Formal Verification of DEVS Simulation: Web Search Engine Model Case Study

Alonso Inostrosa-Psijas
Universidad de Santiago
Santiago, Chile
alonso.inostrosa@usach.cl

Veronica Gil-Costa
Universidad Nacional de San Luis
San Luis, Argentina
gvcosta@unsl.edu.ar

Gabriel Wainer
Carleton University
Ottawa, Canada
gwainer@sce.carleton.ca

Mauricio Marín
CeBiB, DIINF, Universidad de Santiago
Santiago, Chile
mauricio.marin@usach.cl

ABSTRACT
Web search engines (WSE) are complex and highly optimized systems operating over large clusters of processors which manage high and dynamic and unpredictable user query bursts. The modeling, simulation and formal verification of shush systems is a challenge task which includes to manage user behavior and hardware costs. In this paper, we propose a WSE modeled with DEVS to be efficiently deployed on a distributed cluster of computers. The proposed model aims to reduce the communication overhead introduced by the message passage among the different hierarchical components of the DEVS model. This model is formally verified by using an equivalent Timed Automata model.

Author Keywords
Web search engines; DEVS simulation; timed automata.

ACM Classification Keywords
I.6.4 SIMULATION AND MODELING (Model Validation and Analysis): Verification.

INTRODUCTION

In recent years, we have faced a tremendous growth in the number of distributed applications on the Internet. One of the most complex applications includes large scale Web Search Engines (WSE), which are the backbone of existing web search engines. Building such engines is a very complex task, and traditional analysis methods where systems are designed directly at the low hardware and software levels are fast becoming infeasible due to the increasing complexity and market demands [1]. In particular, modeling such systems and providing formal analysis using formal analytical techniques is unfeasible due to the complexity of the problems at hand. Instead, construction of system models and their analysis through simulation reduces both end costs and risks, while enhancing system capabilities and improving the quality of the final products. M&S let users experiment with "virtual" systems, allowing them to explore changes, and test dynamic conditions in a risk-free environment. Modeling complex WSE means we need to address numerous issues: modeling the behavior of the end users, studying the traffic of the user queries, and responding to them using different strategies and heuristics to improve query response time, etc.

In order to do this, in this work we propose to model service-based WSE. We decided to use the DEVS formalism, and in [5], we built a DEVS model and validated such model against an actual implementation, and a process-oriented simulator. Such simulation software can be useful for capacity planning purposes in WSE. We chose to use DEVS, due to many reasons: (1) it provides a rich structural representation of components, (2) it includes a formal specification for explicitly specifying model states and time, and (3) it provides a well-defined specification for coupling of components. DEVS provides a formal foundation to M&S that proved to be successful in different complex systems. This approach combines the advantages of a simulation-based approach with the rigor of a formal methodology. The use of DEVS improves reliability (in terms of logical correctness and timing), enables model reuse, and permits reducing development and testing times for the overall process. Consequently, the development cycle is shortened, its cost reduced, and quality and reliability of the final product is improved.

Although our previous model [5] showed that, when compared to a process-oriented simulator, DEVS was easier to learn and use (while having similar accuracy for the results), we need to deploy the simulator in a distributed platform to run larger simulations and we need to explore the formal capabilities further. In order to do so, we propose to extend the WSE model presented in [5] to reduce the communication overhead introduced by the message passage among the different hierarchical components of the DEVS model and we propose to formalize the systems specification using Timed Automata. By using existing TA model checkers we verify

SummerSim-SCSC, 2016 July 24-27, Montréal, Québec, CA

properties of the WSE model formally. This reduces the complexity of the design, while providing a sound model that is guaranteed to have desirable properties for the search engine. Simultaneously, this provides a correct model for varied experimentation with simulation.

The remainder of this article is organized as follows. In Section 2, we review the DEVS formalism and the timed automatas. In Section 3, we describe a WSE and we present the improved DEVS model for distributed platforms. In Section 4 we translate the DEVS model into an equivalent TA model and perform model checking. Conclusions follow in Section 5.

BACKGROUND

DEVS Formalism

The Discrete-Event System Specification (DEVS) is a modeling and simulation formalism of Discrete-Event Dynamic systems. DEVS describes systems by means of atomic and coupled models. Atomic models are the most elemental and basic entity to represent systems. Atomic models can react to internal and external events. It defines a way of specifying systems whose states change upon the reception of an input event or the expiration of a time delay.

DEVS defines a model according to [10]:

$$DEVS = <X, Y, S, \delta_{ext}, \delta_{int}, \lambda, ta>$$

where
X is the set of *external* events;

Y is the set of *output* events;

S is the set of *sequential* states;

$\delta_{ext} : Q \times X \rightarrow S$ is the *external* state transition function, where $Q = \{(s, e) \mid s \in S,\ 0 \leq e \leq ta(s)\}$ and e is the elapsed time since the last state transition.

$\delta_{int} : S \rightarrow S$ is the *internal* state transition function;

$\lambda : S \rightarrow Y$ is the *output* function;

$ta : S \rightarrow R_0^+ \cup \infty$ is the *time advance* function;

Its semantics are as follows: At any given time, a DEVS model is in a state $s \in S$ and in the absence of external events, it will remain in that state for a period of time as defined by $ta(s)$. The $ta(s)$ function can take any real value between 0 and ∞. A state for which $ta(s) = 0$ is called a **transient state**. On the contrary, if $ta(s) = \infty$, the system will stay in that state forever unless an external event is received. In such a case, s is called a **passive state**.

