to represent systems: atomic model and coupled model. The atomic model is the irreducible model definition that specifies the behaviour for any modeled entity. The coupled model is the aggregation/composition of two or more atomic and coupled models connected by explicit connections between ports. The coupled model can itself be a part of component in a larger coupled model system giving rise to a hierarchical DEVS model construction. The top-level coupled model is usually called the *root coupled model*.

There are varied libraries for expressing DEVS models across the globe, such as DEVSJAVA [6], DEVS/C++ [6], CD++ [7], xDEVS [8], etc., and all of them have efficient implementations for executing the DEVS protocol. Plus, they all manage the simulation time, coordinates event schedules, and supply a library for simulation, a graphical user interface to view the results, and other utilities.

### 2.2. Provenance

The term "provenance" is used to refer to both the concept of how some item came to be as it is, i.e. the process that led to that item, and the representation of that process. The latter can be obtained as the result of a query over a set of documentation regarding past application processes.

The Open Provenance Model (OPM)[3] is a community-driven data model for Provenance that is designed to support interoperability of provenance technologies. Underpinning OPM, is a notion of directed acyclic graph, used to represent data products and processes involved in past computations, and causal dependencies between these. The Open Provenance Model was derived following two "Provenance Challenges" within international and multi-disciplinary activities trying to investigate how to exchange information between multiple systems supporting provenance and how to query it. The core data structure that OPM supports is an OPM graph that consists of nodes and edges. Nodes are physical objects or data products, referred to as artifacts, that are represented as ellipsis, whereas activities or processes that produce and consume artifacts, are represented by rectangles. Edges in OPM graphs are directional, from effect to cause, explaining how an effect resulted from a cause. OPM Graphs are meant to be read from bottom to top, explaining how effects are repeatedly derived from causes. An example of OPM graph [3] is displayed in Figure 1, illustrating how a cake resulted from a baking process and was made of several ingredients.

In OPM there are four types of edges: *Generated by*: an artifact was generated by a process, if the process had to initiate its execution, for the artifact to be produced, *Used*: a process used an artifact, if the artifact had to be present for the process to be able to complete, *Derived From*: an artifact was derived from another artifact if the latter had to exist, for the former to be generated, *Informed by*: a process was informed by another process if the latter had to initiate its execution for the former to complete.

All edges are characterized by the past tense, to denote that they refer to a past execution. Since the aim of OPM graphs is to represent executions that took place in the past, this fact is quite important. Note that artifacts are instantaneous pieces of data or state, whereas processes have a duration. In fact, OPM edges have a more precise definition, based on event ordering. An artifact was generated by a process, if the process had to initiate its execution for the artifact to be produced.

From the example shown at Figure 1, we infer that an artifact cake is *Generated By* the process *bake* that *uses* artifacts *100g of butter*, *2 eggs*, *100g of sugar* and *100g of flour*, and moreover, a *cake* is *derived from* the former artifacts.

## 3. PROVENANCE-AWARE DEVS M&S

### 3.1. Methodology

This section analyzes the set of rules, procedures and methods employed to fulfill the provenance principles. The following methodology refers to anything and everything that can

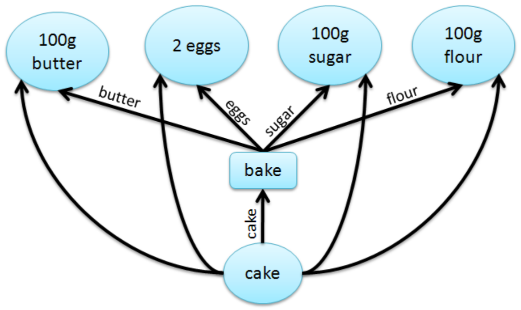

**Figure 1.** Baking Cake

be encapsulated for a series of processes, activities and tasks (who, what, where, when, and why).

