Proceedings of the 2010 Winter Simulation Conference
B. Johansson, S. Jain, J. Montoya-Torres, J. Hugan, and E. Yücesan, eds.

RISE: REST-ING HETEROGENEOUS SIMULATIONS INTEROPERABILITY

Khaldoon Al-Zoubi
Gabriel Wainer

Dept. of Systems and Computer Engineering
Carleton University Centre of Visualization and Simulation (V-Sim)
1125 Colonel By Dr. Ottawa, ON, CANADA

ABSTRACT

Interoperating heterogeneous simulation models and tools is becoming a necessity in today's cross-enterprise collaboration market. Nevertheless, simulation models and engines have evolved apart in many directions, making their interoperability extremely complex. We present the RESTful Interoperability Simulation Environment (RISE), which provides the means for interoperating simulation heterogeneous assets. RISE uses Service-Oriented RESTful web-services, and it is based on three aspects: the framework architecture, the modeling level and the simulation synchronization level. RISE is independent of any simulation engine, theory or an algorithm. However, it provides different rules for simulation domains with conservative or optimistic synchronization algorithms. Further, RISE does not require any implementation changes related to domain modeling or simulation methods. Furthermore, it hides domain internal specifics, giving freedom to define different internal implementation and algorithms. The presented work here is part of the on-going effort in the DEVS community to interoperate different DEVS-based simulation assets.

1 INTRODUCTION

In today's changing world, placing hardware or software assets around the world is becoming not enough reason for a user to be deprived of using them. For example, Universities are increasingly offering on-line programs via the Internet, allowing students to complete their education without being physically on campus (and all students are expected to receive the same level of education regardless of this). Sharing and reusing assets go beyond the education sector, or industry, and it is not only related to the fact that not every institution can afford purchasing the required hardware/software. For instance, conducting experiments between different remote teams enlarge the circle of thinking for solving challenging problems (e.g. epidemics, defense, emergency etc.) quickly and efficiently. Interoperating these scattered assets is the main challenge. Making applications developed independently interact with each other is not trivial, since this interaction involves not only passing remote messages, but also synchronizing them (interpreting messages and reacting to them correctly). Interoperating different companies' assets translate into lower cost and increased return of investment (ROI), because interoperability translates into collaboration bridges among organizations. The ability to create and deploy new products rapidly at low cost is a competitive advantage. Simulation assets also need to be interoperated, allowing collaboration in studying systems across organizations.

Simulation interoperability requires specific simulation models to be executed on simulation engines according to a specific theory shared with different simulation teams. In general, modelers use the simulation engines that they are familiar with, and can be experts within a simulation engine environment, but unable to use others. This partly explains the limited use of distributed simulation in industry, since ROI must always outweigh cost. Therefore, interoperability simulation standards must be easy to understand,

hide modeling specifics and hide simulation engines specifics such as theory and algorithms. These are necessary requirements because they translate into low cost with appealing ROI. In reality, a company, government office or research lab is not going to devote resources in order to interoperate heterogeneous models. On the other hand, avoiding the need to learn heterogeneous simulation environments, simulation algorithms, engine requirements, and modeling specifics is a true step toward low-cost simulation interoperability.

The need for standardized simulation experiment framework, allowing synchronization among different heterogeneous simulators is in a constant growing. The presented work here is part of the on-going effort in the DEVS community to interoperate different DEVS-based simulation assets. Indeed, DEVS implementations use different synchronization protocols to achieve distributed simulation, which added more difficulty for developing practical standards that does not require major software design changes to existing systems (Wainer et al. 2010a; Wainer et al. 2010b).

Based on these ideas, we propose here a plug-and-play simulation interoperability based on the Representational State Transfer (REST) Web-services style (Richardson et al. 2007). The method, named RESTful Interoperability Simulation Environment (RISE), provides a lightweight thin middleware based on RESTful Web-services. RISE allows heterogeneous simulation resources to interoperate at three levels. (1) The interoperability framework architecture level provides the Application Programming Interface (API) that allows modelers to create a simulation environment (including distributing simulations, starting simulation and retrieving results). (2) The model interoperability level provides XML rules for binding different models together (and defining domains as regions responsible of executing a simulation models connected with other domains' heterogeneous models). (3) The simulation synchronization level provides high-level simulation algorithms and synchronization channels.

We show how RISE allows interoperating heterogeneous simulation assets while hiding internal domain implementation using three URIs that can be named and constructed by clients. Existing systems do not to change their software implementations to support RISE. Further, RISE considers all of the necessary issues to enable modelers to conduct complete simulation experiments from the point of setting up the environment until retrieving results.