Transitions that occur due to the expiration of $ta(s)$ are called **internal transitions**. When an internal transition takes place, the system outputs the value $\lambda(s)$, and changes to state $\delta_{int}(s)$. A state transition can also happens when an external event occurs. In this case, the new state is given by δ_{ext} based on the input value, the current state and the elapsed time.

The DEVS formalism includes atomic and coupled models. Atomic models allow to represent the behavior of a system, whereas coupled models represent its structure. A coupled model groups several DEVS models together into a compound model that can be regarded - due to the closure property - as another DEVS model. A coupled model is defined as a structure of the form:

$$DN = <X_{self},\ Y_{self},\ D,\ \{M_i\},\ \{I_i\},\ \{Z_{ij}\},\ select>$$

where D is a set of components, and
for each $i \in D$,
M_i is a component with constraint that
$M_i = <X_i, Y_i, S_i, \delta_{i\ ext}, \delta_{i\ int}, \lambda_i, ta_i>$ is a DEVS model;

for each $i \in D \cup \{self\}$,
I_i is the set of the influences of i;

for each $j \in I_i$,
$Z_{i,j}$ is a function, $i - to - j$ output-input translation.

$select$ is a tie-breaker function
I_i is a subset of $D \cup \{self\},\ i$ is not in I_i,
$Z_{self,j} : X_{self} \rightarrow X_j$
$Z_{i,self} : Y_i \rightarrow Y_{self}$
$Z_{i,j} : Y_i \rightarrow X_j$
$select$: Subset of $D \rightarrow D$
such that for any non-empty subset E
$select(E) \in E$

A coupled model can have its own input and output events, as defined by the X_{self} and Y_{self} sets. Upon the arrival of an external event, the coupled model has to redirect the input to one or more of its components. In addition, when a component produces an output, it has to be mapped as another component's input or as an output of the coupled model itself. The Z function defines these input-output mappings.

Simulation Environment

PCD++ [6] is used to simulate DEVS models in a parallel and distributed platform. It is implemented in C++. A simulation model is partitioned at atomic model level, and each partition is assigned to a Logical Processor (LP). The LPs are distributed among processors and they hold three kinds of elements: Node Coordinator (NC), Flat Coordinator (FC) and a set of "Simulators". The "Simulators" are the atomic models of the DEVS model.

When the sender and the receiver of an event are located on the same physical processor, the FC puts the event in the queue of the destination. On the other hand, if the receiver of the event is not in the same physical processor, the FC sends the event to the NC. Then, the NC determines the physical processor hosting the receiver "Simulator". The NC of the sending processor sends the event to the NC allocated in the destiny processor, which routes the event to its FC which determines the destiny "Simulator".

Timed Automata

A Timed Automaton is a tuple [3]: $A = (N, l_0, E, I)$ where N is a finite set of nodes or locations, $l_0 \in N$ is the initial location, $E \subseteq N \times \beta(C) \times \Sigma \times 2^c \times N$ is the set of edges, and $I : N \rightarrow \beta(C)$ assigns invariants to locations.

Where C corresponds to a set of clock variables, Σ is a set of actions and 2^c is a selection of clocks to be reset to zero. Constraints on clock variables can be used in transitions (called *guards*), or they can be used in locations (where they are called *invariants*). Guards are restrictions that need to be satisfied in order to activate a certain transition. Invariants restrict the time spent on a location of the TA.

In TA, states are pairs of the form $<L, u>$, where L is a location and u is a clock value [7]. An expression of the form $l \xrightarrow{g,a,r} l'$ when $(l, g, a, r, l') \in E$ is a transition from location l to location l' where g is a clock constraint, a is an action and r is a set of clocks to be reset to zero. Two types of transitions are used in TA: delay and action transitions. In a *delay transition* of the form $<L, u> \xrightarrow{d} <L, u + d>$ the time advance d triggers a transition from the start location to an end location. In an *action transition* $<L, u> \xrightarrow{a} <L', u'>$ an action a trigger a transition from location L to L'.

TA allows the modeling of discrete systems with continuous time. A composition of multiple interacting TA models (network of TA) can be use to easily model large systems. In [8] a methodology for verification of RT-DEVS models is presented. It proposes to establish a bisimulation relation among the DEVS model and a corresponding TA model, in which both models must have the same states and similar transitions. If the bisimulation relation is satisfied, the TA model can be used for verification of the bisimilar DEVS model. Verification is performed using the UPPAAL software tool [2] to validate the TA properties.

In the following sections we will show a case study focused on the analysis of WSE. The idea is that we have built a DEVS simulation model. We will show how that model is formally specified in DEVS, converted into a Timed Automata, and formally verified using model checking and UPPAAL.