The output of a simulation execution should provide a document structured in accordance to the OPM for Provenance in regard of any customized user model outputs. In order to make our previous DEVS simulation application provenance-aware, we introduce the PrIMe [9] methodology. The steps through PrIMe are as follows:

- Phase 1
  - Step 1.1: Provenance use case analysis.
  - Step 1.2: Identify use case information items.
- Phase 2
  - Step 2.1: Identify application actors.
  - Step 2.2: Map out actor interactions.
  - Step 2.3: Identify knowledgeable actors.
- Phase 3
  - Step 3.1: Introduce application adaptations.

For an instance, making petty distinctions may look like a good way out, but the output provenance model of a simulation might be overloaded with redundant information and may be difficult to figure out. On the other hand, if the documentation system does not capture enough detail, some relevant workflow might miss. The level of granularity must move in some kind of equilibrium between both.

## 3.2. Provenance Middleware

At the moment, there are just a few provenance tools that provide techniques to adapt an application to a provenance-aware system. Some of the participating teams from the Third Provenance Challenge [4] facilitate the instruments they applied over their challenge proposal. The choice of which technology to apply is not an easy task. The selection must take into account several factors: (a) feasibility, (b) adequacy, (c) interoperability, (d) expertise.

These factors intend to back up one unique purpose, the provenance documentation of a DEVS simulation application. On the one hand, the University of Southampton has achieved a great deal of research over the past years concerning provenance-aware applications within two projects: The EU Provenance Project [10] and the Provenance Aware Service Oriented Arquitecture (PASOA) Project [11]. Although, these provenance projects have a great support and expertise, due to their complexity, implementing them for an initial solution fattens up the task. On the other hand, Tupelo [12] is the most widely used platform for the Third Provenance Challenge [4]. Tupelo is a semantic content repository framework being developed at the NCSA to record provenance records. In addition to a Java OPM binding, Tupelo provides a dedicated provenance Java API which allows to weave our code to record provenance records directly into the workflow code. The use of this provenance API records OPM compliant records while abstracting away from the user the means by which these observations are recorded. Thus, we applied Tupelo fulfilling the principles elucidated above.

## 3.3. Validation & Visualization

The Open Provenance Toolbox [13] provides a series of tools to manipulate OPM graphs from the command line. This tools are listed below.

- OPM RDF to XML: converts OPM graphs represented as *Resource Description Framework (RDF)* into OPM graphs represented as *Extensible Markup Language (XML)*.
- OPM XML to RDF: converts OPM graphs represented in XML into OPM graphs represented as XML/RDF.
- OPM XML validation: validates OPM graphs represented in the xml format.
- OPM to DOT: converts OPM graphs into plain text graph description language DOT files and outputs a graphical view in PDF format.

## 3.4. Provenance Workflow Documentation

This section presents the workflow of the simulation to be documented and defines the granularity level as well. Details the OPM elements, artifacts, agents, processes, roles and edges specified at any DEVS modeling simulation documentacion procedure. The documented workflow keeps track of DEVS functions and the information streaming (states, inputs and outputs) among them and between simulators. DEVS formalisation functions documented as OPM processes are: $\delta_{int}$: internal transition function, $\delta_{ext}$: external transition function, $\delta_{con}$: confluence transition function, $\lambda$: output function, *coupling*: function that maps output ports to input