2 BACKGROUND

RESTful Web-services (Richardson et al. 2007) provides interoperability by imitating the World Wide Web (WWW) style and principles. RESTful Web Services (Richardson et al. 2007) are gaining attention with the advent of Web 2.0 (O'Reilly 2005) and the concept of mashups (i.e. grouping of various services from different providers presented as a bundle). REST lightweight approach hides internal software implementation (in "black boxes" called resources). Each resource exposes uniform channels (interfaces) and describes connectivity semantics between resources in form of messages (usually XML). In contrast, other approaches, such as CORBA or SOAP-based Web-services (Erl et al. 2008), expose functionalities in heterogeneous RPCs that often reflect internal implementation and describe semantics as procedure parameters (making difficult to interoperate existing simulation tools or to achieve plug-and-play distributed simulations. We detailed distributed simulation issues and future trends in (Wainer et al. 2010c). Plug-and-play interoperability middleware is needed to advance distributed simulation in the industry sector as indicated by a number of surveys of experts from different backgrounds (Strassburger et al. 2008). Further, REST enables simulations to mashup with Web 2.0 applications (O'Reilly 2005), and linking any device attached to the Web at runtime.

RESTful services are spread over a set of connected resources where each resource is named with a URI (similar to a website). Service consumers connect with those resources via universal standardized virtual uniform channels where semantic messages are applied to those resources. In our case, the channels are the HTTP methods showed in Figure 1: GET (to read a resource entirely or partially), PUT (to create a new resource or update existing data), POST (to append new data to a resource), and DELETE (to remove a resource). Resources use those channels to transfer their data (representation) among each other, hence transferring their representational state as specified by the name of the Representational State

Transfer (Richardson et al. 2007). RISE, formally known as RESTful-CD++ (Al-Zoubi et al. 2009a; Al-Zoubi et al. 2009b), is a general middleware based on RESTful Web services. It provides a simulation environment independent of formalisms or tools. A simulation engine can be plugged into RISE regardless of its specifics (providing simulation service to modelers). For example, CD++ (Wainer 2009) has been plugged into the middleware to provide distributed simulation among different domains.

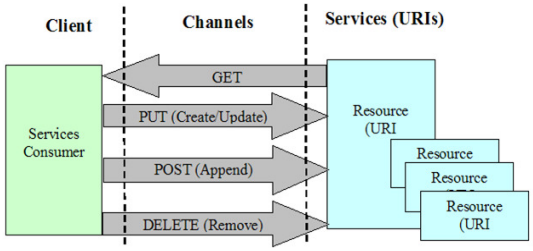

Figure 1: Uniform Channels for RESTful Resources

RISE API is expressed as URI templates that can be created at runtime. Variables in URI templates (written within braces {}) are assigned at runtime by clients before a request is sent to the server, enabling clients to name their URIs. For example, RISE defines the simulation framework URI template as: <…/{userworkspace}/{servicetype}/{framework}>, where {userworkspace} is a specific workspace. The workspace allows users to define their specific URI hierarchy while avoiding naming conflicts. The {servicetype} is the selected simulation service, allowing a client to use different services simultaneously. The {framework} is the simulation experiment framework; hence, a user may create multiple experiment frameworks that use the same simulation service. For instance, URI <…/Bob/DCDpp/MyModel> indicates that the user workspace belongs to user Bob, and the servicetype is DCDpp, which selects the distributed CD++ engine. The framework is named MyModel, which is the name of the simulation experiment. In this case, the modeler may select a different simulation engine (instead of DCDpp) or a different framework (instead of MyModel), because these variables are assigned at runtime according to the API URI template. Therefore, URI templates enable modelers to name their URIs without being in conflict with other users. RISE API is fully described in (Al-Zoubi et al. 2009a; Al-Zoubi et al. 2009b).

3 RISE LAYER INTERFACE

The main objective of RISE is to perform a simulation session between heterogeneous models that are only executable within a specific simulation environment. Model heterogeneity also exists in simulation environments based on the same formalism. Therefore, heterogeneous models need to be interoperated to form single distributed model so that a single simulation session (i.e. single experiment) can be conducted. Modelers are responsible of setting up the simulation experiment and interconnecting those heterogeneous models. The RISE layer places every simulation model in a separate domain where it can be simulated within a compatible environment. A Domain is a simulation environment capable of executing enclosed simulation models. Thus, domains can be viewed as models at the modeling interoperability level whereas viewed as simulation engines at the simulation-synchronization interoperability level. Typically, modelers are expected to collect the models needed, and choose a compatible simulation environment. Afterwards, each model needs to be placed in appropriate domain. Therefore, each domain contains and simulates a model that is specific to that domain and interconnected (via RISE) with other domains models.

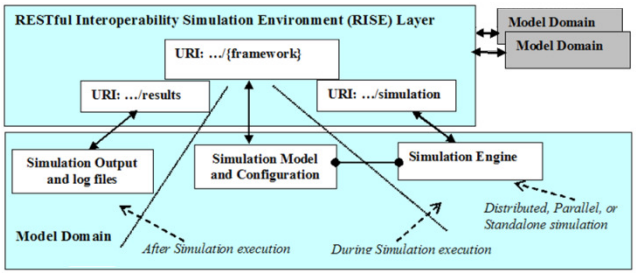

Figure 2: RISE Layer and Model Domains Interface

RISE requires three RESTful resources (URIs) for each domain so that other domains and modelers can use them to setup and conduct simulation, as shown in the RISE component in Figure 2. These resources (URIs) are described as follows: (1) …/{framework}: represents an experiment framework in a domain (i.e. simulation environment). It is named by the modeler upon creation. The modeler uses this URI to submit all necessary information to execute simulation in that domain such as simulation model (e.g. scripts, source code files, etc.) and RISE XML configuration. This URI is the parent of the other two needed resources described next. (2) …/{framework}/simulation: represents active simulation in a domain. The modeler uses this URI to start/abort simulation, and to manipulate simulation during runtime. This URI is also used by simulation engines in domains to exchange simulation synchronization messages. (3) …/{framework}/results: is automatically created by a domain upon completing the simulation successfully, allowing to retrieve the simulation results.