CASE OF STUDY: WEB SEARCH ENGINES

Typically, Web search engines (WSE) are composed by three services devised to quickly process user queries in an online manner: Front-Service (FS), Caching-Service (CS) and Index-Server (IS). The FS comprises several replicated nodes. Each FS node receives and routes users queries, and sends back the top-k document results to users. After a query arrives to a FS node f_i, it asks the CS to determine whether the query has been previously processed. For the CS cluster architecture, we apply an array of $P_c \times D_c$ processors (or CS nodes). A scheduling method in the FS distribute the queries onto the P_c partitions. When a partition p_i has been selected, one of its D_c replicas is chosen in a round-robin way to search for the query. If the query is cached, the CS node sends the query answer to f_i. Afterwards, f_i sends the document results to the user. If the query is not found in cache, the CS node sends a hit-miss message to f_i. At this point, f_i sends the query to the Index Service (IS) which will compute the top-k document results.

The IS contains a distributed index built from the document collection held by the search engine (e.g. HTML documents from a big sample of the Web) [1]. This index is used to speed up the determination of what documents contain the query terms and calculate scores for them. The k documents with the highest scores are selected as the query answer.

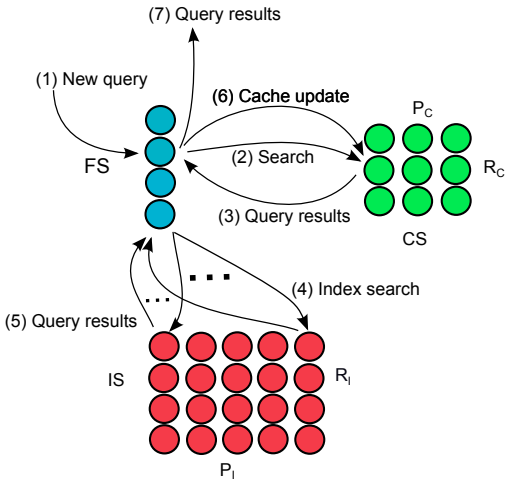

Figure 1. Query processing scheme.

For the IS, the standard cluster architecture is an array of $P_i \times D_i$ processors or index search nodes, where P_i indicates the level of document collection partitioning and D_i the level of document collection replication. The rationale of this 2D array is as follows. Each query is sent to all of the P_i partitions and, in parallel, a random replica in each partition determines the local top-k query results. These results are then collected together by the FS to determine the global top-k results for the query.

Figure 1 shows the operations performed for queries. The diagram represents the query flow through the Web search engine components. It also shows an overall description of the message traffic among processors in a cluster of processors. These messages must pass through a number of communication switches to reach their destinations.

DEVS Model definition

The work in [5] proposed a DEVS model for a WSE devised to be simulated into a single processor using CD++ [9]. It used atomic and coupled models. However, this model may incur into significant communication overhead when the simulation is executed in a distributed platform.

Suppose we have an atomic model At_1 belonging to a coupled model Co_1 allocated into the processor P_0, and a second atomic model At_2 which belongs to Co_1 but it is allocated into the processor P_4. When At_1 sends an event to At_2, the message containing the event is sent from the output port of At_1 to the output port of Co_1. Then, it goes from Co_1 hosted by P_0 to the input port of the replicated model Co_1 allocated into P_4. Finally, the event is sent to the input port of At_2. Many control events are created and passed over these ports. To reduce the communication costs caused by control events, we design a distributed DEVS simulation of a WSE model without coupled models.

To avoid potential communication bottlenecks, in the Figure 2 we propose a new DEVS-based WSE model designed for

PCD++. The new design removes the coupled models and includes new port connections between the atomic models representing WSE services. Thus, the query routing process is performed directly between atomic models.

The gray areas of Figure 2 are used to easily identify the different services of the WSE. The atomic model *QueryGenerator* is responsible for generating the user queries and delivering them to the nodes $FS_1, ..., FS_R$, which are selected in a round-robin fashion. Subsequently, queries are sent to the services CS and IS across the output ports *outCS_i_j*/*OutIS_i_j*, where i identifies the partition and j the replica of the service. The query inter-arrival time is simulated using a negative exponential distribution.

When a FS node receives a new query in its input port in, it sends the query to a Cache Service CS_{ij} across one of its output ports *outCS_i_j*. The selection of the CS node is made by determining first the partition i, via a hash function on the query terms. The selection of the replica j is performed in a round-robin way. Then, the CS node simulates the searching of the query in the cache memory and responds to the FS node with a query marked with label Hit if the query is in cache. Otherwise, the query is labeled with a $Miss$. If the response contains a Hit label, the FS node simulates the Web page construction with the document results for the query and the operation of sending these documents to the user.

However, if the answer has a $Miss$ label, the FS node routes the query to the Index Service to be processed. The query is sent to a replica of each of the P IS partitions through the output ports *OutIS_i_j*. The replicas of each partition are selected in a round-robin way. Each node in the IS simulates a ranking function to determine the top-k local documents and send them to the FS. Once the FS node has received the partial results from each of the nodes IS_i_j, a merge operation is simulated to produce the best k (top-k) document results.