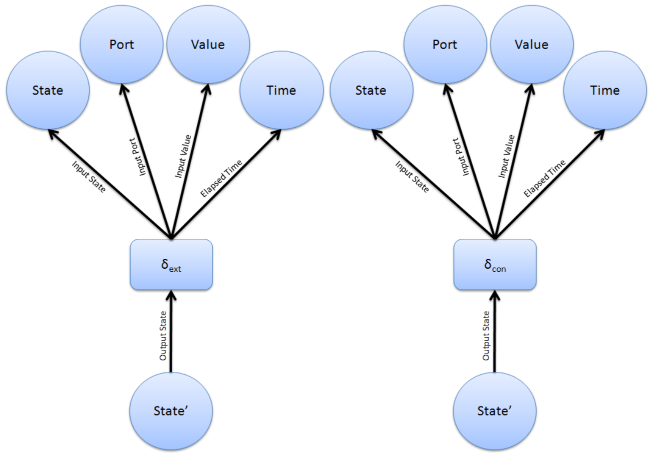

**Figure 2.** OPM DEVS Simulation Elements (1)

ports. Whereas the incoming and outcoming data artifacts of these processes are: State: gathers $\sigma$ (time to next internal event) and phase (status), Port: Input or output port where the messages arrive or depart, Value: Output message value from a model, Time: Elapsed time between events.

In order to link the data artifacts formerly stated, we define the following edges based on event ordering classified within the detailed roles.

- Generated By: $(process \Rightarrow artifact)$
  - Output Port: $\lambda \Rightarrow Port$
  - Output Value: $\lambda \Rightarrow Value$
  - Output State: $\delta_{int}, \delta_{ext}, \delta_{con} \Rightarrow State$
  - Output Port To: $Coupling \Rightarrow Port$
- Used By: $(artifact \Rightarrow process)$
  - Input Port: $Port \Rightarrow \delta_{ext}, \delta_{con}$
  - Elapsed Time: $State \Rightarrow \delta_{ext}, \delta_{con}$
  - Input Value: $State \Rightarrow \delta_{ext}, \delta_{con}$
  - Input State: $State \Rightarrow \delta_{int}, \delta_{ext}, \delta_{con}$
  - input Port From: $Port \Rightarrow Coupling$

In addition to this, we generate a time aware provenance graph. Hence, these edges are supported by time annotations that consist of the observed time of an iteration. Two timestamps (no earlier & no later) that point out a time interval for the given iteration. This time interval may be one timestamp, as if both timestamps *no earlier* and *no later* where the same.

Figures 2 and 3 reveal each possible atomic iteration in our DEVS simulation platform in accordance with the upper mentioned OPM customized elements. These documented process iterations may belong to any of the DEVS model being simulated as part of the overall execution. This ownership