The above-described three URIs are the RISE API of each domain, as shown in Figure 2. Thus, a domain is plugged into RISE layer and is exposed to the outside world via those three URIs. In fact, those URIs are wrappers of RISE relevant information in a domain. The bottom domain block in Figure 2 shows an internal look for the wrapped up functionalities: Simulation outputs (maintained results upon completion), Model related information and the simulation engine that is in charge of executing domain-enclosed model. Simulation engines can be standalone (i.e. sequential simulation on one machine), parallel/distributed (i.e. partitioned simulation over multiple processors regardless of the underlying technology).

Resources …/simulation and …/results cannot exist simultaneously. Further, each domain may append the RISE API templates to any URI design of their choice (a resource URI is the full URI path). Modelers (clients' software) should be aware of the domain URI templates (this design adds flexibility and simplicity at the server side: RISE does not require all servers to use exactly the same URI template, enabling different servers to organize and support other services beside RISE). To implement the API, each resource specification must be clearly defined in terms of supported HTTP channels, message format, and HTTP response codes upon success r failure. RISE strictly follows the Web standards such as HTTP, URI and XML. These specifications are discussed next (summarized in Table 1, Table 2, and Table 3). Specific domains respond with standardized HTTP error codes for rejected requests, such as 404 (Not Found), 403 (Forbidden), 401 (Unauthorized), 400 (Bad Request), 415 (Unsupported Media type), 501 (Not Implemented) and 500 (Internal Server Error).

Table 1 summarizes the framework resource operations. PUT is used to create/update a framework, but may also contain domain-specific configuration (an XML configuration document that contains the RISE models interconnections sent through this channel). POST is used to submit/update all/part of the model specific files. The XML configuration document (via PUT channel) usually describes these files. DELETE is used to remove a framework. GET is used to allow modelers to check on simulation status. The modeler request must contain query variable sim with value status, e.g. …/firemodel/simulation?sim=status. Query variables define the scope of the request, to avoid returning the entire resource representation and domain-specific information (domains may have other uses of this channel). This request is intended for modeler's client to keep checking the simulation status periodically via the main domain. The main domain is selected by the modeler's configuration, and the receiver should

respond with one of the following states: IDLE (simulation never run), INIT (simulation being initialized), RUNNING, ABORTED (simulation stopped before successful completion), ERROR (simulation exited on error), DONE (simulation completed successfully) and STOPPING (simulation DONE, but there is still work to be done for the previous session). The XML message should be in the following format: <Simulation> <Status>DONE</Status></Simulation>.

Table 2 outlines the active simulation resource. PUT is used to create the resource and start the simulation. DELETE is used to stop the simulation and to remove the resource. POST is used to send XML synchronization messages to a domain. GET is used by other domains to check if the simulation is still running. In this case, a domain should respond with the following XML message <Simulation> ALIVE </Simulation> to indicate resource presence. The HTTP server responds with error 404 (Not found) to indicate resource absence, allowing the sender domain to take the appropriate actions. GET is also used to return a domain Global Virtual Time (GVT) used for optimistic synchronization. Some HTTP servers reject NULL messages through the PUT channel; in this case, a dummy message may be sent and ignored at the receiver domain.

Table 3 outlines the simulation results, allowing retrieval of the domain simulation outputs. PUT and POST are disabled since this resource is created upon successful simulation completion. GET is used to retrieve results in a zipped folder, and DELETE is used to force results removal.

Table 1: Resource "…/{framework}" Specifications

Action	Channel	HTTP Success code	Message
Create framework	PUT	201(Created)	XML
configure framework	PUT	200 (OK)	XML
Submit models	POST	200 (OK)	Zipped File
Remove framework	DELETE	200 (OK)	None
Get Simulation Status	GET	200 (OK)	XML

Table 2: Resource "…/{framework}/simulation" Specifications

Action	Channel	HTTP Success code	Message
Start Simulation	PUT	202 (Accepted)	None
Stop/Abort Simulation	DELETE	200 (OK)	None
Send Simulation Message	POST	202 (Accepted)	XML
Check Simulation or Get DGVT	GET	200 (OK)	XML

Table 3: Resource "…/{framework}/results" Specifications

Action	Channel	HTTP Success code	Message
Remove results	DELETE	200 (OK)	None
Download results	GET	200 (OK)	Zipped file
Disabled Channels	PUT/POST	501 (Not Implemented)	Not Applicable

Figure 3 summarizes what we have described in this section. The client first sets up the required frameworks (1) by creating and configuring all domains (including XML configurations with domain specific definitions). Then, models are submitted to their appropriate domains (2). The client software contacts each domain individually rather than submitting everything through a single domain (which is the case of RESTful-CD++). However, the later approach would restrict all servers to have the same full URI path in order to support RISE API resources. This would complicate URI design template for those servers because they are expected to support other services beside RISE, which might find RISE URI template undesirable.