Now we present the formal DEVS definition of the WSE illustrated in Figure 1.

$$WSE = \{X_{\text{wse}}, Y_{\text{wse}}, D_{\text{wse}}, \{M_{d_{\text{wse}}} | d \in D_{\text{wse}}\}, EIC_{\text{wse}}, EOC_{\text{wse}}, IC_{\text{wse}}, select_{\text{wse}}\} \quad (1)$$

Where:

$X_{\text{wse}} = \{\emptyset\}$
$Y_{\text{wse}} = \{\emptyset\}$
$D_{\text{wse}} = \{\text{QueryGenerator}, \text{FS}_i, \text{CS}_{jk}, \text{IS}_{lm}\}$
$\forall\ i \in [1, R_{\text{FS}}];\ \forall\ j \in [1, R_{\text{CS}}], \forall\ k \in [1, P_{\text{CS}}];\ \forall\ l \in [1, R_{\text{IS}}], \forall\ m \in [1, P_{\text{IS}}];$
Where: R_{FS} is the number of nodes in the FS, P_{CS} and R_{CS} are the number of partitions and replicas of the CS, P_{IS} and R_{IS} are the number of partitions and replicas of the IS.
$M_{d_{\text{wse}}} = \{M_{\text{QueryGenerator}}, M_{\text{FS}_i}, M_{\text{CS}_{jk}}, M_{\text{IS}_{lm}}\}$
$EIC_{\text{wse}} = \{\emptyset\}$
$EOC_{\text{wse}} = \{\emptyset\}$

$$\begin{aligned} IC_{\text{wse}} \subseteq \{&((\text{QueryGenerator}, \text{out}_i), (\text{FS}_i, \text{in})); \\ &((\text{FS}_i, \text{outCS}_{jk}), (\text{CS}_{jk}, \text{in})); \\ &((\text{FS}_i, \text{outIS}_{lm}), (\text{IS}_{lm}, \text{in})); \\ &((\text{CS}_{jk}, \text{out}_i), (\text{FS}_i, \text{in})); ((\text{IS}_{lm}, \text{out}_i), (\text{FS}_i, \text{in}));\} \end{aligned}$$

$$\text{select}_{\text{wse}} = \{\text{QueryGenerator}, \text{FS}_i, \text{CS}_{jk}, \text{IS}_{lm}\}$$

Next we give the formal definition of the atomic models of the FS nodes. The DEVS formalism for others WSE components can be easily obtained from these mathematical forms. We do not include them in this paper for lack of space.

$$\text{FS} = \{X_{\text{fs}}, S_{\text{fs}}, Y_{\text{fs}}, \delta_{\text{int}_{\text{fs}}}, \delta_{\text{ext}_{\text{fs}}}, \lambda, ta\} \quad (2)$$

Where:
$IN = \{\text{“}in\text{”}\}$ is the set of input ports, with values $X_{\text{fs } in} = Q$.

Q is the set of Query objects. Each $q \in Q$ has a value determining the atomic model which has send it:
$q_{\text{route}} = \{\text{“}User\text{”}, \text{“}CSHit\text{”}, \text{“}CSMiss\text{”}, \text{“}ISdone\text{”}\}$.

$X_{\text{fs}} = \{(\text{“}in\text{”}, q) | q \in Q\}$ is the set of ports and its input values (Query objects).

$S_{\text{fs}} = \{\text{“Idle”}, \text{“Proc”}\} \times \sigma \times Q \times \text{qSize}$, where “Proc” corresponds to state *Processing*, $\sigma \in \mathbb{R}_0^+$ has the time advance value according to ta, and qSize is a variable containing the current amount (size) of queries in the queue with $\text{qSize} \in \mathbb{Z}_0^+$.

$OUT = \{\text{outCS}_{ij} \cup \text{outIS}_{lm} \cup \text{“done”}\}$ is the set of output ports, with values $Y_{\text{fs } out} = Q$; where outCS_{jk} are the output ports connecting to the port *in* of the CS model $\forall\ j \in [1, R_{\text{CS}}]$ and $\forall\ k \in [1, P_{\text{CS}}]$. outIS_{lm} are the output ports connecting to the port *in* of the IS model, $\forall\ l \in [1, R_{\text{IS}}], \forall\ m \in [1, P_{\text{IS}}]$. “done” is an output port for finished queries.

$Y_{\text{fs}} = \{(p, q) | p \in OUT,\ q \in Q\}$, where q is a Query.

$$\delta_{\text{int}_{\text{fs}}}(\text{“Proc”}, \sigma, \text{qSize}) = \begin{cases} (\text{“Proc”}, 0, \text{qSize} - 1) & \text{If } 0 < \text{qSize} \\ (\text{“Idle”}, \infty) & \text{o.w.} \end{cases}$$

$$\delta_{\text{ext}_{\text{fs}}}(\text{“Idle”}, \sigma, \text{qSize}) = (\text{“Proc”}, 0, 1)$$

$$\delta_{\text{ext}_{\text{fs}}}(\text{“Proc”}, \sigma, \text{qSize}) = (\text{“Proc”}, \sigma - e, \text{qSize} + 1)$$

$$\lambda(\text{“Proc”}, \sigma, \text{Query}) = (out*, \text{Query})$$

$$ta(\text{“Proc”}, \sigma, \text{Query}) = \sigma$$

The input set (X_{fs}) is received through the input port in. (Y_{fs}) is the set of output queries sent through the set of ports OUT. The variable *qSize* is used to represent the number of queued queries in each FS node. There are two possible states: *Processing* or *Idle*. External transitions are triggered when a query arrives to the FS node. If the node is idle, the elapsed time is set to zero, the variable *qSize* is set to 1 and the state changes to *Processing*. If a query arrives when the node is busy (*Processing*), then the time is adjusted according to the time indicated by the time advance function (σ) minus the time elapsed. The variable *qSize* is incremented by one.