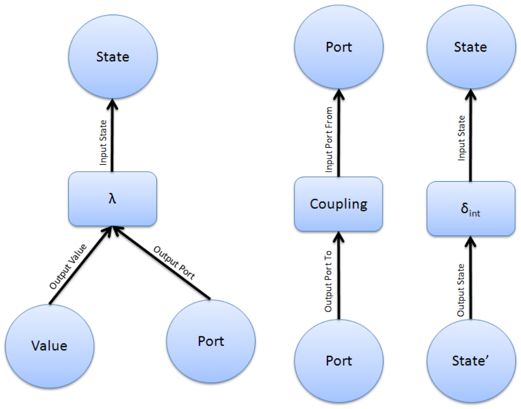

**Figure 3.** OPM DEVS Simulation Elements (2)

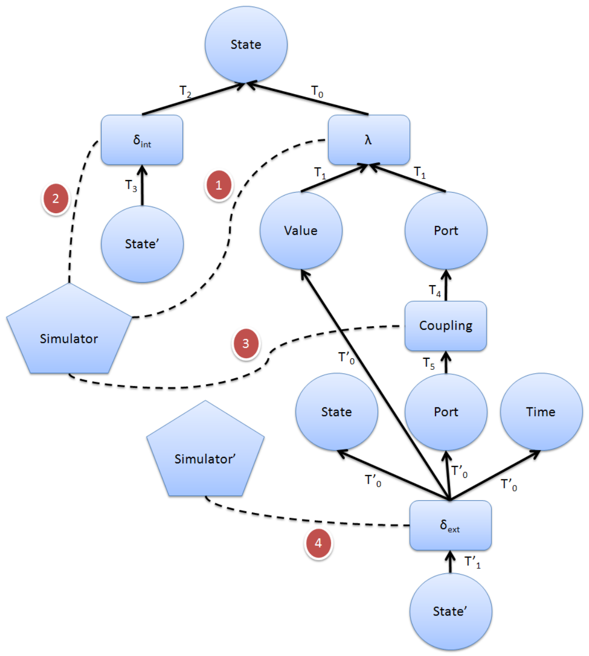

**Figure 4.** OPM DEVS Simulation trace

is expressed by means of the "Controlled By" edge stipulated at the OPM definition. Figure 4 illustrates an example of a brief simulation trace documented with OPM.

The following list enumerates the OPM simulation documentation trace represented by Figure 4.

1. A simulator executes the output function $\lambda$ of the corresponding model, with the current model state as an input argument, and retrieves an output value and an output port where that value is located. In terms of OPM, a state is *used by* $\lambda$ and the output value and the output port is *generated by* $\lambda$.

2. That same simulator executes the internal transition function $\delta_{int}$ of the respective model, with the current

model state as an input argument, and retrieves the new state of the model. In terms of OPM, a state is *used by* $\delta_{int}$ and the output state is *generated by* $\delta_{int}$. Due to DEVS formalism specification this iteration is always executed after the output function $\lambda$, in other words, $T_1 < T_2$.

3. Again, the same simulator calls the *coupling* function which maps an output port of a model to the input port a model. In terms of OPM, an output port is *used by* the *coupling* process and an input port is *generated by* the *coupling* process.

4. Another simulator receives an external output value and an input port previously generated, along with the elapsed time throughout the external transition function $\delta_{ext}$. This iteration means that there exists a coupling between the model being simulated by the previous simulator onwards with the model being simulated by the current simulator. In terms of OPM, an input port, input state, input value and elapsed time are *used by* $\delta_{ext}$ and the output state is *generated by* $\delta_{int}$.

## 3.5. Common Queries

In general, a query is a form of questioning in a line of inquiry. In this sections we refer to the questions we intend to ask to a provenance graph resultant from our DEVS simulation platform. That is, common questions that respond to simulation aspects purely based on DEVS simulation formalism with different objectives, such as validating the simulation performance, or even to elucidate a DEVS component form its behaviour.

The following list enumerates only a small group of the overall possible queries for a DEVS based OPM graph to denote the capability of provenance queries.

1. Does every output function always triggers before the internal event derived from the same state?

2. What is the maximum time model “X” has been waiting for an input?

3. Is value “Z” generated by model “X”?

4. What is the value received by model “X” in port “W” before generating value “Z” at port “Y”?

5. At what time did model “X” reached state “S”?

6. What is the average $\sigma$ value (time for next internal event defined in the state) of model “X”?

7. Which models received value “Z” generated by model “X” at port “Y” after “N” internal transition events?

8. What is the sequence of processes that generated value “Z” as an input to model “X”?

9. Did value “Z” generated by model “X” derived an input at model “W”?

10. Does any model always receives a value at some port with $\sigma = \infty$, moves to a state with $\sigma \neq \infty$, triggers an output function followed by the internal transition and finally passivates ($\sigma = \infty$)?

Some queries provide validation methods of the DEVS formalization within our simulation platform. While others might want to keep track of processes or artifacts derived from the generation of an artifact, the execution of a process, or ensure that a value is never generated, received or vice-versa. Or even try to discover the nature of a model, weather it is only an element in the overall root model that injects a delay, or it’s simply a proportional block, or acts as memoryless Mealy machine where its outputs depends on its inputs.

In addition to this, we define 3 categories of provenance queries: Queries based on the OPM graph topology, Queries based on data values, Queries based on both concepts.