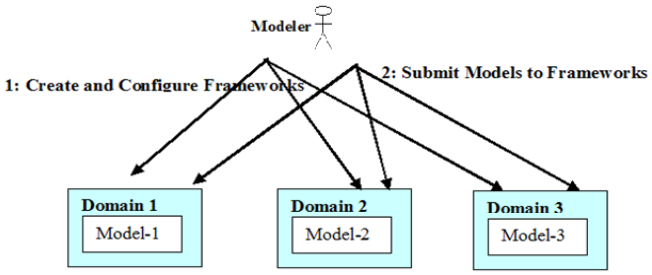

Figure 3: Setting up RISE Experiment across Domains

Clients must authenticate to manipulate domain resources; synchronization messages also need to authenticate during active simulation. To do so, each domain should have a user account in all of other domains. However, creating user accounts is a domain-specific mechanism. To achieve higher security, HTTP can be replaced with HTTPS. On the other hand, we assume here that all received messages should be authenticated according to the HTTP Basic Authentication defined in RFC 2617 (Franks et al. 2007). This method does not add extra overhead, and it is supported by Web browsers and Web programming languages. In this method, the client combines and encodes user name and password into a single string with base 64 encoding.

4 MODELING AND CONFIGURATIONS

Modelers need to configure the frameworks they create by sending XML configuration documents via PUT channels to their URIs. The domains URIs are named by modelers according to the API URI templates described in Section III. Each simulation model needs to be placed in a simulation environment domain capable of executing that model. Consequently, RISE treats all models (domains) as black boxes with input and output ports. In other words, each model views other heterogeneous models as part of its environment. Specifically, modelers need to connect the appropriate input ports to the appropriate output ports, as shown in Figure 4.

In Figure 4, Model-2 influences Model-1 via ports-2 and 3. Model-3 influences both Model-1 and Model-2 via Port -1. This implies that those models output ports can generate RISE external messages to their correspondent input ports during simulation, hence influence recipient models. Note that Model-3's Port #1and #3, and Model-1's Port-4 ports are not used by RISE. These ports can still be used by modelers, for instance, to influence a model dynamically during active simulation.

Figure 5 shows the XML configuration document for the example in Figure 4. This document needs to be submitted to each domain (typically constructed and submitted by client software). The RISE configuration must be the enclosed element <RISE> body, shown in lines 4-30 of Figure 5 where domains are free to define their own specific configuration outside the <RISE> block. Line #3 shows RISE protocol version, and simulation type where "C" type stands for Conservative-based and "O" type for Optimistic-based simulation (see Section I). Lines 4-32 define domains (models) configurations. Line #5 defines the Main domain URI. The Main domain is needed during active simulation, discussed in Section I. Lines 6-31 define RISE ports interconnections. For example, Lines 7-12 define the connection from port #2 in URI <…/Domain-2> to port #1 in URI <…/Domain-1>. In this case, a RISE external message, during simulation, is generated and sent to URI <…/Domain-2>. To do so, RISE expects domains to create routing tables similar to the shown example in Table 4 upon receiving the RISE XML configuration document. The upper two rows (of Table 4) show the routing table of Domain-2, whereas the lower two rows show the routing table of Domain-3, based on the shown configuration in Figure 5. Note that Domain-1 does not have routing table because it does not influence any of the other domains. For example, Domain-3 generates two external messages for each single output message appears on its Port #1: the first one is sent to URI <…/Domain-2> on Port #4, and the second one is sent to URI <…/Domain-1> on Port #3.

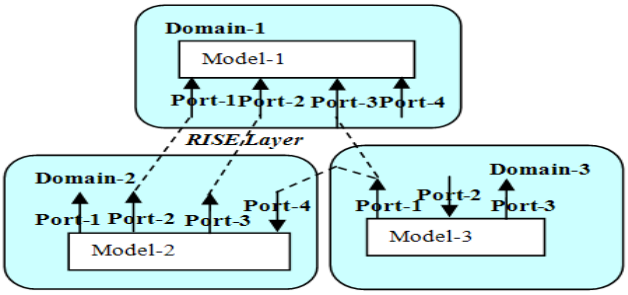

Figure 4: Models Interconnections across Domains

```
1 <ConfigFramework>
2  …
3  <RISE Version="1.0" Type="C">
4   <Domains>
5    <Main><URI>…/Domain-1</URI></Main>
6     <Links>
7      <Link>
8       <From><Port>Port-2</Port>
9              <URI>…/Domain-2</URI></From>
10      <TO><Port>Port-1</Port>
11           <URI>…/Domain-1</URI></TO>
12    </Link>
13    <Link>
14     <From><Port>Port-3</Port>
15           <URI>…/Domain-2</URI></From>
16     <TO><Port>Port-2</Port>
17           <URI>…/Domain-1</URI></TO>
18    </Link>
19    <Link>
20     <From><Port>Port-1</Port>
21             <URI>…/Domain-3</URI></From>
22     <TO><Port>Port-4</Port>
23             <URI>…/Domain-2</URI></TO>
24    </Link>
25    <Link>
26     <From><Port>Port-1</Port>
27             <URI>…/Domain-3</URI></From>
28     <TO><Port>Port-3</Port>
29             <URI>…/Domain-1</URI></TO>
30     </Link>
31    </Links>
32   </Domains>
33 </RISE>
34 …
35 </ConfigFramework>
```