The internal transition function is defined only for the state *Processing* and is triggered after σ units of time. If there are queued queries, then the variable qSize is decreased by one, the elapsed time is set to zero and the FS node remains in the same state. Otherwise, if $\text{qSize} = 0$, the FS node changes its state to *Idle*.

The output function ($\lambda(\text{Proc}, \sigma, \text{Query})$) is defined for the state *Processing*, after a period of time σ and for a given query. This function returns a query (*Query*) which is sent through de port *out**. The output port *out** corresponds to

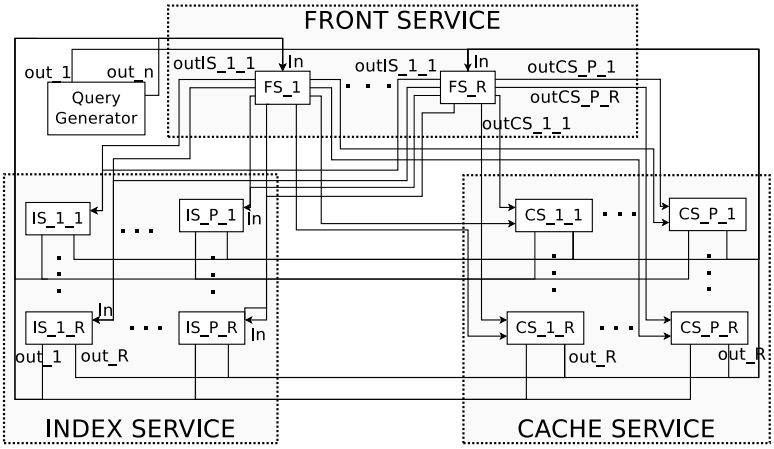

Figure 2. WSE modeled with atomic models.

one of the ports listed in Table 1, selected according to the value q_{route} contained in the query. The time advance function is defined for the state *Processing* and a query (*Query*). It returns a time σ required to process the query.

Table 1. Selection of output ports for atomic models of the FS.

Value q_{route}	Incoming atomic model	Output port
"User"	QueryGenerator	outCS_{jk}
"CSHit"	CS	"done"
"CSMiss"	CS	outIS_{lm}
"ISdone"	IS	"done"

EQUIVALENCY OF DEVS AND TA

The DEVS formalism is based on Systems Theory [10] and corresponds to a specification of discrete events. Several efforts have been started for formal verification of DEVS models [4] but at present there are no existing tools providing the means for formal verification on relevant properties such as reachability, liveliness, etc.

However, the authors in [8] presented a methodology that allows checking DEVS models to ensure they are correct. To this end, first a bisimulation relationship is established between the WSE model developed with DEVS and an equivalent WSE model developed with Timed Automata. This makes it possible to apply different tests to assess the correctness of the WSE model developed with TA. Thus, in this section we will show the effective use of TA model checking techniques for verification of DEVS models, through a WSE model as a case study. This abstraction is necessary to study only properties of concern and to simplify the modeling task.

The bisimulation between two systems A and B, defines the relationship between each state and transition of A and its corresponding state and transition in B. In this work we apply a weak temporal bisimulation equivalence as described in [8]. To determine the equivalence between atomic models of DEVS and its corresponding models in TA, we begin the analysis of the different transitions (internal and external) and the states of each atomic element.

Figure 4 and Figure 3 show the atomic model of a FS node developed with DEVS and its equivalent with the TA. At a given time, these models may be in one of two possible states: *Idle* or *Processing*. The modeled nodes can remain in each state for different units of time and can receive queries regardless of which state they are. Initially, the nodes are in the state *Idle*, and remain in that state until they receive a user query or a query from another service node. When this happens, the node changes its state from *Idle* to *Processing*.

Step1: from DEVS to TA

Figure 4 shows the states and the transitions of the DEVS model of a WSE. Internal transitions are represented by segmented lines and external transitions with continuous lines. From now on, we use "D" and "T" for the DEVS and TA models respectively.

- *fsQSize* is the amount of queued queries and its is equivalent to the *qSize* variable of the DEVS model. In the TA model, this variable corresponds to four different variables used to know the service which send the query.
  - *fsQSizeUser* amount of queries received from the user.
  - *fsQSizeCS* amount of queries received with label $Miss$ from the the CS.
  - *fsQSizeHitCS* amount of queries received with label Hit from the the CS.
  - *fsQSizeIS* amount of queries processed by the CS.
- *maxQSize* is the maximum amount of queued queries (size of the input buffer).
- *hold*, is the time to process a query in a FS node.
- *eTime (or elapsedTime)*, is the elapsed time when processing a query in the WSE.

To determine the bisimulation of both models we check if there is a relationship ***R*** of bisimilarity of its internal transitions.

- $(Proc_D \ \boldsymbol{R} \ Proc_T)$: There are two internal transitions, where the source is the state $Proc_D$. One is recursive on the $(Proc_D$ state and the other connects to the state $Idle_D$.
  - **Recursive transition:**
    Corresponds to the recursive transition of Figure 4.