## 4. CASE STUDY - PROCESSOR MODEL

Figure 5 depicts the ef-p model, which is a simple DEVS coupled model consisting of three atomic models and one coupled model. The generator atomic model generates job-messages at fixed time intervals and sends them via the “out” port. The transducer atomic model accepts job-messages from the generator at its “arrived” port and remembers their arrival time instances. It also accepts job-messages at the “solved” port. When a message arrives at the “solved” port, the transducer matches this job with the previous job that had arrived on the “arrived” port earlier and calculates their time difference. Together, these two atomic models form an experimental frame coupled model. The experimental frame sends the generators job messages on the “out” port and forwards the messages received on its “in” port to the transducers “solved” port. The transducer observes the response (in this case the turnaround time) of messages that are injected into an observed system. The observed system in this case is the processor atomic model. A processor accepts jobs at its “in” port and sends them via “out” port again after some finite, but non-zero time period. If the processor is busy when a new job arrives, the processor discards it. Finally the transducer stops the generation of jobs by sending any event from its “out” port to the “stop” port at the generator.

This section illustrates the OPM documentation results of the aforementioned experiment. The OPM generated by each simulation is initially an RDF type document in accordance with Tupelo’s OPM binding as an OWL ontology. In order

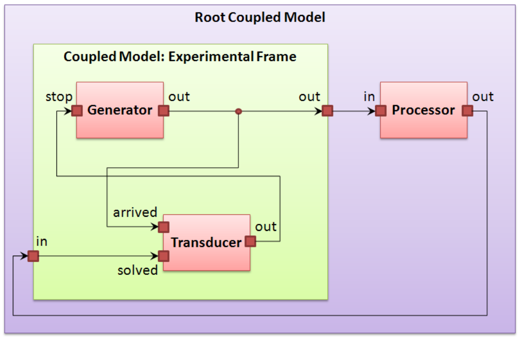

**Figure 5.** Experimental Frame & Processor DEVS model

to get a graphical output, we apply the OPM toolbox. Execute the OPM RDF to XML tool to convert the outputted document based on Tupelo's OPM bindings to a document based on the Open Provenance Model XML Schema. Finally, running the OPM to DOT tool to obtain a graphical representation of our experiment as a plain text DOT file and a graphical view in PDF format shown below. Figure 6 represents the graphical output from this experiment. However, the resultant graphical representation exceeds this document figure layer output maximum size, that's why we provide three Figures (7, 8, 9) that zoom in to Figure 6 accompanied by an explanation. This OPM representation is analogous with section 3 where boxes represent a process iteration ($\delta_{ext}$, $\delta_{int}$, $\delta_{con}$, $\lambda$ or coupling) and ellipsis represent data artifacts (state, port , value or time). Each node is identified by a name that details the DEVS model it belongs to, the process being executed or the artifact being consumed or generated, and the interaction index.

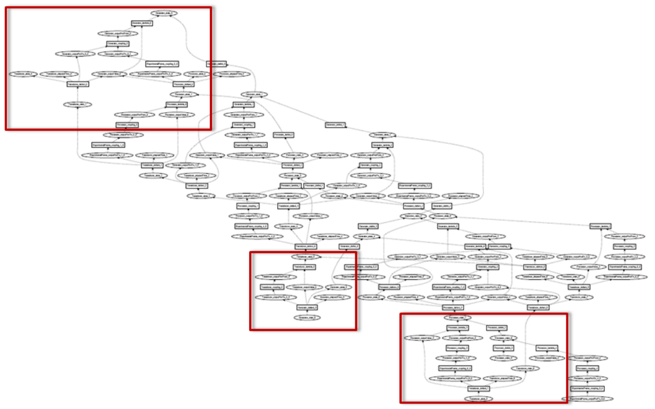

**Figure 6.** Experiment

Figure 7 gathers the left upper side of Figure 6. This view represents the beginning of the simulation, where all models start from their initial states (*Generator_state_0*, *Transducer_state_0* and *Processor_state_0*). As seen on the figure, the Generator model triggers the output function $\lambda$ (*Generator_lambda_0*) generating a job (*Generator_outputValue_0*) as an output message throughout an output port (*Generator_outputPortFrom_0*) mapped to (*Generator_coupling_0*) the destination model inports (*Generator_outputPort_0_0* and *Generator_outputPort_0_1*). These artifacts along with the initial states (*Transducer_state_0* and *Processor_state_0*) are consumed by the external functions of the Transducer (*Transducer_deltext_0*) and Processor (*Processor_deltext_0*). These processes generate the according new state of the Transducer (*Transducer_state_1*) and the Processor (*Processor_state_1*).