Figure 5: RISE Domain XML Configuration Document (see Figure 4)

Table 4: Domains RISE Routing Tables for Figure 4

Domain	Source Port	Destination Port	Destination URI
Domain-2 Routing Table	Port-2	Port-1	…/Domain-1
	Port-3	Port-2	…/Domain-1
Domain-3 Routing Table	Port-1	Port-4	.../Domain-2
	Port-1	Port-3	…/Domain-1

5 RISE SIMULATION SYNCHRONIZATION

RISE domains could also be simulation engines, each of them responsible of simulating a single model regardless of their type (i.e. standalone, parallel or geographically distributed) or their applied algorithms. Simulations within a domain may apply optimistic-based or conservative-based algorithm approaches or define their own techniques to enhance performance within a domain. In fact, from RISE perspective, all of these issues are internal and domains specifics, hence hidden from the RISE layer. Modelers start a simulation on the main domain, which, in turn, starts simulations on all other domains, as shown in Figure 6. The main domain is previously selected by modelers' configuration (Line #6 in Figure 5).

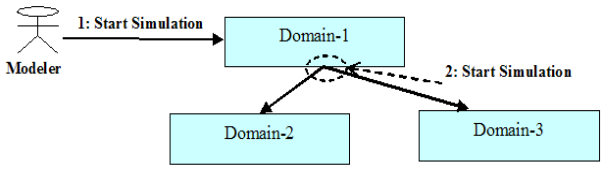

Figure 6: Starting Simulation Example (see Figure 5)

Upon accepting the request, domains are expected to respond with HTTP code 202 (Accepted, shown in Table 2). This avoids possible HTTP time-outs, particularly if the initialization phase requires a long time (for instance, if initialization in a domain involves many geographically distributed machines). Domains should start their simulation initialization phase immediately; however, conservative domains are not allowed to execute any events with time-stamp larger than zero. Furthermore, domains must buffer any messages received from the RISE layer if those messages are received during initialization (until the initialization phase is complete). Conservative RISE requires all domains to transmit safe events (i.e. stamped with the current/future RISE simulation time) through RISE. On the other hand, optimistic RISE requires domains to send/handle any event stamped with or later than the Global Virtual time (GVT).

5.1 Conservative-based Synchronization

The idea is always to satisfy the local causality constraint (Fujimoto 2000) ensuring a safe timestamp-ordered processing of simulation events within each domain. The common implementation of this conservative simulation cycle is to have a central time unit calculates the minimum global time in each participant's logical processor (LP). This is done by having each LP to send this information to the time unit, and to wait for the minimum time in which they can safely process their local events. In RISE, upon simulation startup, the main domain starts on all support domains (as shown in Figure 6) and then creates a RISE Time Manager (RISE-TM) to deal with synchronization. RISE-TM is reached at the same URI of the main domain simulation, thus, it could be implemented in the central component of the main domain. Thus, the synchronization between the main domain and the RISE-TM is internal implementation specific. However, we separate them in our discussion here for clarity.

RISE-TM executes a simulation cycle in the following steps, shown in Figure 7: (1) Execute all events in all domains at current RISE time. This starts a new simulation cycle with current or newly calculated RISE time. RISE-TM starts with time zero, so this message must be buffered by a domain if the initialization phase is not completed yet. Once a domain completes execution all of its events with RISE time, it responds to the RISE-TM with one XML message containing all external messages generated for other domains stamped with RISE time (or larger), and its next time. The next time is the time of next event in a domain larger than RISE time. If no more events exist, this value is then set to "-1", indicating infinity. (2) Once RISE-TM receives all replies from relevant domains, it calculates the next RISE time and starts a new simulation cycle. Further, the RISE-TM merges all generated external messages and passes them to all relevant domains at the beginning of a simulation cycle. Note that the new simulation cycle might be a continuation of current simulation cycle since external messages might be stamped with current RISE time. Note further, the RISE-TM stops simulation, if it calculates a new RISE time to be in-

finity. Figure 8 shows an example of a domain response to RISE-TM. In this case, executing Domain-2 events generated two external events for Domain-1 (see Figure 4).