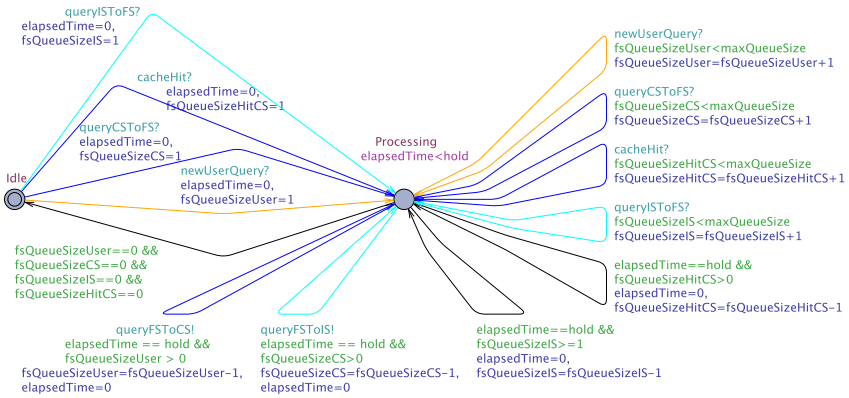

Figure 3. TA FS node model in UPPAAL.

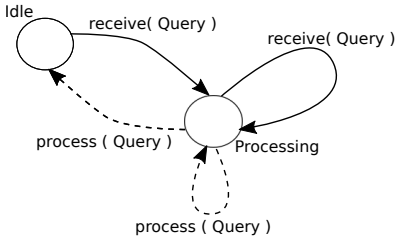

Figure 4. Atomic model with DEVS for a FS node.

It represents the fact that a query is being processed (*process (query)*) and there are queued queries. The TA model of Figure 3 has five recursive transitions (two of them labeled as "queryFSToCS!" and "queryFSToIS! ", and the others with arcs colored in black). They all belong to the same recursive transition of the DEVS model, the have the same temporal condition and they differ only in the action performed. They modify different variables of state, so for practical purpose we consider them as only one transition.

Lets consider the execution of the internal transition in DEVS:

$$(\text{Proc}_D, e) \xrightarrow{d} (\text{Proc}_D, 0) \text{ if qSize} > 0$$

as a transition delay d, where $0 \leq e < ta(Proc_D)$ and $0 \leq d < ta(Proc_D) - e$. Once the transition is triggered, the value of the time elapsed is set to $e = 0$ in the destination state. To satisfy the bisimulation we define:

$$(\text{Proc}_T, \text{eTime} = e) \xrightarrow{d} (\text{Proc}_T, \text{eTime} = e + d) \text{ if fsQSize} > 0$$

Where:

$$\text{fsQSize} = \text{fsQSizeUser} + \text{fsQSizeCS} + \text{fsQSizeHitCS} + \text{fsQSizeIS}$$

for the same value of d as the one used in the DEVS model. When considering the invariance of the initial state, we have *eTime* $<$ *hold* in $Proc_T$, so by replacing *eTime* and d we obtain:

$$ta(\text{Proc}_D) \leq \text{hold} \quad (3)$$

This means that the transition will be triggered as soon as the condition in Eq. 3 is not satisfied, and $qSize > 0$. After the transition is triggered, the clock is set to (*eTime* $= 0$). Notice that both variables *fsQSize* (from the TA model) and *qSize* (from the DEVS model) are equivalent and are used to measure the amount of queued queries in a FS node.

– **Transition from *Processing* to *Idle*:**
We have to check the bisimilarity between the states $Proc_D$ and $Proc_T$ with the timed transition from $Proc_D$ to $Idle_D$. If we consider the execution of the internal transition of DEVS: $(Proc_D, e) \xrightarrow{d} (Idle_D, e)$, where $0 \leq e < ta(Proc_D)$ and $d = ta(Proc_D) - e$, then the execution of the transition in the TA is $(Proc_T, eTime = e) \xrightarrow{d} (Idle_T, eTime = e + d)$. Both transitions are equivalent if they have bisimilar states as the initial state and the destination state (after the same delay). To this end, the same delay value d in both transitions is used. The transition begins in $Proc_T$ with *eTime* $= e$, where the value of e is the same as in the transition of the DEVS model.
After a delay d (equal to the transition in DEVS) the TA switches to the state $Idle_T$ and the clock *elapsedTime* value is increased by d units of time. Therefore, *eTime* $= e + d$, and considering the condition of the transition (guard condition) *eTime* $=$ *hold* of the transition of the TA, we have:

$$ta(\text{Proc}_D) = \text{hold} \quad (4)$$

So, when using a constant *hold* in the guard transition of the TA with a value equal to the advance of time of the $Proc_D$, the TA transition is triggered as its equivalent in DEVS. The condition of the transition *fsQSize* $= 0$ in the TA is equal to the internal transition in DEVS. This transition represents the fact that all queries have been processed and the FS node is now available (*Idle*).

Therefore, from Eq. 3 and Eq. 4 we have:

$$ta(\text{Proc}_D) = \text{hold} \quad (5)$$

which guarantees the bisimulation relationship of the internal transitions of the DEVS model with the transitions in the TA.