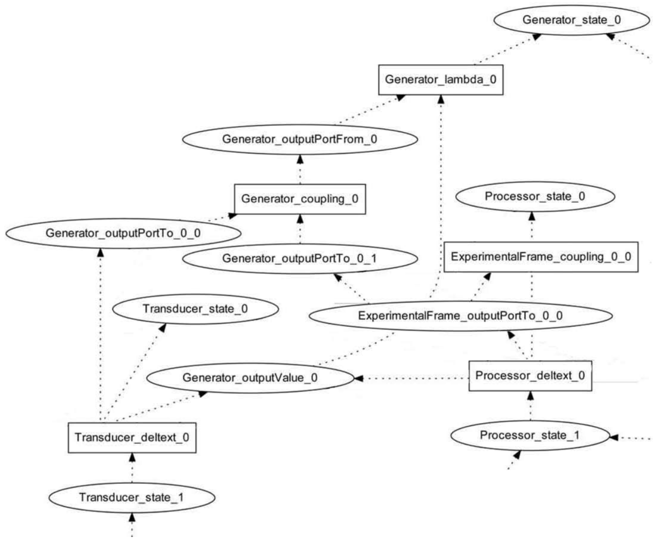

**Figure 7.** Experiment (1)

Figure 8 gathers the middle down side of Figure 6. This view illustrates how the Transducer triggers its $\lambda$ function (*Transducer_lambda_0*) and outputs a message (*Transducer_outputValue_0*) consumed by the Generator external function (*Generator_deltext_0*) and stops the job generation process since the state generated (*Generator_state_6*) is not consumed by any process.

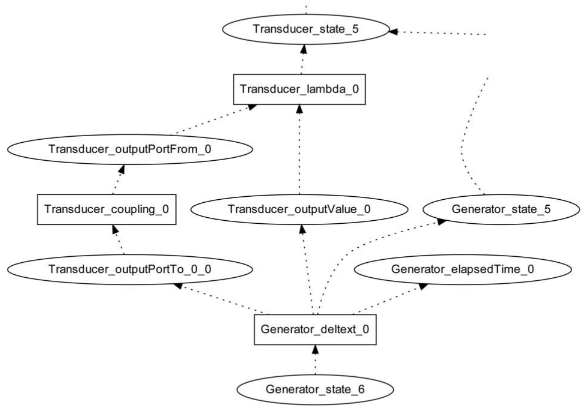

**Figure 8.** Experiment (2)

Figure 9 gathers the lower right side of Figure 6. This view illustrates how the Processor triggers its λ output function (*Processor_lambda_3*) because it has finished the job previously assigned by the Generator. And sends a message (*Processor_outputValue_3*) to the Transducer external function (*Transducer_deltext_7*) reporting the finalization of the previous job. However, the node path that generated the Transducer previous state (*Transducer_state_8*) points out that the Transducer has already triggered its λ function (*Transducer_lambda_0*) reaching its internal time event and therefore the job is not scored.