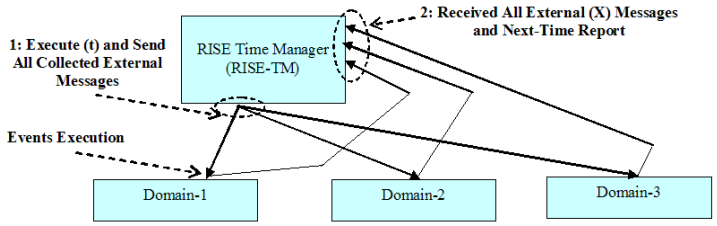

Figure 7: Conservative Simulation Cycle at Time t

```
1 <RISE Version="1.0">
2 <URI>…/Domain-2</URI>
3  <XEvents>
4    <MessagesCount>2</MessagesCount>
5       <XEvent>
6          <Time>00:00:01:000</Time>
7          <Port>Port-1</Port>
8          <Value>9</Value>
9          <URI>…/Domain-1</URI>
10       </XEvent>
11       <XEvent>
12          <Time>00:00:02:000</Time>
13          <Port>Port-2</Port>
14          <Value>10</Value>
15          <URI>…/Domain-1</URI>
16       </XEvent>
17    <Time>00:00:01:000</Time>
18  </XEvents>
19  <Next>00:00:03:000</Next>
20 </RISE>
```

Figure 8: Domain to RISE-TM Simulation Messages

Line 2 in Figure 8 indicates the URI of the source domain. Lines 3-18 enclose all of the RISE external messages generated. Line 4 specifies the count of enclosed messages. Lines 5-10 define the first external message. Line 6 specifies the execution time of this message, which must be at or later than RISE time (with format HH:MM:SS:MS). It is important to standardize the time format because numbers may interpreted differently by different models (e.g., 5 might mean five seconds in a domain and five nanoseconds in another). However, this XML document can easily be extended (with future RISE versions) to format times with less than milliseconds, if needed. Line 6 indicates that this external event must be executed at 1 second of RISE time. Line 7 specifies the model destination port (see Table 4). Line 8 specifies the message content. Message contents are real numbers, which is usually handled in programming similar to Java/C++ double type. Further, the XML document can easily be extended to handle other types in the future, if needed. Line 9 indicates the destination domain (see Table 4). Lines 11-16 defines the second external event, but with different execution time, as indicated in Line 12. Line 17 specifies the minimum time of all enclosed external messages. RISE-TM must include this time when calculating next RISE time. Line 19 specifies the time of the next event of that domain. RISE-TM must include this time when calculating next RISE time. Further, it is recommended that RISE-TM does not include domains in the next simulation cycle if they have nothing to do. Note that this value must be set to "-1", indicating infinity, if there is no more events in that domain. This XML document ***guarantees*** that all of the domain events stamped with RISE time have executed. This guarantee must be ensured by the RISE-TM by ensuring that the "next" event time (Line 19 in Figure 8) is larger than the current RISE time, since it is the time of the next event in a domain. Therefore, domains must only respond one time with this XML document.

```
1 <RISE Version="1.0">
2  <Time>00:00:01:000</Time>
3  <XEvents>
4    <MessagesCount>2</MessagesCount>
5       <XEvent>
6          <Time>00:00:01:000</Time>
7          <Port>Port-1</Port>
8          <Value>9</Value>
9          <URI>…/Domain-1</URI>
10       </XEvent>
11       <XEvent>
12          <Time>00:00:02:000</Time>
13          <Port>Port-2</Port>
14          <Value>10</Value>
15          <URI>…/Domain-1</URI>
16       </XEvent>
17  </XEvents>
18 </RISE>
```

Figure 9: RISE-TM to Domain Start Cycle Message

Line 2 in Figure 9 specifies the RISE time; every event with this time must be executed in this cycle. Lines 3-16 enclose all collected external messages from all domains. Therefore, domains must ignore other events. This is easily done via comparing its URI to events destination URIs (see Lines 9 and 15 in Figure 9). It is possible for a RISE-TM implementation to send only a domain message for that domain, but domain implementation should not assume this. Figure 10 shows a simulation example. RISE-TM starts requiring all domains to execute the events at time zero. Domain-1 responds with its next event time, in this case 2. Domain-2 responds with time 2 as its next event timestamp, and generates external message stamped with time 0, intended for Domain-1. RISE-TM calculates current time to be zero and starts a new simulation cycle. The second simulation cycle is a continuation of the previous one. Domain-1 executes the external message X (0) and it schedules itself at time 2. In third cycle, RISE-TM advances time to 2, and requires Domain-1 and Domain-2 to execute the events at this time. Domain-1 schedules itself at time 6, while Domain-2 schedules itself at time 8 and generates external message X(7) intended for Domain-1. In the fourth cycle, the RISE-TM unit advances time to 6 and passes X(7) to it. Domain-1 does not execute X(7), but inserts in its events queue until RISE time becomes 7.