Step 2: from TA to DEVS

In this section we show how to satisfy the other direction of the bisimulation. That is, how the TA model of Figure 3 becomes the DEVS model of Figure 4.

- ($Proc_T$ ***R*** $Proc_R$): There are five transitions in the TA which are equivalent to the internal transitions of the DEVS model. Four transitions are recursive (two of them in colour black and the others have the labels *queryFStoCS!*, *queryFStoIS!*) in the state $Proc_T$. The fifth transition connects the state $Proc_T$ to the state $Idle_T$.

  - **Recursive transitions:** All of them have the same temporal condition. The difference is the action performed. Thus, for practical purposes they are considered as only one, and are defined as:

    $$(\text{Proc}_T, \text{eTime} = e) \xrightarrow{d} (\text{Proc}_T, \text{eTime} = e + d)$$

    If the invariant condition eTime $<$ hold is true, the TA for the FS node remains in the state Proc_T. Therefore:

    $$e + d < \text{hold} \tag{6}$$

    The transition with delay d is defined in the DEVS model as $(Proc_D, e) \xrightarrow{d} (Proc_D, e+d)$. To stay in the state $Proc_D$ after d units of times, the time advance has to be smaller than $ta(Proc_D)$. Therefore, we have:

    $$e + d < ta(\text{Proc}_D) \tag{7}$$

  - **Transition from *Processing* to *Idle*:** Corresponds to the TA transition:

    $$(\text{Proc}_T, \text{eTime} = e) \xrightarrow{d} (\text{Idle}_T), \text{eTime} = 0)$$

    The behavior of this transitions is as follows. The initial state is Proc_T with time elapsed e, and after d units of time, the TA model changes to the state Idle_T and the time variable resets to eTime $= 0$. In other words, to move from the state Proc_T to Idle_T, the invariant ($eTime < hold$) has to be false and the guard condition of the transition ($eTime = hold$) has to be true. Therefore:

    $$e + d = \text{hold} \tag{8}$$

    From DEVS, this transition is defined as:

    $$(\text{Proc}_D, e) \xrightarrow{d} (\text{Idle}_D, 0)$$

    To trigger this transition the model has to satisfy:

    $$e + d = ta(\text{Proc}_D) \tag{9}$$

Therefore, from Eq. 8 and Eq. 9 we obtain that hold $= ta(\text{Proc}_D)$. Then, we can say that Eq. 6 and 7 are equivalent and the simulation relationship from the TA model (Figure 3) to its equivalent DEVS model (Figure 4) is verified.

As there is a simulation relationship in both way (from DEVS to TA and viceversa), the internal transition of the DEVS model is bisimilar in time and the behaviour is equivalent to the timed transition in the TA model. This has been checked for a constant *hold* which represents the processing time in the state Proc_D. Then, we can say that (Idle_D ***R*** Idle_T) and (Proc_D ***R*** Proc_T).

External transitions in DEVS

In this section we show that the external transitions of the DEVS model are bisimilar to the external transition of the TA model. In Figure 4, the external transitions are represented as continue lines, which correspond to the incoming queries from users, from the IS and from the CS.

The DEVS model has two external transitions. The first one connects the state $Idle_D$ to the state $Proc_D$. When a query arrives, and if the FS node is available (the current state is $Idle_D$), the variable *qSize* is set to 1 and the model change the state to $Proc_D$.

The second external transition, is recursive on the state $Proc_D$:

$$\delta_{\text{ext}_{\text{fs}}}(\text{“Idle”}, \sigma, \text{qSize}) = (\text{“Proc”}, \sigma, 1)$$

$$\delta_{\text{ext}_{\text{fs}}}(\text{“Proc”}, \sigma, \text{qSize}) = (\text{“Proc”}, \sigma - e, \text{qSize} + 1)$$

If a query arrives and the current state is $Proc_D$, the new query is allocated in the queue of the FS node. Thus, the variable *qSize* is increased by 1.

As both external transitions do not consider time restrictions, they can be expressed as action transitions:

1. $Idle_D \xrightarrow{a} Proc_D$
2. $Proc_D \xrightarrow{a} Proc_D$

With this expression we can represent the external transitions as transitions of the TA:

1. $(Idle_T, eTime) \xrightarrow{a} (Proc_T, eTime = 0)$.
2. $(Proc_T, eTime) \xrightarrow{a} (Proc_T, eTime)$

These expressions are used to obtain the relation ***R*** between the DEVS model and the TA model. Then, ($Idle_D$ ***R*** $Idle_T$) and ($Proc_D$ ***R*** $Proc_T$) are satisfied. In the same way, we can verify the relationship from TA to DEVS, which shows that there is a bisimulation relationship between the DEVS model and the TA model.

Evaluation of correctness of the WSE model

In this section we use model checking on the equivalent TA model of the DEVS model, to answer questions about our DEVS model behavior that otherwise would have needed to fully simulate all possible executions of the DEVS model to get the answers. In this section we analyze the following properties: Reachability, Safety, Liveness and Deadlock free. To this end, we formulate formal queries to the UPPALL Model Checker Tool version 4.1.19 (*November 2014*) [2].