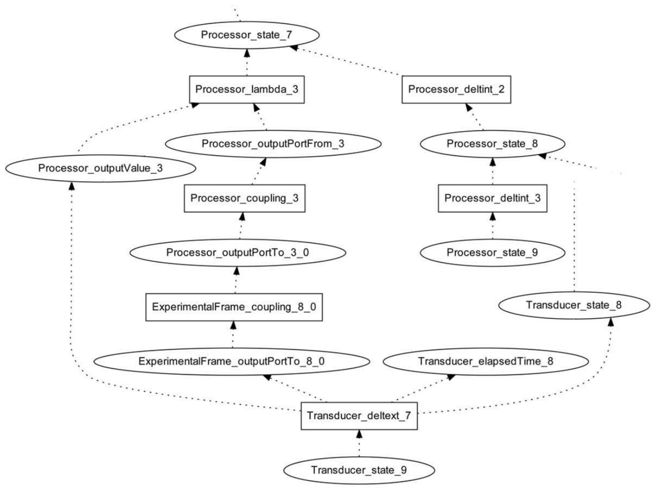

**Figure 9.** Experiment (3)

Extracting this information from graphical views may be useful for some simple and quick consultations. But when we encounter a greater challenge such as how to infer relevant and meaningful data from our provenance documentation, it is necessary to move on to Provenance queries.

All the information extracted from the OPM graphs can be done automatically throughout Provenance queries. Furthermore, these queries may involve a degree of sophistication unable to be managed by a simple graphical perception. They are capable of moving throughout the generated OPM paths, check data values, analyze topology, extract process execution timestamps, etc ... Combining them we may infer information of a simulation behaviour that is being kept hidden or located at the background.

Unfortunately, the Open Provenance community has not identified yet a standard query language for OPM graphs. Nevertheless, common patterns of provenance queries begin to emerge, and API's are being designed to support them directly [3]. Tupelo provides a set of API's to gather information from OPM graphs, but unlikely, they differ much from a query language. However, the initial output of the simulation is formatted in RDF, and with a more general purpose, there exist RDF query languages such as *SPARQL Protocol and RDF Query Language*[14]. SPARQL allows for a query to consist of triple patterns, conjunctions, disjunctions, and optional patterns. Implementations of SPARQL for multiple programming languages exist, therefore it can be a convenient mean to extract information, although each query must be adapted and customized for the OPM notation.

## 5. RELATED WORK

A similar research but with a different view point is the work presented by Guy K. Kloss and Andreas Schreiber [15] as part of the Grid Provenance project [10]. They announce the lack of mechanisms to trace the generation of simulation results and the underlying processes. And seem to demonstrate how trust and confidence in simulation results can be achieved by provenance-aware applications. However, in distinction from our approach, they focus on results of simulations accordant to a set of given parameters, simulation characteristics and setup, instead of the internal processes carried on by each simulated model while performing a simulation. In other terms, most of the work done in provenance can be classified in several categories: provenance-aware storage systems, architectures for provenance systems, application based, and methodologies. Provenance-aware storage systems and architectures research areas center on evolving provenance technology. On the contrary, we search for the most suitable provenance technology for our DEVS simulator that must be capable of dealing with the interoperable Open Provenance Model, instead of building up a provenance system from the bottom.

## 6. CONCLUSION AND FUTURE WORK

In this document, we presented a Provenance-aware DEVS simulation platform. This arrangement was initially motivated by the need of uncovering complex DEVS models simulation behaviour and to assist the validation of a non coordinated distributed DEVS simulation framework meant to work with real scenarios along with simulated models. Adding Provenance to DEVS simulations by means of the Open Provenance Model provides descriptive documentation of a simulation. Documentation that represents data products and processes involved in past computations, and causal dependencies between these. This dependencies are seen as relations or waypoints of possible paths. Thus, a DEVS simulation may benefit from provenance in several manners: Keeping track of simulation models outputs and inputs, and internal behaviour based on inputs and time events, Discovering paths or process executions that made possible the generation of an output or the triggering of other process, Recognizing models based on their internal event iterations ordering, Validating data values of incoming or outcoming messages, Validating in runtime a simulation in accordance with the provenance path that is being followed.