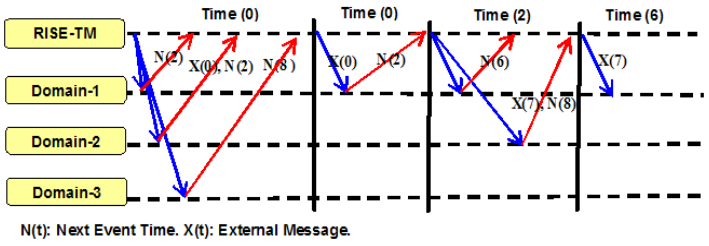

Figure 10: Conservative Simulation Cycles Example

In the presented approach here, all external simulation messages are transmitted at no cost, since they are piggyback along with the synchronization messages that are required for the conservative-based domains to know the global simulation time (i.e. called here RISE time) before executing local events safely. Thus, this enhances performance because external messages in distributed environment are expensive that take in range of milliseconds to seconds. Further, it ensures accurate RISE time calculation without having complex synchronization schemes, since all of the needed information is available for the RISE-TM upon calculation new RISE time, without the worry about any possible transit messages in the network or messages that about to be generated from a domain. Furthermore, the "next" time information received from domains enables the RISE-TM to consider relevant domains in a simulation cycle, hence speeding up a cycle execution. The "next" time value serves as a dynamic "lookahead" value at the simu-

lation level. Lookahead value is the time distance between two simulation processors, which ensures a processor to process events in that time distance safely without synchronization with its influencers. The lookahead value can also be defined at the application (model) level, but it is difficult to extract by modelers in complex applications. However, it is worth to note that RISE can be extended to handle application level configured lookahead values. In this case, domains become allowed to execute events within the window of current RISE time plus the lookahead time distance.

The example in Figure 10 is different from the parallel DEVS implementation described in (Al-Zoubi et al. 2009b), in which a simulation cycle occurs in two phases (the first phase collects external messages and the second phase executes all internal and collected external messages). Both methods are functionally the same, since collected external events are always executed after internal transitions. However, the method described in (Al-Zoubi et al. 2009b) is tied to DEVS theory and conservative simulation. On the other hand, the technique in Figure 10 uses simulation basics that are easier to adopt by different simulation engines and algorithms. The basic mechanism of advancing a discrete-event simulation is to guarantee that all events occur in correct chronological order; hence, all future events are kept in a list according to their timestamps. In this case, the simulation steps are summarized as follows (Banks et al. 2005). Step 1: Remove imminent event from list. Step 2: Advance simulation clock to imminent event timestamp. Step 3: Execute imminent event. Step 4: Insert any generated events, if any, into the list according to their timestamps. These steps are what RISE does, since it views domains as simulation event lists. RISE equivalent steps are summarized as follows. Step 1: RISE-TM advances simulation clock, since imminent event(s) time is already known. Step 2: Remove imminent events from relevant lists and executes them. This step is performed in domains locally. Step 3: Insert any generated events, if any, into their appropriate lists according to their timestamps. Domains insert events in their local lists while RISE-TM routes all exterior events to other domain lists. Of course, the event list in a RISE domain might be spread over multiple lists over multiple processors, but this information is hidden from the RISE layer.

5.2 Optimistic-based Synchronization

Conservative algorithms avoid violating local causality constraints while optimistic algorithms allow such violations to occur but provide techniques to undo any computation errors. Time-Warp (Fujimoto 2000) is the most well known optimistic algorithm, in which each LP maintains a Local Virtual time (LVT) and it advances "optimistically" without explicit synchronization. To fix errors detected, the LP must rollback to the event before the received *straggler* event. Part of undoing an event computation is to undo other events scheduled by this event on the other processors. Such messages are called *anti-messages*. Time Warp requires a great deal memory throughout the simulation execution. Therefore, an LP must guarantee that rollbacks will not occur before a certain virtual time. In this case, an LP must not receive a positive/negative message before a specific Global Virtual time (GVT). Releasing memory for information older than GVT is performed via a mechanism called *fossil collection*. From the RISE layer viewpoint, simulation is simpler than the conservative RISE, because the actual work is performed internally in each domain where the RISE layer only becomes a common protocol. Therefore, all domains become peers to each other, and communicate directly; hence, the RISE-TM unit is completely removed. Domains can exchange external events in groups or separately, using the format shown in previous section. RISE expects specific domains to keep track of all output messages through RISE layer to other domains. Thus, to undo a previously sent event, an anti-message must be sent, as follows by setting "Type" to "-"(negative sign), as follows:

```
<XEvent Type="-">
    <Time>00:00:01:000</Time>
    <Port>Port-1</Port>
    <Value>9</Value>
    <URI>…/Domain-1</URI>
</XEvent>
```

A Domain GVT (DGVT) is not enough to reclaim memory by fossil collection: a domain may still receive external events from other domains. Therefore, other DGVTs must be considered into RISE GVT calculations.

6 CONCLUSIONS

We presented here the RESTful Interoperability Simulation Environment (RISE) layer, which allows of interoperating heterogeneous simulation models and tools regardless of their underlying technology or algorithms. RISE achieves this goal while hiding internal domains implementations behind only three URIs that can be named and constructed by clients. Thus, RISE does not require existing systems to change their software implementations. We showed that RISE approach provides modelers a simulation experiment framework on how to set the experiment environment, including bridging different models, synchronization simulation according to both conservative and optimistic algorithm techniques, and results retrieval. Thus, RISE presents the complete steps to conduct a simulation experiment.