Reachability

This property is used to determine that all states are achievable. To verify this property, we evaluate the queries `E<> FS.Idle` and `E<> FS.Proc` on the TA model. After the execution of these queries, UPPAAL shows the response `Property is satisfied`, which means that the states are achievable. In other words, at some point in the simulation, the FS nodes of a WSE are available (*Idle*) or processing (*Proc*).

Safety

This property is used to verify that there are not anomalous behaviors in the system. For this reason, we define which combinations of states and values of state variable are correct.

- At state *Idle* all queues must be empty:

```
E[] (FS.Idle and not (FS.fsQSizeUser>0
     or FS.fsQSizeCS>0 or
     FS.fsQSizeHitCS>0 or
     FS.fsQSizeIS>0))
```

  When we execute this query with UPPAAL, we get the answer `Property is satisfied`, indicating that these behaviors do not occur.

- The input queues of the FS node has one or more queries, and less than a maximum value (*maxQueueSize*):

```
A[] (FS.fsQSizeUser>=0 and
     FS.fsQSizeCS>=0 and
     FS.fsQSizeHitCS>=0 and
     FS.fsQSizeIS>=0 and
     and CS.csQSize>=0 and
     FS.fsQSizeUser<=maxQSize and
     FS.fsQSizeCS<=maxQSize and
     FS.fsQSizeHitCS<=maxQSize and
     FS.fsQSizeIS<=maxQSize)
```

  Again, the UPPAAL tool shows the message `Property is satisfied`.

Liveness

This property is used to verify if any state is eventually reached from another state. To do this, an initial state and a final state are defined. First we consider the initial state *Idle* and the destination state *Processing* by executing the query `FS.Idle --> FS.Proc`. In this case, the UPPAAL tool shows the message `Property is not satisfied`.

The liveness property is not satisfied because in the model developed with TA, there is no invariant to limit the time to stay in the initial state *Idle* and to force the change to another state. However, for the case study of this work -WSE based on services- does not correspond to include an invariant which satisfies this property, because in practice it could happen that a FS node does not receive queries and stays indefinitely in the state *Idle*. In other words, some FS nodes could possibly never receive queries.

Next, we evaluate if the property is verified considering the initial state *Processing*. In this case, the query to be executed with UPPALL is `FS.Proc --> FS.Idle`. The UPPALL tool shows the message `Property is satisfied`, indicating that after all queries are processed, the FS node change its state to *Idle*. The invariant ($eTime < hold$) defined for the state *Processing* represents the fact that it is necessary to elapse *hold* units time to process a query, and it allows to satisfy the property.

Even though the property is not always satisfied, it is expected that FS nodes will receive queries, changing the state from *Idle* to *Processing*. Thus, not satisfying this property does not represent a fault in the model developed for the WSE.

Deadlocks

We analyze if deadlocks can occur. To this end, we execute the query `A[] not deadlock`. To satisfy this property include we include the condition fsQSizeUser $<$ maxQSize, in Figure 3, on the transitions representing the arrival of queries from the IS and from the CS. However, this condition does not exist in the WSE model developed with DEVS. This condition limits the size of the input queue of the FS node modeled with the TA. If not limited, the TA would present deadlocks, as the input queue could grow infinitely. However, this condition is not fictitious because in practice no queue (implemented computationally) can reach an infinite number of elements because memory is limited.

CONCLUSION

In this paper we presented a DEVS model for a WSE for efficient distributed simulations. The proposed model aims to reduce the amount of messages transmitted among different elements of the DEVS model.To verify the DEVS model of a WSE we performed a bisimulation in order to obtain an equivalent TA model. We performed model checking on the TA model to check the correctness of the proposed model

ACKNOWLEDGMENTS

This work has been partially funded by CONICYT Basal funds FB0001.

REFERENCES

1. Baeza-Yates, R. A., and Ribeiro-Neto, B. A. *Modern Information Retrieval - the concepts and technology behind search, Second ed.* Pearson Education Ltd., 2011.
2. Behrmann, G., David, A., and Larsen, K. A tutorial on uppaal. In *SFM-RT* (2004), 200–236.
3. Bengtsson, J., and Yi, W. Timed automata: Semantics, algorithms and tools. In *Petri Nets* (2004), 87–124.
4. Hwang, M. H., and Zeigler, B. P. Reachability graph of finite & deterministic devs. In *Proceedings of 2006 DEVS Symposium* (2006), 48–56.
5. Inostrosa-Psijas, A., Wainer, G. A., Costa, V. G., and Marín, M. Devs modeling of large scale web search engines. In *WSC* (2014), 3060–3071.
6. Liu, Q., and Wainer, G. Parallel environment for devs and cell-devs models. *SIMULATION 6*, 83 (2007), 449–471.
7. Saadawi, H., and Wainer, G. Rational time-advance devs (rta-devs). In *SpringSim* (2010), 143:1–143:8.
8. Saadawi, H., and Wainer, G. Principles of discrete event system specification model verification. *Simulation 89*, 1 (2013), 41–67.
9. Wainer, G. A. Cd++: a toolkit to develop devs models. *Softw., Pract. Exper. 32*, 13 (2002), 1261–1306.
10. Zeigler, B. P., Kim, T. G., and Praehofer, H. *Theory of Modeling and Simulation*. Academic Press, Inc., 2000.