All this information can be queried over an OPM graph resultant from a DEVS simulation. The outputted information of a DEVS simulation is data with dependencies. There are already various scientific disciplines such as machine learning, data and process mining, and pattern matching that are concerned with the design and development of algorithms that allow computers to automatically learn to recognize complex patterns. Hence, assembling both worlds looks like a promising research area. As an example, the Workflow Patterns initiative [16] and the process mining group from the Eindhoven University of Technology. These research areas may supply the necessary techniques to validate and verify DEVS models behaviour over a simulation.

The provenance-based method proposed in this document seems to work on very detailed information from the DEVS simulation. Hence, a mayor issue appears: Can this approach scale to larger simulations where hundreds/thousands of models exist in the simulation? Yes, although with the initial approach the simulation performance should decline and the provenance documentation output complexity will increase. But the provenance-based method can be modified to document only certain DEVS models of the overall simulation and specific DEVS simulation processes. Whereas the complexity issue is not alarming because an end-user application that works over DEVS provenance should scale up to different levels of abstraction according to the final user qualification and research purpose.

## 7. ACKNOWLEDGEMENT

The contributions of authors José L. Risco-Martín, Alejandro Moreno and Joaquín Aranda in this paper are sustained by the Spanish Ministry of Education and Science projects DPI2006-15661-C02-01 and DPI2006-15661-C02-02.

## REFERENCES

[1] B. P. Zeigler, T. Kim, and H. Praehofer, *Theory of Modeling and Simulation: Integrating Discrete Event and Continuous Complex Dynamic Systems*. Academic Press, 2000.

[2] R. Bose, I. Foster, and L. Moreau, "Report on the international provenance and annotation workshop," Provenance Community, Tech. Rep., 2006.

[3] L. Moreau, N. Kwasnikowska, and J. V. den Bussche, "The foundations of the open provenance model," University of Southampton, Tech. Rep., 2009.

[4] "Provenance challenge." [Online]. Available: http://twiki.ipaw.info/bin/view/Challenge/WebHome

[5] E. Kofman, "Discrete event simulation of hybrid systems," *SIAM Journal on Scientific Computing*, vol. 25, no. 5, pp. 1771–1797, 2004.

[6] Arizona Center of Integrative M&S (ACIMS), "Arizona Center of Integrative M&S (ACIMS)," http://www.acims.arizona.edu, 2008.

[7] G. Wainer, "CD++: a toolkit to develop devs models," *Softw. Pract. Exper.*, vol. 32, no. 13, pp. 1261–1306, 2002.

[8] José L. Risco Martín and J. M. Cruz, "xDEVS: DEVS java API," http://www.dacya.ucm.es/jlrisco.

[9] S. Munroe, S. Miles, L. Moreau, and J. Vázquez-Salceda, "Prime: a software engineering methodology for developing provenance-aware applications," in *Proceedings of the 6th international workshop on Software engineering and middleware*. ACM, 2006, pp. 39 –46.

[10] University of Southampton, "The provenance project." [Online]. Available: http://www.gridprovenance.org/

[11] University of Southampton, "Provenance aware service oriented architecture project." [Online]. Available: http://www.pasoa.org/

[12] NCSA, "Tupelo semantic content repository." [Online]. Available: http://tupeloproject.ncsa.uiuc.edu/

[13] L. Moreau, "Open provenance toolbox." [Online]. Available: http://openprovenance.org/

[14] W3C, "Sparql query language for rdf," W3C Recommendation, January 2008. [Online]. Available: http://www.w3.org/TR/rdf-sparql-query/

[15] G. K. Kloss and A. Schreiber, *Provenance and Annotation of Data*. IPAW 2006, 2006, ch. Provenance Implementation in a Scientific Simulation Environment, pp. 37–45.

[16] Eindhoven University of Technology and Queensland University of Technology, "Workflow patterns initiative." [Online]. Available: http://www.workflowpatterns.com