REFERENCES

Al-Zoubi K., and G. Wainer. 2009. Performing Distributed Simulation with RESTful Web-Services Approach. In *Proceedings of the 2009 Winter Simulation Conference*, ed. M. D. Rossetti, R. R. Hill, B. Johansson, A. Dunkin and R. G. Ingalls. Austin, Texas: Institute of Electrical and Electronics Engineers, Inc.

Al-Zoubi, K., and G. Wainer. Using REST Web-Services Architecture for Distributed Simulation. In *Proceedings of the 2009 ACM/IEEE/SCS Principles of Advanced and Distributed Simulation (PADS)*, 114-121. Lake Placid, New York.

Banks, J., J. S. Carson, B. L. Nelson, and D. M. Nicol. 2005. Discrete-event system simulation. Upper Saddle River, New Jersey: Prentice-Hall, Inc.

Erl T., A. Karmarkar, P. Walmsley, H. Haas, L. Yalcinalp, K. Liu, D. Orchard, A. Tost, and J. Pasley. 2008. Web Service Contract Design and Versioning for SOA. Upper Saddle River, New Jersey: Prentice-Hall, Inc.

Franks J., P. Hallam-Baker , J. Hostetler, S. Lawrence, P. Leach, P. Luotonen, and L. Stewart. 1999. HTTP Authentication: Basic and Digest Access Authentication. RFC 2617. Available via <http://www.ietf.org/rfc/rfc2617.txt> [Accessed October 2008].

Fujimoto, R. M. 2000. Parallel and distribution simulation systems. New York: John Wiley & Sons.

O'Reilly, T. 2005. What Is Web 2.0. Available via < http://oreilly.com/web2/archive/what-is-web-20.html> [Accessed May 2009].

Richardson, L., and S. Ruby. 2007. RESTful Web Services: 1st edition. Sebastopol, California: O'Reilly Media, Inc.

Strassburger, S., T. Schulze, and R. Fujimoto. 2008. Future trends in distributed simulation and distributed virtual environments: results of a peer study, In *Proceedings of the 2008 Winter Simulation Conference*, eds. S. J. Mason, R. R. Hill, L. Mönch, O. Rose, T. Jefferson, J. W. Fowler, 777-785. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Wainer G., K. Al-Zoubi, S.Mittal, J.L. Risco Martín, H. Sarjoughian, and B. P. Zeigler. 2010 (expected publication). An Introduction to DEVS Standardization. Book Chapter, Discrete-Event Modeling and Simulation: Theory and Applications. G. Wainer, P. Mosterman (Editors). CRC Press. Taylor and Francis.

Wainer G., and P. Mosterman. 2010 (expected publication). Discrete-Event Modeling and Simulation: Theory and Applications. CRC Press. Taylor and Francis.

Wainer G., and K. Al-Zoubi. 2010. An Introduction to Distributed Simulation. Chapter 11, Modeling and Simulation Fundamentals: Theoretical Underpinnings and Practical Domains. Banks C., Soklowski J. (Editors). Wiley. New Jersey.

Wainer G. 2009. Discrete-Event Modeling and Simulation: a Practitioner's approach. CRC Press. Taylor and Francis.

AUTHOR BIOGRAPHIES

KHALDOON AL-ZOUBI is a PhD Candidate in Electrical Engineering within the Department of Systems and Computer Engineering in Carleton University, Ottawa, Canada. He is also a senior software analyst and programmer with over 12 years of industry experience occupying a number of seniority and leadership positions. His industry experience spreads over wide range of areas such as embedded software and mobility, air-traffic software management and telecommunications, and security software for explosive and narcotics detections. His email is <kazoubi@connect.carleton.ca>.

GABRIEL WAINER, SMSCS, SMIEEE, received the M.Sc. (1993) and Ph.D. degrees (1998, with highest honors) of the University of Buenos Aires, Argentina, and Université d'Aix-Marseille III, France. In July 2000, he joined the Department of Systems and Computer Engineering, Carleton University (Ottawa, ON, Canada), where he is now an Associate Professor. He has held positions at the Computer Science Department of the University of Buenos Aires, and visiting positions in numerous places, including the University of Arizona, LSIS (CNRS), University of Nice and INRIA Sophia-Antipolis (France). He is author of three books and over 190 research articles, edited four other books, and helped organizing over 90 conferences. He was PI of different research projects and recipient of various awards (NSERC, Precarn, Usenix, CFI, CONICET, ANPCYT, CANARIE, IBM Eclipse Innovation, SCS and others). He is the Special Issues Editor of the Transactions of the SCS, and Associate Editor of the International Journal of Simulation and Process Modeling. He was a member of the Board of Directors of the SCS, and a chair of the DEVS standardization study group (SISO). He is Director of the Ottawa Center of The McLeod Institute of Simulation Sciences and chair of the Ottawa M&SNet, and one of the investigators in Carleton University Centre for advanced Simulation and Visualization (V-Sim). His current research interests are related with modelling methodologies and tools, parallel/distributed simulation and real-time systems. His e-mail and web addresses are <gwainer@sce.carleton.ca> and <www.sce.carleton.ca/faculty/wainer>.